Modeling Transport Processes in Directional Solidification: I. Instabilities and Nonlinear Evolutions in Binary Alloys; II. Direct Parallel Solution of PDE's

by

M. Reza Mehrabi

Submitted to the Department of Chemical Engineering in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Chemical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

© Massachusetts Institute of Technology 1994. All rights reserved.

Author

Department of Chemical Engineering
February 10, 1994

Certified by

Robert A. Brown
Professor
Thesis Supervisor

Accepted by

Robert E. Cohen
Chairman, Committee for Graduate Students

JUN 06 1994
Modeling Transport Processes in Directional Solidification:
I. Instabilities and Nonlinear Evolutions in Binary Alloys;
II. Direct Parallel Solution of PDE’s

by

M. Reza Mehrabi

Submitted to the Department of Chemical Engineering on February 10, 1994, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Chemical Engineering

Abstract

Mathematical modeling has had a major impact on the understanding of solidification processes important in the manufacturing of metal alloys and semiconductor crystals. Almost all models of these complex processes which include heat transfer, phase change and convection in the melt require numerical solution, because of the complexity of the formulation and the nonlinearity of the models. The level of detail in the model and its predictive capability are contingent upon the efficiency of the numerical method for solution. The research in this thesis addresses separately the development of numerical simulation methods and the solution of a particular physical problem important in alloy solidification.

Modeling, linear stability analysis and nonlinear numerical simulations are used in the first portion of this thesis to study the nonlinear structure of morphological and thermosolutal instabilities in a two-dimensional model of directional solidification. Here a planar melt/crystal interface that separates a binary melt from its crystal in unidirectional solidification becomes unstable at a critical concentration \( c_0 = c_{0c} \). Linear stability analysis as a function of the spatial wavelength of the instability leads to two types of transitions: a morphological instability that occurs at small wavelengths and that does not depend on bulk convection, and a thermosolutal instability that relies on a convective instability in the melt.

Finite element analysis of the nonlinear steady and time-periodic states that evolve from the neutral stability curve indicate that the convective instability couples strongly to the interface shape with increasing \( c_0 - c_{0c} \) from zero. Steady states develop highly deformed interface shapes for small values of \( c_0 - c_{0c} \). Oscillatory states show similar behavior. These results indicate the difficulties associated with distinguishing morphological instabilities from instabilities caused by melt convection.

The second portion of this thesis is devoted to developing new methods for lowering the costs of computations for simulations. Lowering the costs expands the limits of practical simulations and allows more thorough studies. These pave the way for
quantitative characterization and ultimately for the design of the production systems. Parallel processing is explored with very promising results for solving the complex mathematical models. A robust method is developed for direct concurrent solution of the sparse linear systems of equations that arise in the context of finite or spectral element discretizations in conjunction with Newton's method for solving the nonlinear set of algebraic equations. Incomplete nested dissection and domain decomposition are used to distribute the domain among the processors and to organize the matrix into sections in which pivoting is performed to stabilize the factorization of indefinite equation sets. The resulting algorithm is highly parallel and memory efficient. The efficient use of sparsity allows the size of problems to increase almost as a linear function of the number of processors. The number of messages and the total volume of messages, which are indications of algorithm's efficiency, are reduced greatly compared to other published research.

The developed method, called concurrent factorization and storage (CFS), performs superbly for complex flow problems involving forced or natural convection. The performance of CFS is evaluated by solving the problems of natural convection in an enclosed cavity, and lid-driven flow in a cavity, on a 32-node Intel iPSC/860 hypercube. These calculations demonstrate that the finite element/Newton method coupled with the CFS algorithm for LU-factorization is a robust and efficient method for solving two-dimensional nonlinear transport problems using MIMD parallel computers. The results show that the speedup associated with the LU-decomposition, which is the most computationally intensive part of the algorithm, is 0.65 for 32-processors and estimated to be 0.5 for 512-processors. Notwithstanding, timing comparisons show that CFS is faster than two other state-of-the-art solvers of transport problems. Moreover, the robustness of the LU-decomposition for solution of the indefinite linear equation sets and the rapid convergence of Newton's iterations for nonlinear problems make the finite-element/Newton method a viable algorithm for solving any problem that the memory of the MIMD computer can accommodate.

Thesis Supervisor: Robert A. Brown
Title: Professor
for Mahin Hamidi, who did not live to see its completion, but
without whom it would never have been begun.

Wind from the east, oh Lapwing of the day,
I send thee to my Lady, though the way
Is far to Saba, where I bid thee fly;
Lest in the dust thy tongueless wings should lie,
Broken with grief, I send thee to thy nest,
    Fidelity.

Or far or near there is no halting-place
Upon Love's road-absent, I see thy face,
And in thine ear my wind-blow sound:
North winds and east waft them where they are bound,
Each morn and eve convos of greeting fair
    I send to thee.

Divan Hafez
Translated by Arthur J. Arberry
Acknowledgments

I would like to thank my advisor, Professor Brown, for his guidance, ideas, and instrumental criticisms. During my few years at MIT, he taught me more than I can express. I also would like to thank him for meticulously reading through this manuscript, providing helpful suggestions for improving it. I am grateful to the members of my committee, Professors Robert C. Armstrong, Anthony T. Patera, and Adel F. Sarofim, for their time, efforts, and fruitful discussions.

I am obliged to Alparslan Öztekin for being an excellent friend as much as for performing the linear stability analysis presented in Chapter 2. I would like to acknowledge D. H. Kim who developed the main part of the code used in nonlinear computations of Chapter 2. I am indebted to Paul Fischer for fruitful discussions and for his performing the computations by Nekton presented in Chapter 5. I would like to acknowledge Einar Rønquist for the use of his spectral element subroutines for calculations presented in Chapter 5. I am grateful to Todd Salamon for performing the computations by the finite element/Newton algorithm with the frontal solver presented in Chapter 5, and would like to acknowledge Steve Burdette for writing the code for the frontal solver, and also David Bornside for writing the rest of the program.

I am much indebted to my teachers at MIT: Robert Balluffi, Daniel Blankschtein, Robert Brown, William Deen, Jonathan Harris, Jack Howard, Ali Nadim, Adel Sarofim, and Nick Trefethen; to my teachers at Princeton: Jay Benziger, Pablo Debenedetti, Chris Floudas, William Russel, Dudley Saville, and especially Roy Jackson and Robert Prud’homme, and to my teachers at Diablo Valley College: Jim Ardini, Don Brunner, and Richard Cooper, who taught me the foundations. I would like to thank my companions and colleagues with whom I talked, laughed, and cried most: Howard Covert, Angelos Kyrilidis, Dimitris Maroudas, and Kostas Tsiveriotis. I would like to thank my other officemates and colleagues for companionship, help, and productive discussions: Narasimha Acharya, Mark Bennett, Jeff Byars, Manuel Cruz, Lars Genieser, Mariano Gurfinkel, Gil Huppert, Hideki Kawai, Tom Kinney,
Chuck Krueger, Tatsuo Mori, Suresh Ramalingam, N. Ramprasad, Talid Sinno, Mike Szady, Yong Xia, Kurt Zhiang, and WeiQun Zhou.

I made many other friends at MIT with whom I shared my joys and frustrations, among them Efstathios Avgoustiniatos, João Ferreira, Joy Mendoza, Marc Moran, Costakis Patrickios, Gordon Smith, and Xinjin Zhao. I have been especially lucky for having Leo Lue as a good friend. I feel honored to know such a bright, wise, and honest individual.

I would like to thank Ms Arline Benford for all her help and good spirits. I also would like to thank Elaine Auiero-Peters, Jean Bueche, Janet Fischer, Nancy Masley, Carol Phillips, and all the other members of the Chemical Engineering staff who have given me help and support.

I would like to acknowledge Intel Corporation, for providing the ivc compiler at no charge, and for technical assistance, MIT Project Athena, for useful tools such as Xess, Matlab, Tecplot, and Maple, and for valuable help from the consultants; Netlib service at Oak Ridge National Laboratory, for providing basic numerical routines; and Paul Thomas, for the use of his Pplot program.

The funding for this project is gratefully acknowledged; it was provided mainly by the Advanced Research Projects Agency and the Microgravity Sciences and Applications Program of the National Aeronautics and Space Administration of the United States Government

***

The help and support of my parents, Mahin Hamidi and Hooshang Mehrabi, have been incalculable and invaluable. This work would not have happened without their endless devotion. I am particularly indebted to my mother whose tireless toil was instrumental in giving me the opportunity to study in the United States. Unfortunately, she passed away shortly before this work was finished. I also would like to thank my brother, Ali Mehrabi, for all he has done for me, and I wish him luck in his studies. The calligraphy on the dedication page is his work.

I am grateful to my wife, love, and companion, Rachel Stern, for bearing my frustration during difficult times. Her vigor is never ending, and my life has not had
a dull moment since we united. I have been, and hope to continue to be, so fortunate as to have her always by my side, and I thank her for all the happiness she has given me. I also would like to thank her for reading this manuscript, and for her suggestions on improving it.

I have shared my happiness and my sorrow with my aunts, Azar and Lili Mehrabi, since I came to the United States. They have been like my second parents. I thank them for all their support. I also thank my uncle and his family, Joe, Shahin, Hoodin, and Vandad Hamidi, who took me in and helped me in many ways when I came to this country. I would like to acknowledge the help, support, and love of many other relatives: Mehri and Sharif Vizheshpour; Cobra, Nasrin, Farrokh, and Hosein Mehrabi; Amir, Ashraf, Shiva, Shideh, Shahin, and Shahmir Hamidi; Ali, Behrouz, Firouz, Kamrouz, Nila, Puya, Sam, Mahnaz, Nancy, and Pari Hashemian; Nersi, Niloofar, and Niimah Pahlavan; Dr. Motahari; Rahmat Shaabani; and Naneh.

My most consistent companion during the composition of this manuscript has been Taos-Maus. It must be admitted that I had my misgivings in adopting the orphan puppy in September, just as I had to begin to put all my energy into writing. But Rachel insisted, and Taos was in the office with me before I knew it. Nonetheless, she brought with her a warmth that all the occupants of the office felt, well almost all. Though she has given us and the furry members of the family, Natanya and especially Anton, plenty of trouble, I do not think I would have finished any sooner without her.

I know that I must have forgotten someone, but I hope whoever that someone is will forgive me.

M. Reza Mehrabi
Cambridge, MA
February 1994
Contents

1 Introduction .................................................. 37
  1.1 Crystal Growth ........................................... 41
    1.1.1 Materials ........................................... 42
    1.1.2 Systems ............................................ 44
  1.2 Transport Processes ...................................... 46
  1.3 Theoretical Design ...................................... 50
    1.3.1 Existing Models .................................... 52
  1.4 Strategies ............................................... 54
  1.5 Outline ................................................ 55

2 Binary Alloy Solidification .................................. 57
  2.1 Introduction ............................................ 62
    2.1.1 Previous Research on Solutal Instabilities ........... 67
    2.1.2 Mechanisms of Instability ........................... 79
    2.1.3 Objectives and Overview ............................. 82
  2.2 Mathematical Model ...................................... 84
    2.2.1 System ............................................ 84
    2.2.2 Field Equations and Boundary Conditions ............ 87
    2.2.3 Convectionless, Unidirectional Solidification ....... 94
    2.2.4 Equations for Linear Stability ..................... 97
  2.3 Numerical Methods ...................................... 101
CONTENTS

4.2.2 Overview ................................................. 205
4.2.3 Graphs and Elimination Trees ......................... 207
4.2.4 Ordering ............................................... 224
4.2.5 Symbolic Factorization ................................... 236
4.2.6 Numerical Factorization .................................. 236
4.2.7 Triangular Solutions ...................................... 243
4.2.8 Scheduling ............................................... 244

4.3 CFS Algorithm .............................................. 247
4.3.1 Overview ............................................... 249
4.3.2 Domain Decomposition by Nested Dissection .......... 251
4.3.3 Preprocessing ........................................... 255
4.3.4 Formulation ............................................. 260
4.3.5 Multivariable PDE's ..................................... 261
4.3.6 LU-Factorization ......................................... 261
4.3.7 Triangular Solutions ...................................... 264

4.4 Summary ................................................... 275

5 Numerical Experiments ...................................... 277
5.1 The Intel iPSC/860 ........................................ 278
  5.1.1 Architecture and Machine Specifications ............... 278
  5.1.2 Performance of the i860 Chip .......................... 279
5.2 Implementation on iPSC/860 ................................ 295
  5.2.1 Program Structure .................................... 295
  5.2.2 Portability ........................................... 298
5.3 Tests with Poisson's Problem ................................ 299
  5.3.1 Message Traffic ...................................... 300
  5.3.2 Bilinear Finite Elements .............................. 303
  5.3.3 Biquadratic Finite Elements ......................... 313
  5.3.4 Spectral Elements .................................... 333
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>Multivariable PDE’s</td>
<td>355</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Natural Convection in a Cavity</td>
<td>358</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Lid-Driven Flow in a Cavity</td>
<td>402</td>
</tr>
<tr>
<td>5.5</td>
<td>Problem Sizes</td>
<td>421</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusions</td>
<td>425</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
<td>427</td>
</tr>
<tr>
<td>6.1</td>
<td>Binary Alloy Solidification</td>
<td>427</td>
</tr>
<tr>
<td>6.2</td>
<td>Parallel Solution of PDE’s</td>
<td>431</td>
</tr>
<tr>
<td>6.3</td>
<td>Future Directions</td>
<td>433</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Convection and Morphology</td>
<td>433</td>
</tr>
<tr>
<td>6.3.2</td>
<td>CFS Code Optimization</td>
<td>433</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Transient Parallel Simulations</td>
<td>434</td>
</tr>
<tr>
<td>6.3.4</td>
<td>General Domain Partitioning</td>
<td>434</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Multiphase Problems</td>
<td>435</td>
</tr>
<tr>
<td>6.3.6</td>
<td>$hpr$-Refinement</td>
<td>437</td>
</tr>
<tr>
<td>6.3.7</td>
<td>Three-Dimensional Simulations</td>
<td>437</td>
</tr>
</tbody>
</table>
List of Figures

1-1 Critical resolved shear stress $\tau$ (10$^7$ dyne/cm$^2$) versus thermal conductivities $k$ (W/cm-K) for various semiconductors [33] .................................. 43

1-2 Various crystal growth configurations. .................................. 45

1-3 Schematic representation of the vertical Bridgman-Stockbarger crystal growth configuration. The important transport processes are indicated in various parts of the system. Only half of the system need be shown because of the symmetry about the centerline. .................................. 47

2-1 Approximate phase diagram for a binary alloy at low solute concentrations [33] .................................. 64

2-2 Exponential concentration profile in the melt in the absence of flow. 64

2-3 Radial and axial segregation of solute as a function of the intensity of melt convection in crystal growth [30] .................................. 66

2-4 Critical concentration as a function of wavenumber as predicted by Mullins and Sekerka’s formula (Eq. (2.11)) for the tin-lead parameters shown in Table 2.1 .................................. 71

2-5 The concentration of tin in lead at the onset of instability during directional solidification at $V = 40 \mu \text{in/s}$ as a function of the spatial frequency $\omega$ of a sinusoidal perturbation. The gravitational acceleration $g = 0$ so that only morphological instability can occur. A shallow minimum occurs at $\omega = 690 \text{ cm}^{-1}$ [65] .................................. 74
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-6</td>
<td>The concentration of tin in lead at the onset of instability during directional solidification at $V = 40 , \mu m/s$ as a function of the spatial frequency $\omega$ of a sinusoidal perturbation. The solid-liquid interface is rigid (planar) so that only convective instability can occur [65].</td>
<td>75</td>
</tr>
<tr>
<td>2-7</td>
<td>The concentration of tin in lead at the onset of instability during directional solidification at $V = 40 , \mu m/s$ as a function of the spatial frequency $\omega$ of a sinusoidal perturbation. Both convective and morphological instabilities can occur; compare with Figs. 2-5 and 2-6. The solid curves show the onset of non-oscillatory instabilities ($\sigma_i = 0$); where the dashed curves show the onset of oscillatory instabilities (the value of $\sigma_i$ is give in the inset) [65].</td>
<td>76</td>
</tr>
<tr>
<td>2-8</td>
<td>The concentration $c_0$ at the onset of instability during directional solidification at 1.0 $\mu m/s$, and a temperature gradient in the liquid of 200 K/cm as a function of the wavenumber of a sinusoidal perturbation for the density change with composition $\beta = 0$, and various values of the gravitational acceleration (in terms of the earth's gravitational acceleration $g_e$) [60].</td>
<td>77</td>
</tr>
<tr>
<td>2-9</td>
<td>Diagram of plane-front solidification.</td>
<td>85</td>
</tr>
<tr>
<td>2-10</td>
<td>Computational domain.</td>
<td>86</td>
</tr>
<tr>
<td>2-11</td>
<td>Modified (or quasi-) Newton's method. A variation of Einstein's summation notation is used for convenience: when the same subscript appears in a term on two or more variables, the summation is implied over that index.</td>
<td>108</td>
</tr>
<tr>
<td>2-12</td>
<td>Computing a solution on a bifurcating family from the basic state solution. $\varepsilon$ is a measure of deviation from the basic state. The failed attempts are shown in dashed lines. One failed attempt results in convergence to base and the other to divergence.</td>
<td>115</td>
</tr>
</tbody>
</table>
2-13 Tin-lead stability diagram for rectangular geometries where the sidewalls are planes of reflective symmetry, based on the work by Coriell and McFadden [60]; $V_0 = 1 \mu/s$ and $G = 200$ K/cm. The number in parentheses associated with each curve denotes the number of waves for the disturbances which are introduced in the computational domain of fixed width as determined by the abscissa. 

2-14 Typical mesh used for tin-lead computations. 

2-15 The first two bifurcating solution families for the tin-lead alloy in a rectangular geometry with shear-free sidewalls; $\lambda = 3$ cm. 

2-16 The first two bifurcating solution families for tin-lead alloy in a rectangular geometry with shear-free sidewalls; $\lambda = 6$ cm. The lower one-wave family connects to the upper two-wave family, as indicated by an open circle (no connection), or a closed circle (connection). 

2-17 The first bifurcating solution family (positive side) for the tin-lead alloy in a rectangular geometry with rigid sidewalls; $\lambda = 3$ cm. 

2-18 The first bifurcating family (negative side) for tin-lead alloy in the cylindrical geometry with a shear-free sidewall; $R = 1.5$ cm. 

2-19 Connectivity of the first two bifurcating families (negative side) for the tin-lead alloy in the cylindrical geometry with rigid sidewalls; $R = 1.5$ cm. 

2-20 The medium and fine meshes used in calculations for the succinonitrile-acetone alloy; the coarse mesh is the one used for tin-lead; see Fig. 2-14. 

2-21 Connectivity of the first two bifurcating families of the succinonitrile-acetone alloy in a rectangular geometry with shear-free sidewalls; $\lambda = 3$ cm. The upper one-wave family connects to the lower two-wave family as indicated by an open circle (no connection), and a closed circle (connection).
2-22 Interaction of the one-wave (1) and two-wave (2) solution families close to the codimension two point. ....................................................... 133

2-23 Neutral stability curves for succinonitrile-acetone alloy in rectangular geometries with shear-free sidewalls. These curves were calculated by monitoring the sign of det(J) in nonlinear calculations rather than by linear stability analysis. The number in parentheses associated with each curve denotes the number of waves for the disturbances which are introduced in the computational domain of fixed width as determined by the abscissa. ....................................................... 134

2-24 Neutral stability curves computed for the succinonitrile-acetone alloy with vanishing heat of solidification....................................................... 138

2-25 Interface shapes, concentration and flow fields computed at $\omega = 25 \text{ cm}^{-1}$. ....................................................... 139

2-26 Neutral stability curves computed for succinonitrile-acetone alloy. ....................................................... 140

2-27 Average interface concentration as a function of dimensionless time for an oscillatory solution. ....................................................... 142

2-28 Interface and flow field in a period for an oscillatory solution. The dashed contours signify counter-clockwise flow. ....................................................... 143

2-29 The onset of instability for succinonitrile-acetone with a capillary parameter of 0.062 cmK. The vertical dashed line shows the range of $c_0$ for which bifurcating solutions are shown in Fig. 2-31. ....................................................... 146

2-30 An enlargement of Fig. 2-29 showing qualitatively the behavior at the onsets of instability which are studied in the bifurcation diagram of Fig. 2-31 for $\omega_0 = 24.5 \text{ cm}^{-1}$. ....................................................... 147

2-31 The Bifurcation diagram for the range of $c_0$ corresponding to the vertical dashed line in Fig. 2-29. ....................................................... 148

3-1 A simplistic pipeline for floating point multiplication [79] ....................................................... 156

3-2 Pipelined execution of an $N$-step process [79] ....................................................... 157

3-3 Schematic of a shared memory system [79] ....................................................... 158
LIST OF FIGURES

3-4 Common interconnecting networks of parallel computers. .......... 160
3-5 An efficient serial solution procedure for solving transport equations in
directional solidification. ............................................ 166
3-6 Schematic of the three-dimensional heat equation in a unit cube. .... 171
3-7 The width of the two subdomains’ shared boundary with the increasing
order of interpolation for finite differences and finite elements. .... 175
3-8 Finite and spectral elements. ........................................ 176
3-9 Two-dimensional Poisson’s problem discretized with bilinear finite el-
ements. ................................................................. 178
3-10 The sparsity of the coefficient matrix for Fig. 3-9 ordered with a band
method. ................................................................. 180
3-11 The sparsity of the coefficient matrix for Fig. 3-9 ordered with nested
dissection. ............................................................ 181
3-12 The sparsity of the coefficient matrix for Fig. 3-9 with band ordering
after Gaussian elimination. Darker regions are the fill-in introduced by
elimination and pivoting within the band. ............................ 182
3-13 The sparsity of the coefficient matrix for Fig. 3-9 ordered with nested
dissection after Gaussian elimination. Darker regions are the fill-in in-
roduced by elimination and pivoting within subdomains and separators. 183
3-14 CGNR algorithm [204] ............................................. 187
3-15 Matrix storage for matrix vector multiplication in parallel. ........ 188
3-16 Algorithm for LU-decomposition with pivoting. .................... 189
3-17 Forward elimination algorithm. .................................... 189
3-18 Back substitution algorithm. ....................................... 189
3-19 Gaussian elimination algorithm organized in tasks. ................ 190
3-20 Gaussian elimination task graph for $N = 5$. ........................ 191
3-21 An algorithm for Gaussian elimination in parallel on a SIMD machine. 192
3-22 Serial frontal decomposition of a band matrix [180] .................. 194
3-23 Domain decomposition of a rectangular domain into two subdomains. 196
4-1 A square domain discretized into four bilinear elements. ............... 209
4-2 The graph for a square domain discretized into four bilinear elements. 209
4-3 A labeled graph based on band ordering for a square domain discretized
into four bilinear elements and the sparsity pattern of the associated
matrix. ................................................................. 211
4-4 A labeled graph based on nested dissection ordering for a square do-
main discretized into four bilinear elements and the sparsity pattern of
the associated matrix. .................................................. 212
4-5 A labeled graph based on minimum degree ordering for a square domain
discretized into four bilinear elements and the sparsity pattern of the
associated matrix. ............................................................. 212
4-6 The adjacent set for $Y = \{x_1, x_2\}$ is $\text{Adj}(Y) = \{x_3, x_7, x_8, x_9\}$ for this
graph and matrix. ......................................................... 213
4-7 The first three stages for $LU$-decomposition for the matrix $A$ in Fig. 4-3
and the corresponding graph. Nonzero entries of the matrix are shown
by $\bigcirc$. The fill-ins are shown by $\otimes$ in the matrix and are represented
by dashed edges in the graph. $k$ is the elimination step or the pivot
number. ................................................................. 216
4-8 The pattern for the filled matrix $F = L + U$ of Fig. 4-7 and its as-
associated graph. Nonzero entries of the matrix are shown by $\bigcirc$. The
fill-ins are shown by $\otimes$ in the matrix and are represented by dashed
edges in the graph. .......................................................... 217
4-9 The elimination tree and the associated filled matrix from which the
tree was obtained for the graph and matrix shown in Fig. 4-8, where
the band ordering was used. The shaded off-diagonal entries in the
filled matrix represent the edges in the tree. .................................. 221
4-10 The elimination tree for the nested dissection ordering shown in Fig. 4-4. 222
4-11 The elimination tree for the minimum degree ordering shown in Fig. 4-5. Although in this case the tree structure is identical to the one for nested dissection (Fig. 4-10), minimum degree ordering usually gives longer, more slender trees than nested dissection. ........................................ 222

4-12 The storage array for a matrix ordered with the band scheme shown in Fig. 4-3 .......................................................... 227

4-13 The envelope for the matrix of Fig. 4-3 .......................................................... 229

4-14 The star-shaped section graph of the arc-length continuation for tracking a nonlinear solution in the parameter space. ........................................ 231

4-15 The minimum profile ordering compared to minimum band ordering of the graph shown in Fig. 4-14 [111] ........................................ 232

4-16 A separator dividing the graph into two section graphs and the associated matrix [111] ......................................................... 234

4-17 Variations of the Gaussian elimination algorithm based on the method of update: (a) row; (b) column; and (c) submatrix, based on the presentation by Heath et al. [134] for Cholesky factorization. .................. 239

4-18 Communication pattern of fan-out algorithm for a model problem [134]. 242

4-19 Communication pattern of fan-in algorithm for a model problem [134]. 242

4-20 The elimination tree wrap-mapped onto the processors [134]. ............. 246

4-21 An example of the subtree-to-subcube mapping for the tree of Fig. 4-20, as given by Heath et al. [134] ............................................. 248

4-22 Domain decomposition by nested dissection of a rectangular two-dimensional domain. A binary reflected Gray code is used for numbering the subdomains to reduce hypercube communication [42] ............................... 249

4-23 Elimination tree for the domain shown in Fig. 4-22 ............................. 253

4-24 The sparsity of the exterior matrix pertinent to the subdomain of processor 9 in the example of Fig. 4-22 .................................. 256
4-25 The sparsity of the exterior matrix pertinent to the subdomain of processor 9 in the example of Fig. 4-22 shown in more detail. The sparsity of the interior matrix also is shown. .................................................. 257

4-26 The adjacent set for the subdomain 6 in the filled graph of the example which is shown in Fig. 4-22 ................................................................. 259

4-27 The adjacent set for the separator 22 in the filled graph of the example which is shown in Fig. 4-22 ................................................................. 259

4-28 The adjacency structure used for computer representation of the graph in Fig. 4-4 which was ordered by nested dissection. ............................ 260

4-29 Representation of a supernode in the matrix. ........................................ 267

4-30 Schematic representation of an $n \times n$ square domain divided using nested dissection into $m \times m$ subdomains among $P$ processors. ........ 268

5-1 A general configuration for Intel iPSC/860 [151] .................................... 280

5-2 Performance of the LINPACK benchmark program with the if77 compiler on the i860 chip; the computation speed as a function of the order of the matrix. ......................................................... 282

5-3 daxpy subroutine from LINPACK. ......................................................... 284

5-4 Performance of the LINPACK benchmark program with the if77 compiler on the i860 chip with the coefficient matrix upper triangular; computation rate as a function of the order of the matrix. .................. 285

5-5 Performance of the LINPACK benchmark program with the if77 compiler on the i860 chip with the coefficient matrix equal to identity; computation rate as a function of the order of the matrix. .................. 286

5-6 $LU$, FE, and BE portions of the LINPACK benchmark program showing the vector operations with vector lengths. ................................. 288

5-7 The timing data and fit with Eq. (5.6) for the speed data shown in Fig. 5-2 ................................................................................................. 290
5-8 Performance of the LINPACK benchmark program with the Kuck BLAS on the i860 chip; the computation speed as a function of the order of the matrix. ........................................ 292

5-9 Performance of the LINPACK benchmark program with the Kuck BLAS on the i860 chip for an upper triangular coefficient matrix; the computation speed as a function of the order of the matrix. ............ 293

5-10 Performance of the LINPACK benchmark program with the Kuck BLAS on the i860 chip for an identity coefficient matrix; the computation speed as a function of the order of the matrix. .......... 294

5-11 Flow diagram of the domain decomposition with nested dissection algorithm as implemented on the Intel hypercube. ....................... 297

5-12 The number of messages for the domain divided among 16 processors as a function of the number of unknowns in one direction compared to the recent work of Mu and Rice [201] ...................... 301

5-13 The volume of messages for the domain divided among 16 processors as a function of the number of unknowns in one direction compared to the recent work of Mu and Rice [201] ...................... 302

5-14 Mapping an element from the global coordinates to local coordinates shown graphically. .................................................. 305

5-15 Speedup for LU-decomposition of Poisson's problem discretized with bilinear finite elements as a function of the number of unknowns. The domain is divided 16 times in all cases. ....................... 311

5-16 Computation times for solution of Poisson's problem discretized with bilinear finite elements as a function of the number of unknowns. The domain is divided 16 times in all cases. ....................... 312

5-17 The computational speed for LU in MFLOPS versus the number of equations for Poisson's problem discretized with biquadratic finite elements on one processor. ............................... 318
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-18</td>
<td>Computational speed for FE in MFLOPS versus the number of equations for Poisson’s problem discretized with biquadratic finite elements on one processor.</td>
<td>319</td>
</tr>
<tr>
<td>5-19</td>
<td>Computational speed for BE in MFLOPS versus the number of equations for Poisson’s problem discretized with biquadratic finite elements on one processor.</td>
<td>320</td>
</tr>
<tr>
<td>5-20</td>
<td>Computational speed for (LU)-decomposition in MFLOPS versus the number of equations for Poisson’s problem discretized with biquadratic finite elements on more than one processor.</td>
<td>321</td>
</tr>
<tr>
<td>5-21</td>
<td>Computation time for (LU)-decomposition as a function of the number of equations for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>322</td>
</tr>
<tr>
<td>5-22</td>
<td>Computation time for FE as a function of the number of equations for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>324</td>
</tr>
<tr>
<td>5-23</td>
<td>Computational rate for FE as a function of the number of equations for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>325</td>
</tr>
<tr>
<td>5-24</td>
<td>Computation time for BE as a function of the number of equations for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>327</td>
</tr>
<tr>
<td>5-25</td>
<td>Computation rates for BE as a function of the number of equations for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>328</td>
</tr>
<tr>
<td>5-26</td>
<td>The speedup for (LU)-factorization as a function of the number of unknowns for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>329</td>
</tr>
<tr>
<td>5-27</td>
<td>The speedup for FE as a function of the number of unknowns for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>330</td>
</tr>
<tr>
<td>5-28</td>
<td>The speedup for BE as a function of the number of unknowns for Poisson’s problem discretized with biquadratic finite elements.</td>
<td>331</td>
</tr>
</tbody>
</table>
5-29 Convergence of solution of Poisson's problem discretized with bilinear finite elements, biquadratic finite elements, and $8 \times 8$ spectral elements as a function of the number of unknowns $N$ where $e \equiv || \hat{u} - u ||_{\infty}$. 335

5-30 Computation rate for $LU$ as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. All operations are included, i.e. those involving zero and nonzero operands. 337

5-31 Computation rate for $LU$-factorization as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. Only operations involving nonzero operands are included. 338

5-32 Computation rate per processor for $LU$-factorization as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. Nonzero operations are included, i.e. those involving nonzero operands. The curve for $P = 1$ nearly coincides with the curve for $P = 2$ and is replotted in the inset for clarity. 341

5-33 Computation times for $LU$-factorization for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns for various numbers of processors. 342

5-34 Computation rate for FE for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes all operations involving zero or nonzero operands. 343

5-35 Computation rate for FE for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes only operations involving nonzero operands. 344

5-36 Computation times for FE as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. 345

5-37 Computation speeds for BE for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes all operations involving zero or nonzero operands. 346
5-38 Computation speeds for BE for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes only operations involving nonzero operands. 347

5-39 Computation time for BE as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. 348

5-40 Speedups for $LU$-factorization for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. 349

5-41 Speedups for $LU$-factorization for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns for small values of $N$ to show the cross-over points, where using more processors is clearly less efficient. 351

5-42 Efficiency for $LU$-decomposition as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. 352

5-43 Speedups for FE as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. 353

5-44 Speedups for BE as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. 354

5-45 Diagram of the model problem of two-dimensional natural convection in an enclosed cavity. 358

5-46 Streamlines and isotherms for the steady solution and real and imaginary parts of the critical eigenvector at the first Hopf bifurcation point, for $Pr = 0.015$, $A = 4$, and a rigid upper surface [270]. 359

5-47 Oscillatory convection shown as instantaneous streamlines eight times during one cycle, for a value of the Grashof number close to threshold, an aspect ratio of 4, a Prandtl number of 0.015, and a rigid upper surface [270]. 360
5-48 Streamlines and isotherms for the steady solution and real and imaginary parts of the critical eigenvector at the first Hopf bifurcation point, for Pr = 0.015, A = 4, and a stress-free upper surface [270] .................................................. 362

5-49 Oscillatory convection shown as instantaneous streamlines eight times during one cycle, for a value of the Grashof number close to threshold, an aspect ratio of 4, a Prandtl number of 0.015, and a shear-free upper surface [270] .................................................. 363

5-50 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.015, Gr = 5,000, A = 1.0. A 4 x 4 mesh of rectangular elements is used. The results were computed on 4 processors. .................................................. 368

5-51 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.015, Gr = 5,000, A = 1.0. An 8 x 8 mesh of rectangular elements is used. The results were computed on 4 processors. .................................................. 369

5-52 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.015, Gr = 5,000, A = 1.0. A 16 x 16 mesh of rectangular elements is used. The results were computed on 4 processors. .................................................. 370

5-53 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.015, Gr = 5,000, A = 1.0. A 32 x 32 mesh of rectangular elements is used. The results were computed on 16 processors. .................................................. 371

5-54 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.0, Gr = 20,000, A = 4.0. An 8 x 2 mesh of rectangular elements is used. The results were computed on 4 processors. .................................................. 372
5-55 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.0$, $Gr = 20,000$, $A = 4.0$. A $16 \times 4$ mesh of rectangular elements is used. The results were computed on 4 processors. ........................................... 373

5-56 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.0$, $Gr = 20,000$, $A = 4.0$. A $32 \times 8$ mesh of rectangular elements is used. The results were computed on 4 processors. ........................................... 374

5-57 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.0$, $Gr = 20,000$, $A = 4.0$. A $64 \times 16$ mesh of rectangular elements is used. The results were computed on 16 processors. ........................................... 375

5-58 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. An $8 \times 2$ mesh of rectangular elements is used. The results were computed on 4 processors. ........................................... 376

5-59 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. A $16 \times 4$ mesh of rectangular elements is used. The results were computed on 4 processors. ........................................... 377

5-60 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. A $32 \times 8$ mesh of rectangular elements is used. The results were computed on 4 processors. ........................................... 378

5-61 The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. A $64 \times 16$ mesh of rectangular elements is used. The results were computed on 16 processors. ........................................... 379
5-62 Convergence of natural convection in an enclosed cavity with Newton iteration. The parameters are Gr = 20,000, Pr = 0.015, and A = 4. The initial guess is the converged solution for Gr = 5,000, Pr = 0.015, and A = 4. .......................... 381

5-63 Maximum values of velocity components, $z \equiv |u|_{max}$ or $|v|_{max}$, as a function of the number of unknowns for the four finite element meshes which are used for natural convection in an enclosed cavity with Gr = 20,000, $\varepsilon = 0.015$, and A = 4. ........................................ 382

5-64 Computation speed for LU-factorization as a function of the number of unknowns for natural convection in an enclosed cavity. Operation count includes only nonzero scalar multipliers in vector operations. . . . 385

5-65 Computation times for LU-factorization as a function of the number of unknowns for the calculation of natural convection in an enclosed cavity. ........................................ 387

5-66 Computation speed for FE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. The operation count involves only the vector operations with nonzero scalar multipliers. 388

5-67 Computation times for FE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. .............. 389

5-68 Computation speed for BE as a function of the number of unknowns for natural convection in an enclosed cavity. The operation count involves only the vector operations with nonzero scalar multipliers. .............. 390

5-69 Computation times for BE as a function of the number of unknowns for computation of natural convection in an enclosed cavity. ............. 391

5-70 Computation times for formulation as a function of the number of unknowns for the computation of natural convection in an enclosed cavity. ........................................ 392
5-71 Speedups for $LU$-factorization as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedup calculation is based only on vector operations involving nonzero scalar multipliers.

5-72 Speedups for FE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. The speedup calculation is based only on vector operations involving nonzero scalar multipliers.

5-73 Speedups for BE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. The speedup calculation is based only on vector operations involving nonzero scalar multipliers.

5-74 Speedups for formulation as a function of the number of unknowns for calculations of natural convection in an enclosed cavity.

5-75 Diagram of the lid-driven flow in a cavity. The bottom wall moves to the right with a constant dimensionless speed of one.

5-76 The velocity and pressure fields for lid-driven flow in a cavity with $\Lambda = 4.0$ and a $21 \times 21$ mesh run on 1 processor.

5-77 The velocity and pressure fields for lid-driven flow in a cavity with $\Lambda = 4.0$ and a $42 \times 20$ mesh run on 2 processors.

5-78 The velocity and pressure fields for lid-driven flow in a cavity with $\Lambda = 4.0$ and a $40 \times 38$ mesh run on 4 processors.

5-79 The velocity and pressure fields for lid-driven flow in a cavity with $\Lambda = 4.0$ and a $76 \times 36$ mesh run on 8 processors.

5-80 The velocity and pressure fields for lid-driven flow in a cavity with $\Lambda = 4.0$ and a $68 \times 68$ mesh run on 16 processors.

5-81 The velocity and pressure fields for lid-driven flow in a cavity with $\Lambda = 4.0$ and a $136 \times 64$ mesh run on 32 processors.
5-82 The computation times for $LU$-factorization as a function of the number of unknowns for lid-driven flow in a cavity with $A = 4$. ... 415

5-83 The computation time for FE as a function of the number of unknowns for lid-driven flow in a cavity with $A = 4$. ... 416

5-84 The computation times for BE as a function of the number of unknowns for lid-driven flow in a cavity with $A = 4$. ... 417

5-85 Computation times for formulation as a function of the number of unknowns for calculation of lid-driven flow in a cavity with $A = 4$. ... 419

5-86 Maximum number of unknowns as a function of number of processors for various levels of interior dissection for Poisson’s problem discretized with biquadratic finite elements. ... 422

5-87 Maximum number of unknowns as a function of number of processors for various levels of interior dissection for lid-driven flow in a cavity. ... 423

5-88 Maximum number of unknowns as a function of number of processors for various levels of interior dissection for natural convection in an enclosed cavity. ... 424
List of Tables

1.1 Current models for vertical Bridgman growth [163] .......................... 53

2.1 Parameters for the tin-lead alloy [60] used for the linear stability calculations presented in Fig. 2-4 .................................................. 70

2.2 Scales used for putting the variables in dimensionless form. ............ 91

2.3 The definition of dimensionless variables as produced by using the scales which are summarized in Table 2.2 ................................. 92

2.4 The definition of dimensionless parameters as produced by using the scales which are summarized in Table 2.2 ................................. 92

2.5 Physical properties used for the tin-lead alloy [60, 66] ..................... 117

2.6 Physical properties used for the succinonitrile-acetone alloy [238, 16] 119

2.7 Physical properties used for the succinonitrile-acetone alloy [63] ...... 135

4.1 Leading-order terms in operation counts for direct solution of a full matrix $A$ of order $N$ and the corresponding band matrix of order $N$ where the bandwidth $\beta \ll N$ [69] ................................. 228

4.2 Leading-order terms in operation counts for direct solution of a band matrix for an $n \times n$ centered, finite difference mesh where the order is $N = n^2$ and the bandwidth $\beta = n \ll N$. ................................. 229

4.3 Leading-order terms for storage requirements and computational complexities of $LU$-factorization for band and nested dissection orderings. 235

5.1 Intel iPSC/860 system specifications [151] ................................. 281
5.2 Maximum computation rate $R$ for different numbers of processors from Fig. 5-20. The corresponding efficiencies $\varepsilon$ also are shown. 318

5.3 The highest values of speedups $S$ for $LU$ and the corresponding efficiencies $\varepsilon$ for Poisson's problem discretized with biquadratic finite elements. 332

5.4 The highest values of speedups for FE and BE for Poisson’s problem discretized with biquadratic finite elements. 332

5.5 Maximum possible computation rate $R$ in MFLOPS for different numbers of processors from Fig. 5-31. 340

5.6 The highest values of speedups for $LU$ and the corresponding efficiencies for Poisson's problem discretized with $8 \times 8$ spectral elements. 350

5.7 The highest values of the speedups for FE and BE for Poisson’s problem discretized with $8 \times 8$ spectral elements. 355

5.8 Critical Grashof number $Gr_c$ for the onset of oscillatory convection for natural convection in a cavity with $A = 4$ [270]. 359

5.9 The variables and the corresponding scales used for putting the variables into dimensionless form. 364

5.10 The definition of dimensionless variables as produced by using the scales which are summarized in Table 5.9. 364

5.11 Maximum values for $v(x, 0.5)$ and $u(A/4, y)$ computed here compared to the values reported in the literature for natural convection in an enclosed cavity. The parameters are $Pr = \{0, 0.015\}$, $Gr = 20,000$, $A = 4.0$. 383

5.12 Maximum possible computation rate $R$ for different number of processors from Fig. 5-64. 385
5.13 Computation times per Newton iteration of the largest mesh possible for computation of natural convection in an enclosed cavity on one to 32 processors. The percentage of the total time consumed are shown in parentheses for each part of the solution. 393

5.14 Speedups and efficiencies for the largest number of unknowns $N$ on $P$ processors for calculation of natural convection in an enclosed cavity. 396

5.15 The highest values of speedups for FE and BE for calculation of natural convection in an enclosed cavity. 399

5.16 Performance comparison of CFS with Nekton for calculation of natural convection in an enclosed cavity with $Gr = 1$, $Pr = 0.015$, and $A = 4$. The calculation by CFS is 6.7 times faster. 403

5.17 The scales used to put the variables into dimensionless form for lid-driven flow in a cavity. $U$ is the dimensional value of the speed of the lid. 406

5.18 The definition of dimensionless variables as produced by using the scales which are summarized in Table 5.17 for governing equations of lid-driven flow in a cavity. 406

5.19 Computation times for one Newton iteration of the largest mesh possible for computation of lid-driven flow in a cavity on one to 32 processors. The percentages of the total time consumed are shown in parentheses for each part of the solution. 418

5.20 Performance comparison with a serial frontal solver on a Cray X-MP. 421
Chapter 1

Introduction

The vitality of the microelectronics community depends on the growth of high-quality, large synthetic crystals for use as substrates in device fabrication. Single crystalline materials are grown from melt with a variety of techniques. The crystals are made of a single component, mainly silicon, or of multiple components, such as the binary alloys CdTe, GaAs, and InP. The process of crystal growth entails slow solidification of the material from melt. The solidification is started from a small high-quality crystal to ensure growth of a single crystal in the correct crystalline direction. In all variations of the crystal growth systems, an overall temperature gradient is imposed to drive solidification. The main components of a crystal growth system are the melt, crystal, container, and the furnace.

In an ideal directional solidification system, the imposed temperature gradient is perfectly perpendicular to the plane of solidification, which is the melt/crystal interface. Lateral temperature gradients are not desirable because they cause thermal stresses which lead to the formation of vacancy and dislocation defects in the crystals. A curved melt/crystal interface is an indication of the existence of lateral temperature gradients when the melt is pure. When the melt is an alloy, however, a curved interface may be an indication of lateral variations in composition, because the melting temperature is a function of composition. The furnace temperature is
designed to increase in the direction opposite to gravity in order to suppress natural convection, which causes deflections in the melt/crystal interface. The rise in temperature, opposite to gravity, causes the density to decrease with increasing height for normal materials. Even when the temperature gradient is unidirectional and no heat is lost laterally, the plane of solidification may become curved due to nonlinear interactions among various modes of transport. Some sources of these interactions are: buoyancy effects of concentration and temperature, latent heat of solidification, and the dependence of the melting temperature on the interface curvature and on the concentration of the solute.

The optimization and design of crystal growth systems is complicated. Experimental studies based on technologically important materials require working in very high-temperature environments, where accurate observation and measurements are difficult and expensive. This thesis explores these issues theoretically because theoretical studies are less costly and reveal the role of transport processes more clearly and easily on the quality of the finished product. However, these studies are complex because of strong interactions among various modes of transport and thermodynamics.

Optimization of all crystal growth configurations, which are described in Section 1.1, involves similar issues. One is the means of controlling the temperature field at the interface, which affects the shape of the melt/crystal interface and the thermal stresses in the crystal. Another issue is the controlling of dopant concentration in the crystal to meet the design specifications for the level and uniformity of concentration. Flow in the melt directly affects both the temperature and concentration distributions. The effects of flow are more pronounced on the concentration field because of low Prandtl and high Schmidt numbers which result from the high thermal diffusivities, low kinematic viscosities, and very low species diffusivities which are characteristic of the semiconductor melts. The coupling of the interface shape to the temperature and concentration fields results in mathematical models that are
posed as complex free-boundary problems with largely varying characteristic length and time scales. The nonlinear interactions among various nodes of transport and the melt/crystal interface result in instabilities which cause convection in the melt and deflection of the planar melt/crystal interface.

Research has been extensive in both the theoretical and in experimental areas of crystal growth. Reviews by Brown [33, 28], and Pimputkar and Ostrach [219] describe the role of transport processes in crystal growth. Moreover, several comprehensive references exist that report experimental results and simple theoretical models; the books by Flemings [98], Brice [26], and Rosenberger [229] are good examples.

The effect of fluid flow on crystal growth and the properties of the crystal has been subject to a great deal of theoretical research [40, 144, 59, 48, 33, 30]. Linear stability analysis has been performed for determining the onset of instabilities of a planar melt/crystal interface to small disturbances [59, 65, 203, 60]. For example, the critical bulk concentration of the solute in binary alloys, at which the planar melt/crystal interface becomes unstable, is computed as a function of the wavenumber of the disturbances. The results of linear stability analysis invariably indicate two types of transitions: morphological instabilities which occur at high wavenumbers and are independent of the convective flow, and convective instabilities which occur at low wavenumbers and rely on natural convection in the melt. These results indicate that the morphological and convective instabilities are loosely coupled. The nonlinear studies of the states that evolve from convective or morphological instabilities have been guided by the general belief of loose coupling between flow and morphology. The systems have been studied either by fully nonlinear computations for the macroscopic transport driven by natural convection [47, 30, 189, 188], or ignoring natural convection, by the computations for the effects of surface energy and the resulting development of cellular microstructure at the interface [260, 16, 171, 256].

The first objective of this thesis is to study the two-dimensional behavior of dilute binary alloys in directional solidification by considering simultaneously both the
thermosolutal convection and the interface morphology. Fluid flow and heat- and mass-transport in the melt are modeled along with the location of the freely moving melt/crystal interface and with heat transport in the solid. The numerical methods developed by Brown and colleague are employed for studying the stability and nonlinear dynamics of directional solidification \cite{163, 30, 2, 48, 46, 75}. Linear stability analysis, as developed by Coriell et al. \cite{59} and by Öztekin and Pearlstein \cite{212, 213}, is used in conjunction with nonlinear analysis in order to determine the limits of stability of the planar melt/crystal interface.

The instabilities and nonlinear dynamics explored are caused by the coupling of the solute with other modes of transport phenomena and with interface morphology. Here, the planar melt/crystal interface becomes unstable at a critical solute concentration: \(c_0 = c_{0c}\). Linear stability analysis is used to determine this point at which the solution with a planar melt/crystal interface becomes unstable in the presence of low-amplitude disturbances. The convection in the melt and interface morphology couples much more strongly than is generally believed, as the results in Chapter 2 will show. The nonlinear steady and time-periodic states that evolve from various modes of instability indicate strong coupling between the flow and interface morphology for small values of \(c_0 - c_{0c}\); hence the difficulties associated with distinguishing morphological instabilities from instabilities caused by melt convection.

Advancements in both hardware and software in recent decades have made the solution of complicated two-dimensional and axisymmetric models for transport in crystal growth systems possible. Vector supercomputers are used for various large-scale simulations, which continue to enlarge both our qualitative and quantitative knowledge about crystal growth phenomena \cite{163, 30, 3, 2, 47, 48, 46}. Nevertheless, the ever-growing demand for more realistic models requires larger and more complicated simulations, which for financial reasons require faster solution of the mathematical models. Furthermore, faster simulations make possible more thorough parametric studies which involve calculations of solutions at many different parame-
1.1 Crystal Growth

Ter values. Thorough parametric studies are time consuming and costly because even though a single solution may be computed quickly, often many solutions need to be calculated.

Parallel processing has received considerable attention in the recent years because the costs are lower for a parallel computer than for a serial computer of similar speed. Moreover, the gap in costs becomes wider as the speed increases. The speed of the fastest parallel computer is larger than the speed of the fastest serial computer simply because a parallel computer can be constructed which employs many processors of the serial computer. Therefore, parallel processors are potentially extremely advantageous, but efficient use of them requires the development of new, efficient algorithms.

The second objective of this thesis is improvement of the existing numerical methods, resulting in a fast and efficient parallel algorithm for direct solution of partial differential equations. Although the method was developed as a vehicle to help crystal growth simulations, it can be generalized to apply to a variety of physicochemically complex problems.

The remainder of this Introduction is organized as follows. Crystal growth systems are described in Section 1.1. A description of important macroscopic transport processes in directional solidification is presented in Section 1.2. Theoretical efforts for understanding the transport processes in directional solidification are summarized in Section 1.3. The strategies followed in this thesis for better understanding of transport processes in directional solidification and for building towards the ultimate goal of theoretical design are laid out in Section 1.4. Finally, an outline of the following chapters is described in Section 1.5.

1.1 Crystal Growth

Growing single crystals of semiconductors and semitransparent oxide materials from melt entails solidifying the molten substance starting with a small single crystalline
Various materials are grown and various techniques are used which are described below. Transport processes play an essential role in the quality of the finished product. For example, convection in the melt distributes the dopants and shapes the melt/crystal interface, and gradients in temperature induce stresses in the crystal which affect defect generation and migration. The reader is referred to Brown [33] for a thorough review on transport phenomena in crystal growth, and to Kim [163] for an introduction to crystal growth processes, materials, and their uses.

Large crystals currently grown from melt for the electronics industry include the classes of semiconductors and semitransparent oxides. To these the class of high $T_c$ superconductors has been added recently, but the growth methods for these materials are not well-established, and the technology has not matured enough for theoretical modeling of the growth processes to be useful.

### 1.1.1 Materials

**Semiconductors**

Silicon is the most popular semiconductor material grown from the melt today. It has better semiconductor properties than germanium, the original material used for transistors and integrated circuits. It is cheap, nontoxic, and after oxygen the most abundant element on earth. Despite these benefits, some compound semiconductors have better electronic and optoelectronic properties. These include gallium arsenide (GaAs) and mercury cadmium telluride (Hg$_{1-x}$Cd$_x$Te). However, high quality crystals of these materials are usually harder to grow because of two reasons. One is the additional problem of concentration nonuniformity in multi-component crystals. The other is the quick generation of material defects. These compounds have lower critical resolved shear stresses (CRSS) and lower thermal conductivities, as shown in Fig. 1-1 [33]. Lower thermal conductivities cause higher temperature gradients and thermal stresses, and lower CRSS's cause quicker dislocation generation with stress. Therefore, the growth of crystals of multicomponent materials involves more complicated
Figure 1-1: Critical resolved shear stress $\tau$ ($10^7$ dyne/cm$^2$) versus thermal conductivities $k$ (W/cm-K) for various semiconductors [33].
transport behavior because of the addition of species transport. Furthermore, the temperature has to be controlled more precisely because vacancies and dislocations are generated at lower values of thermal stress.

**Semitransparent Oxides**

There is a whole relatively new class of optoelectronic materials which have semitransparent properties and are usually oxide mixtures of semiconductor elements. Semitransparent oxide materials are important in optoelectronic applications [141, 13, 120]; examples of these materials include sapphire, bismuth germanium oxide (Bi$_{12}$GeO$_{20}$), and bismuth silicon oxide (Bi$_{12}$SiO$_{20}$). Because of the semitransparent properties of these materials, internal heat radiation plays an important role in addition to other modes of transport phenomena. Heat radiation could be so strong as to change the behavior of the system qualitatively, as discussed below.

**1.1.2 Systems**

The crystal growth configurations are categorized into *confined* and *meniscus-defined* systems [33]. In confined crystal growth systems, including directional solidification systems [98], the melt and the crystal are confined in a container, whereas in the meniscus-defined systems, the melt and the solid are held in contact by the surface tension forces alone. The major crystal growth configurations are Czochralski, floating zone, Bridgman, and gradient freeze. Schematic representations of these configurations are shown in Fig. 1-2. The vertical Bridgman-Stockbarger configuration is an example for the confined category, and the floating zone method is an example for the meniscus-defined category.

In the *Czochralski* method [33, 281, 145], the crucible is charged with the molten material, a seed is lowered in the melt and is slowly pulled out. The crystal pulls the melt out with it where the melt crystallizes. The process is batch-wise and the volume of the melt decreases with time. Crystals of very large diameter are grown with this
Figure 1-2: Various crystal growth configurations.
method; for example, silicon crystals of 8 inches in diameter are not uncommon.

In the floating zone method [218, 202], a polycrystalline rod is melted with a circumferential heater and resolidified into a single crystal. The melt is held in place by surface tension forces, so gravity limits the diameters of crystals that can be grown on earth, unless synthetically generated magnetic forces help counteract gravity such as in the case of radio-frequency induction heating. Also, large radial temperature gradients are unavoidable because the heater is positioned radially around the melt, as shown in Fig. 1-2. The large radial temperature gradients result in curved melt/crystal interfaces and high thermal stresses. Alternatively, the floating zone method eliminates the contaminations introduced by contact with the containers such as crucibles and ampoules.

In vertical Bridgman-Stockbarger method [27, 246], the ampoule is charged with the molten material and the material is crystallized by moving the ampoule in a furnace. In a variation of this method termed gradient freeze, the temperature of the furnace is lowered in time to freeze the molten material. The advantages of the Bridgman method are its suitability for materials of high vapor pressure, and its better control of crystal diameter. Also, a vertical Bridgman system is easier to adjust, and in fact is the only system that could show truly axisymmetric behavior in practice. The confined nature of the Bridgman system does not allow easy expansion of the crystal with temperature which may cause high thermal stresses. A variation of the vertical Bridgman and gradient freeze is the horizontal Bridgman where the container is laid horizontally, and the surface is open to a controlled atmosphere. The directional solidification systems studied here represent idealizations of the vertical Bridgman system.

1.2 Transport Processes

A schematic representation of the vertical Bridgman confined solidification system
Figure 1-3: Schematic representation of the vertical Bridgman-Stockbarger crystal growth configuration. The important transport processes are indicated in various parts of the system. Only half of the system need be shown because of the symmetry about the centerline.

is shown in Fig. 1-3. The system consists of the solid, melt, ampoule, and furnace. Only half of the diagram need be shown because the system is axisymmetric. An ampoule, which is aligned vertically, confines the melt and the crystal. The system is heated along the top section and cooled along the bottom section. The section close to the melt/crystal interface is adiabatic to minimize the radial variations of temperature as much as possible. As the system moves downward with the growth velocity $V_g$ the melt crystallizes.

The simplest picture of a directional solidification process is one without any secondary flows. In the absence of externally applied radial temperature gradients, such a system has no variations in the lateral direction in the temperature, flow or concentration fields, as shown in Fig. 2-9 in Chapter 2. The steady one-dimensional solute transport in this system was analyzed by several authors [33, 255, 98] and is described in Chapter 2. The analysis of the transient one-dimensional directional solidification was performed analytically by Smith et al. [244] for a simplified system
in which heat diffusion is very rapid compared to the solute diffusion; later, numerical schemes were proposed by Bourret et al. [22], and Derby and Brown [75], for systems in which the transport of heat and solute and the interface location do not decouple.

In general, imperfect boundary conditions and instabilities cause secondary flows and lateral variations in the field variables. Therefore, several modes of transport exist simultaneously. Besides heat conduction, which is the basic mode of transport, the melt flows due to natural convection caused and driven by the temperature and concentration nonuniformities. In the melt, species and heat diffuse and are convected by the flow. The melting temperature depends on the concentration, the surface curvature, and the local temperature; therefore, the interface may deflect and become nonplanar. Furthermore, the length scales for interface deflection may range anywhere from ampoule dimensions (measured in centimeters) to microscopic dimensions (measured in microns) because of the largely varying length scales between the surface energy and other field variables. Also, heat transports in the melt and in the solid couple at the interface through the melting temperature and heat fluxes. The main mode of transport in the solid and in the ampoule is heat conduction. Diffusion of species in the solid is usually quite small and can be ignored. However, diffusion of species close to the melt/crystal interface cannot always be ignored. In fact, it is necessary to consider this phenomenon in the models for the study of cellular structure of the melt/crystal interface [16, 171]. Heat is convected, conducted, and radiated to the ampoule from the furnace. These effects usually are combined into an effective Biot number and thus a natural boundary condition is imposed on the surface of the ampoule. Heat radiation may be important in either section depending on the physical properties of the melt, solid, or crystal. Mathematical models for the vertical Bridgman system are discussed by Kim [163] and are summarized in Section 1.3 according to their level of sophistication.

The importance of some transport processes is immediately obvious. Conduction of heat is the essence of growth and cannot be ignored. Thus, the simplest
approximations for growth processes need to include the effects of heat conduction. Concentration field is important in multi-component systems both in its own right and because it affects the stability of the system through coupling with the flow equations and the melting temperature at the interface. These couplings are discussed in more detail in Chapter 2. Flow in the melt is usually significant, and its effects are immediately observed in dopant distribution as striations in the crystal. Solute striations in the crystal, which are fingerprints of flow history, have been observed and indicate flow transitions from steady, laminar convection to time-period and chaotic flow [41]. Curvature of the melt/crystal interface may or may not be significant; the interface could be nearly flat, somewhat curved, or deeply curved; however, as the results in Chapter 2 indicate, the amount of interface deflection may not be predetermined intuitively, or from the results of linear stability analysis. The history and significance of the various modes of transport are discussed thoroughly by Brown [33] and will not be repeated here. The paragraph below contains a brief discussion on the importance of internal heat radiation, which is usually ignored.

Due to lack of data on radiative properties of the materials at high temperatures and molten states, the significance of internal heat radiation in semiconductor crystal growth is not readily apparent. Furthermore, the physical properties are usually strongly dependent on the temperature. For example, the infrared absorption spectrum of solid silicon is a strong function of free carrier concentration and varies over several orders of magnitude [272, 214]. In general, semiconductors of low resistivity have a high concentration of free carriers and are more opaque because of absorption of radiation by free electrons. Crystalline silicon shows semitransparent characteristics in the infrared region at room temperature [272, 214] and becomes more opaque at somewhat higher temperatures, but data near the melting temperature do not exist. High temperatures increase the opacity because they increase the concentration of free carriers. Radiative heat fluxes also increase with temperature as the fourth power. Whether radiative heat transfer deserves a detailed treatment depends not
only on the opacity of the material but also on its temperature. Importance of heat radiation in semitransparent oxide growth is quite clear as the works by Yuřerev and Kolensnikova [276] and Yuferov and Vasil’ev [277] in conjunction with the experimental work by Novak et al. [209] and Vasil’ev et al. [263] demonstrate. Yuřerev and Kolensnikova [276] and Yuferov and Vasil’ev [277] have reviewed the effects of internal radiation in semitransparent crystals grown from melt. They have shown that radiative heat transfer can cause a supercooled region close to the interface and may lead to a morphological instability and formation of cells at the interface, even in a pure melt. They point out that these theoretically predicted morphological instabilities are consistent with experimental observations that the interface boundary is greatly stretched in a semitransparent melt [209, 263]. This behavior is not seen in opaque materials. Neglect of internal heat radiation has resulted partly from the complexity of this mode of transport. Calculations including even simple models of internal radiation are very intensely computational [23, 24, 187]. Because of this difficulty, solution of realistic models including internal radiation is only just becoming feasible.

1.3 Theoretical Design

Theoretical design of crystal growth systems requires solution of mathematical models which are accurate enough to give quantitative predictions. Two types of obstacles make quantitative characterization of the crystal growth systems difficult: lack of data for accurate thermophysical properties, and the limit for feasible simulations with current numerical methods and speed of computers.

Data are either lacking altogether or are not accurate in most instances for thermophysical properties of semiconductor materials. This is due to the practical difficulties caused by high-temperature, hostile environments characteristic of crystal growth systems. In many cases, the thermophysical properties are functions of system variables, such as temperature. In some cases, secondary effects, such as heat radiation, con-
taminate the measurements. For example, thermal conductivity measurements often are polluted by heat radiation effects at high temperatures [5].

The lack of accurate thermophysical properties data is addressed by parametric studies [274, 162] which examine the behavior of a system and the sensitivity of its solutions to the inaccuracies in the parameters. Furthermore, parametric studies help optimize the process by examining the behavior of the system through a range of adjustable operating parameters and thereby choosing the best set.

Moreover, transport processes in crystal growth are quite complicated, and even today's sophisticated models have to omit features which are important for quantitative characterization of crystal growth systems [33]. The existing models cannot give quantitative predictions in most cases even with accurate thermophysical properties data.

Current models account for two-dimensional or axisymmetric behavior of convection and diffusion of heat, mass, and momentum in the melt, diffusion of heat in solid and ampoule, and even for the interface as a free-boundary, as summarized in the following Section and in Table 1.1. These models are valuable vehicles in more accurately explaining the nature of transport processes. In some cases, the existing models have been even semi-quantitative. Because the crystal growth systems are quite complex, even qualitative behavior often is impossible to predict without the solution of very complex mathematical models. The current models have improved considerably and are still evolving. However, extending these two-dimensional and axisymmetric models to realistic simulations requires a great leap forward: the solution of time-dependent, even weakly turbulent, three-dimensional models [33]. Internal radiative heat transport, which is inherently three-dimensional, needs to be included when appropriate, for example for growth of semitransparent oxides.

The persisting demand for ever faster algorithms for simulations necessitates improving the current computational techniques [211] and exploiting the potential computational power of novel computer architectures [96]. Part of this thesis is devoted to
achieving this goal both by improving the solution method and by taking advantage of the power of parallel computing. Faster, more efficient computational methods are useful in two ways. They extend the boundaries of currently feasible simulations, and they make more complete parametric studies practical by lowering the costs of each simulation. Therefore, not only will more complicated models be possible to solve, but also the behavior of the system may be examined more thoroughly.

1.3.1 Existing Models

Table 1.1 summarizes the major models developed to date for simulation of vertical Bridgman-Stockbarger growth [163]. The first column indicates the equations solved, the second column the number of spatial dimensions included and whether the model is steady or transient, and the third column the researchers. A superscript of $A$ indicates an analytical method employed for solution of the model and a superscript of $N$ a numerical method. Details of these models are documented and discussed in the work of Kim [163] and Brown [33], and are not repeated here.

The mathematical models, which have been solved successfully, have evolved in several decades from one-dimensional transient, or two-dimensional steady-state models, with one field equation for energy or solute, to two-dimensional transient models with three field equations for energy, solute, and flow. For solutions of similar accuracy, computational complexity of the models increases exponentially with the number of dimensions and geometrically with the number of field equations. To see these relationships, consider the following two examples. The counterpart for a one-dimensional model with one field variable and $N$ unknowns has $N^2$ unknowns in two-dimensions and $N^3$ unknowns in three-dimensions. Therefore, the number of spatial dimensions is an exponent on the number of unknowns $N$. Adding a second field variable to the model with one field variable and $N$ unknowns increases the number of unknowns to $2N$. Therefore, the number of field variables is a multiplier for the number of unknowns $N$. Since the computational complexity is proportional
### 1.3 Theoretical Design

<table>
<thead>
<tr>
<th>Modes of Transport</th>
<th>Spatiotemporal Dimensions</th>
<th>References</th>
</tr>
</thead>
</table>
| Energy             | 1-D steady-state          | Naumann [205]\(^A\), Ravishankar and Fu [223]\(^A\)  
                     |                           | Jasinski et al. [157, 156]\(^A\)  
|                    | 2-D steady-state          | Chang and Wilcox [49]\(^A\), Sen and Wilcox [241]\(^N\)  
                     |                           | Fu and Wilcox [101]\(^N\), Naumann [206]\(^A\)  
                     |                           | Chin and Carlson [50]\(^N\), Naumann and Lehoczky [207]\(^A\)  
                     |                           | Huang et al. [142]\(^N\), Jasinski and Witt [155]\(^N\)  
                     |                           | Taghavi and Duval [251]\(^A\)  
|                    | 1-D transient             | Clyne [56]\(^N\), Fu and Wilcox [102]\(^N\)  
                     |                           | Sukanek [248, 249]\(^A\)  
|                    | 2-D transient             | Borisov et al. [19]\(^N\)  
| Solute             | 1-D transient             | Pohl [220]\(^A\), Smith et al. [244]\(^A\),  
                     |                           | Hulme [143]\(^A\), Memelink [195]\(^A\),  
                     |                           | Clayton et al. [55]\(^N\), Verhoeven [265]\(^N\),  
                     |                           | Kobayashi [165]\(^A\)  
|                    | 2-D steady-state          | Coriell and Sekerka [64]\(^A\),  
                     |                           | Coriell et al. [58], Carlson et al. [37]\(^N\)  
|                     |                           |  
| Energy Solute      | 1-D transient             | Favier [90]\(^N\), Bourret et al. [22]\(^N\),  
                     |                           | Derby and Brown [75]\(^N\)  
| Flow               | 2-D steady-state          | Chang and Brown [47, 48]\(^N\),  
                     |                           | Müller et al. [191, 190]\(^N\),  
                     |                           | Carlson et al. [58]\(^N\), Arnold et al. [6]\(^N\)  
|                   | 2-D steady-state          | Chang and Brown [46]\(^N\),  
                     |                   | Adornato and Brown [3, 2]\(^N\)  
|                   | 2-D transient             | Brown and Kim [30]\(^N\)  

\(^A\): analytical method; \(^N\): numerical method.

Table 1.1: Current models for vertical Bridgman growth [163].
to $N$ to some power, the above relationships for the number of unknowns apply to
the computational complexity too. The exponential rise of the number of unknowns
with the spatiotemporal dimensions has inhibited realization of three-dimensional
models except in very simplified cases, where only natural convection is considered in
geometries similar to the growth configurations [21, 197, 210, 85].

1.4 Strategies

This thesis has aimed to fulfill two objectives simultaneously. One is the study of
binary alloy solidification with the help of current state-of-the-art techniques and
tools: linear stability analysis, fully nonlinear steady-state and transient calculations,
and bifurcation theory. The second is the development of a parallel method for fast,
efficient solution of intricate, nonlinear mathematical models for physicochemically
complex problems.

Studies on directional solidification (Chapter 2) clarify the nature of nonlinear
states introduced by the solutal field and by its couplings to other transport equa-
tions. The solutal field couples to the energy equation through the dependence of
the melting point on concentration and to the flow equations through the depend-
ence of density on the concentration. The evolution of the nonlinear states by each
of these instabilities is quite different. The behavior of the systems in many cases is
counter intuitive. For example, solution families which evolve from the thermsolutal,
convective instabilities of the convectionless, basic solution develop highly deformed
interfaces, which usually are associated with morphological instabilities.

The parallel method developed here (Chapter 4) and implemented for some two-
dimensional problems (Chapter 5) has several advantages. It reduces the computa-
tional complexity, even in the case of serial computations. Also, it takes advantage
of parallel computing very efficiently, resulting in large savings in solution time and
cost of computations. Moreover, it paves the way for efficient solution of differential
1.5 Outline

Linear stability and nonlinear analysis are presented for directional solidification of binary alloys in Chapter 2. The system is described in detail along with the mathematical model and numerical methods used for linear stability analysis and fully nonlinear steady-state and transient calculations. Results are presented from two possible types of instabilities brought about by the coupling of the concentration field with other field equations; and the linear stability and the nonlinear behavior are analyzed. Implications for practical crystal growth systems are discussed.

An overview of parallel processors is presented in Chapter 3. The available computer architectures and the evolution of parallel processors are discussed briefly. The potentials of parallel processing and the economic incentives also are analyzed. After this introduction, various important issues regarding the solving of partial differential equations in parallel are discussed. These issues include the choice of algorithm, discretization technique, formulation and solution of linear systems, and desirable goals for a good parallel algorithm.

The algorithm devised here, which is an efficient method for concurrent solution of complex models by domain decomposition, nested dissection ordering, and direct solution of the linear equation sets, is described fully in Chapter 4. This algorithm, called concurrent factorization and storage (CFS), is a robust method for solution of elliptic partial differential equations discretized by finite or spectral element methods where the resulting nonlinear algebraic equations sets are solved by Newton's method. A complete background is given for solution of sparse, linear systems of equations in parallel prior to the description of the CFS algorithm.

The results from implementation of CFS on an Intel iPSC/860 hypercube parallel computer are presented in Chapter 5. The performance of the computer is
discussed along with the implementation issues. Results are presented for various test problems including natural convection in an enclosed cavity heated from the sides [232] and for a lid-driven cavity [34, 215]. The performance of the algorithm is analyzed in detail in terms of computation speeds, absolute computation times, speedups and efficiencies. Furthermore, CFS is compared to two state-of-the-art solvers on two platforms: Nekton, a highly efficient parallel Navier-Stokes solver on the Intel iPSC/860 [216, 226, 181, 136], and a highly optimized algorithm based on the frontal solution technique [153, 139, 140] developed by our research group on the Cray X-MP.

The results of this thesis are summarized in Chapter 6 both for binary alloy solidification and for parallel computations. There are many possible directions for future research; these too are briefly discussed in Chapter 6.
Chapter 2

Instabilities and Nonlinear Interactions in Binary Alloy Solidification

During directional solidification of a binary alloy, convection in the melt results from several sources. The most important source of convective flow in the terrestrial growth is the variation of the density due to temperature and composition differences in the sample. Additional flow can be driven by the volume changes which usually accompany phase change; and if any free surfaces, such as gas-liquid and liquid-liquid, are present, surface energy gradient (Marangoni) flows are possible [238].

The effect of fluid flow on crystal growth and the properties of the crystal has been subject to extensive research in the recent years [40, 144, 59, 48, 33, 30]. It is well-known that a destabilizing temperature field, which occurs when the temperature increases along the direction of gravity, or when lateral temperature gradients exist, drives secondary flow in the fluid. Therefore, the Bridgman crystal growth configuration for directional solidification (Fig. 1-2) is designed such that the temperature field is stabilizing and the lateral temperature gradients are minimized, as shown in Fig. 1-2. Lateral gradients in the temperature cannot be avoided in practice, and
heat leakage along the ampoule walls cause horizontal density gradients that lead to natural convection in the melt, as Brown and colleagues have demonstrated through numerical simulations [47, 29, 2]. Natural convection can be completely suppressed in the case of a pure fluid if the temperature field is stabilizing and contains no horizontal temperature gradients. However, in binary alloy solidification, if the rejected solute is lower in density than the solvent, fluid flow also may occur because of thermosolutal instabilities, even if the density of the fluid decreases with height and the temperature field is perfectly stabilizing. The conditions which lead to this instability are more complicated than the mere increase of density with height; these conditions are determined by an analysis of the behavior of the fluid elements which are displaced vertically [59, 65, 189]. The main cause for these thermosolutal instabilities is the different rates at which heat and solute diffuse [259]. Coriell and McFadden [60] have demonstrated that thermosolutal instabilities may occur even if the solute that is rejected is denser than the fluid.

Convection redistributes the heat and the solute in the melt and causes variations in the temperature and in the concentration in the lateral directions. Therefore, the composition of the crystal will be less homogeneous in the presence of convection. Steady natural convection causes lateral variations in the crystal composition and non-steady flow causes longitudinal in addition to lateral variations in composition. Moreover, convective flows cause local variations in the temperature and concentration and therefore modify the local conditions which determine the morphological stability [203]. Therefore, localized cellular microstructure may occur in the system as a result of highly nonlinear couplings between the flow, the temperature, the concentration, and the morphology, as the experimental results by Schaefer and Coriell [238] have demonstrated.

Linear stability analysis has been used to determine the onset of instabilities for vertical directional solidification of binary alloys, in which the plane of solidification is perpendicular to the direction of gravity [203, 65, 59, 60]. Linear stability diagrams
are constructed which determine the critical onset for a parameter, such as inlet concentration, usually as a function of the wavenumber of the instabilities. Generally, the minimum of the neutral stability curves determines the onset of instabilities. For example, convective instability can occur for tin concentrations above 0.00032 weight percent for the growth of lead-tin alloy with a liquid temperature gradient of 200 K/cm and a growth rate of 1 $\mu$m/s [59, 189].

Tiller et al. [255] proposed the simplest model for solutal instabilities, called constitutional supercooling; this model is described in more detail along with other background in Section 2.1.1. Mullins and Sekerka [203] performed linear stability analysis for binary alloy solidification. These authors included the effects of solute and heat transport with the constraint of equilibrium thermodynamics at the interface to analyze the stability of the planar interface to small disturbances. Later, Coriell and coworkers [59, 65] extended these analyses to include the effects of fluid flow, and thus determined conditions for the onset of thermosolutal instabilities. Including the fluid flow did not largely affect the morphological instabilities discovered by Mullins and Sekerka [203] at high wavenumbers; however, the interaction of convection with morphology modified the results for low wavenumbers [59, 65]. Therefore, the high wavenumber ends of the neutral stability curves have been labeled as morphological branches and the low wavenumber parts of these curves as convective branches. Universally, the morphological instabilities indicate onsets for steady convection, whereas the convective instabilities may indicate onsets for either oscillatory or steady convection. These results are discussed in more detail below in Section 2.1.1.

Fully nonlinear simulations have been performed for understanding the role of transport processes in the melt. These investigations are summarized in a review paper by Brown [33]. The highly nonlinear couplings among the velocity, pressure, temperature, concentration, and the free melt/crystal interface result in very complex, formidable free-boundary problems, even for the steady-state cases. Brown and colleagues [163, 30, 2, 48, 46, 75, 16, 171] have pioneered computing fully nonlinear
states accounting for all of these complicated transport behaviors. This field also
has been investigated by many other researchers through weakly nonlinear analy-
sis [159, 158, 160] and nonlinear calculations assuming a planar interface [139, 188].
Part of the nonlinear calculations have been focused on the microstructure of the cellu-
lar structure in morphological instabilities, which are affected strongly by the surface
energy [16, 171]; these have assumed a fixed temperature field and ignored the flow
altogether. Other nonlinear calculations have been used to study the macroscopic
behavior considering fluid flow, heat and mass transfer, and the free melt/crystal in-
terface without the effects of the surface energy. Because the length scales associated
with the morphological instabilities are small, it has been argued that the micro-
scopical details of the melt/crystal interface are unimportant in the overall structure
of the interface [33]. However, the reverse certainly is not true. These instabilities
are affected by the overall flow structure because natural convection changes the con-
centration and temperature near the interface. These local changes result in regions
that are morphologically unstable; therefore, cellular structure may develop. This in
turn modifies the flow field and the macroscopic shape of the interface, as observed
experimentally by Schaefer and Coriell [238].

In the current work, fully nonlinear computations have been performed to study
the evolution of nonlinear modes for morphological and convective instabilities in
binary alloy solidification. Steady-state and transient computations have been per-
formed for the fluid motion, heat and mass transport, and the free melt/crystal inter-
face including the effects of surface energy. Linear stability analysis has been used to
aid the nonlinear simulations and to determine the limits of the stability of the basic
state, which is the simplest state without natural convection and with a nonplanar
melt/crystal interface. The case studies have been performed for the tin-lead and the
succinonitrile-acetone binary systems. These systems have been used widely by other
investigators [59, 60, 16, 121, 72, 171] as testing grounds for both theoretical and
numerical investigations of binary alloy solidification for the reasons discussed in Sec-
tion 2.4. Linear stability and nonlinear evolutions have been studied in rectangular and cylindrical geometries with no-slip and shear-free boundary conditions. The rectangular geometry with shear-free boundary conditions on the sidewalls corresponds to the infinite domain which has been used for linear stability analysis [59, 60, 203, 213]. The infinite domain approximation is valid when the wavelength of the disturbances for instabilities, which correspond to the waves on the melt/crystal interface in the low amplitude limit, are small compared to the ampoule dimensions in a real crystal growth configuration. The cylindrical geometry with rigid walls is similar to the actual vertical Bridgman crystal growth configuration. The cylindrical geometry with rigid walls is a better approximation when the wavelength of the disturbances causing the instability are comparable to the ampoule dimensions.

The rest of this Chapter is organized as follows. The problem is introduced in Section 2.1. The description in Section 2.1 also includes the physical system, modes of instability, mechanisms of instability, and the previous work which has been done in the area. The mathematical model and the basic state for linear stability analysis and nonlinear calculations are described in Section 2.2. The numerical methods for linear stability calculations, for nonlinear steady-state and transient calculations, and for finding the bifurcating solution families and tracking those nonlinear states are laid out in Section 2.3. The results for instabilities which come about because of the dependence of the melting temperature on the concentration field and are driven by unequal temperature gradients at the melt/crystal interface are discussed in Section 2.4. The dependence of density on concentration and temperature cause thermosolutal instabilities; these are analyzed in Section 2.5. Linear stability results and nonlinear steady-state and oscillatory solutions are reported and the interaction of oscillatory and morphological solution families studied. Finally, conclusions are presented for the Chapter in Section 2.6.
2.1 Introduction

In unidirectional solidification of a binary alloy, the molten material is confined in a container, usually called the *ampoule*. Solidification starts from a *seed* and continues in one direction until freezing is complete. Unidirectional solidification is used for materials wherein the object is to grow a single crystal while minimizing grain boundaries and other crystalline defects. This chapter is limited to systems in which the solidification front is perpendicular to gravity, like the vertical Bridgman-Stockbarger [27, 98], and gradient freeze crystal growth configurations, shown in Fig. 1-2.

Directional solidification systems are used for fabricating a wide variety of industrial materials. For example, vertical Bridgman-Stockbarger configuration [27, 98] is used for crystallizing semiconductors, metals, oxide materials, and organic materials. Often, the alloys are binary, and one of the components, the solute, is mixed in small quantities. The systems are heated from above to reduce instabilities caused by differential buoyant forces which in turn are caused by the temperature gradients in the melt. In these systems, the melt is positioned in a cylindrical ampoule above the crystal. The surrounding furnace is designed to impose an axial temperature gradient that drives solidification. Although the imposed temperature gradients are mostly axial, some lateral temperature gradients are inevitable because of heat transfer through the surrounding ampoule; these gradients must be minimized, especially close to the interface, in order to reduce thermal stresses which lead to defect formation in the crystal. The systems are transient because of the batch nature of the operation; however, they can be treated as quasi-steady for long ampoules and short times [47, 46, 163]. The batch nature of the operation results in *batch transients* whose time scales are on the order of the total processing time. However, batch transients are not the only cause for the time-dependent behavior. Oscillatory and chaotic states are always possible because of the nonlinear nature of the directional solidification systems. The time-scale for these fluctuations, called *true transients*, is usually much smaller than the time-scale for batch transients due to low growth
rates which lengthen the latter time-scale. Time-dependent fluctuations in the flow result in solute striations in the crystal which have been observed experimentally [41]. Because of the inherent transient behavior of real systems, a more complete theoretical study of directional solidification requires time-dependent simulations. Brown and Kim [30, 163] extended the previous quasi-steady models by Brown and colleagues [47, 48, 46, 2], and performed time-dependent simulations for the Bridgman system; however, these authors only studied the batch transients of the system. The results presented in this Chapter are for the true transients.

The relationship between the concentration of the solute in the solid and the liquid is approximately linear when the solute concentration and the growth velocity are low so that the interface is near equilibrium. This relationship is expressed as \( c_s = \kappa c \), where \( \kappa \) is called the equilibrium distribution coefficient, \( c \) is the concentration of the solute in the melt, and \( c_s \) is the concentration of the solute in the crystal. This linear approximation is shown schematically on the phase diagram of Fig. 2-1. In the absence of convection, the one-dimensional solutal balance is

\[
D \frac{\partial^2 c}{\partial y^2} + V_s \frac{\partial c}{\partial y} = 0
\]

(2.1)

where \( D \) is the diffusion coefficient, \( y \) is the axial coordinate, and \( V_s \) is the crystal growth rate [33]. The boundary conditions are solute conservation at the interface and bulk concentration far from the interface

\[
-D \left. \frac{\partial c}{\partial y} \right|_{y=0} = V_s (1 - \kappa) c|_{y=0}
\]

(2.2)

\[
c|_{y \to \infty} = c_0
\]

(2.3)

Solution of Eq. (2.1) with the boundary conditions (2.2) and (2.3) gives

\[
c = c_0 \left( 1 + \frac{1 - \kappa}{\kappa} e^{-\frac{V_s y}{D}} \right)
\]

(2.4)
Figure 2-1: Approximate phase diagram for a binary alloy at low solute concentrations [33].

Figure 2-2: Exponential concentration profile in the melt in the absence of flow.
That is, the rejection or incorporation of the solute causes an exponential profile for the concentration in the melt as portrayed in Fig. 2-2. The length scale for the exponential decay is \( l_g = D/V_g \), which is usually small compared to the length of the melt region in crystal growth configurations. For example, the exponential decay length is \( l_g = 1 \) mm if the solute diffusion coefficient is \( D = 10^{-5} \) cm\(^2\)/s and the growth rate is \( V_g = 1 \) \( \mu \)m/s. These parameters are typical for succinonitrile-acetone as used in the experimental system of Lee and Brown [170].

**Axial segregation** is the term used for the axial nonuniformity of the solute concentration in the crystal; it occurs even in the absence of flow, because solute is rejected or incorporated at the interface, and because the volume of the ampoule is finite. Axial segregation is represented through the *effective distribution coefficient* [33], which is the average interface concentration in the crystal divided by the volume-averaged concentration in the melt

\[
k_{\text{eff}} \equiv \frac{\langle c_s \rangle_i}{\langle c \rangle} \tag{2.5}
\]

Buoyancy and ampoule rotation cause convection in the melt which redistributes the concentration field resulting in **radial segregation**. Moreover, convection causes changes in axial segregation. Radial segregation \( \Delta c \) is the lateral fractional variation in the solutal concentration in the crystal. If the diffusion coefficient of the solute is negligible in the crystal, \( \Delta c \) can be written in terms of the concentration of solute at the interface < \( c > i \) as

\[
\Delta c \equiv \frac{c_{\text{max}} - c_{\text{min}}}{\langle c \rangle_i} \tag{2.6}
\]

For the one-dimensional, steady-state model given by Eq. (2.4), \( \Delta c = 0 \) and \( k_{\text{eff}} = 1 \). The radial and axial segregations for different flow regimes are shown schematically in Fig. 2-3 as a function of the intensity of the convection. Small flows result in large axial segregations. An increase in the flow increases the radial segregation without reducing the axial segregation significantly. Further increasing the flow results in good mixing and reduces both the radial and axial segregations. Therefore, some
Figure 2-3: Radial and axial segregation of solute as a function of the intensity of melt convection in crystal growth [30].
flow may be desirable in order to evenly distribute the solute.

The presence of the solutal field further complicates control of the crystal growth by introducing new modes of instability into the system. The temperature field is a stabilizing force in directional solidification because the hot zone appears above the cold zone, and the density decreases inversely with the temperature for a normal material. However, the solutal field may be destabilizing if a heavy solute is rejected at the interface \((\kappa < 1)\), or if a light solute is incorporated at the interface \((\kappa > 1)\). Either case may cause instabilities which provoke double-diffusive flow [259, 59]. The term double-diffusive refers to the density dependence on both temperature and concentration, and the diffusion of heat and species. Furthermore, the concentration field may cause instabilities which appear because of the dependence of the melting temperature on the concentration. These instabilities may occur even if the solutal field is stabilizing. These instabilities were first discovered by Coriell and McFadden [60] and will be discussed below in more detail in Section 2.1.1. Once the solutal field introduces instabilities, the nonlinear couplings among the temperature field, the flow field, and the melt/crystal interface cause a variety of interesting nonlinear dynamics.

### 2.1.1 Previous Research on Solutal Instabilities

Cellular and dendritic patterns in the melt/crystal interface are formed because of interactions of the solute, temperature and flow fields with the interface. Investigations began more than three decades ago to know the criteria which lead to the breakdown of a planar melt/crystal interface during the unidirectional freezing of a dilute binary alloy. The simplest model was constructed by Tiller et al. [255] and is referred to as the constitutional supercooling or the microscale instability of a planar melt/crystal interface. The constitutional supercooling instabilities are the simplest explanation for the formation of cellular structure. The assumptions are: (1) that the melt moves only at the growth velocity \(V_g\); and (2) that the concentration gradient is fixed at \(G\). The exponential concentration profile described by Eq. (2.4) indicates a build up of
solute at the interface in the melt region assuming $\kappa < 1$. This build up of solute lowers the melting temperature, which is approximated as

$$T_m = T_{m0} + m(c - \frac{c_0}{\kappa})$$  \hspace{1cm} (2.7)$$

where $m$ is negative and the surface energy is neglected. If the temperature gradient $G$ is low enough, the melt close to the interface may be supercooled so that a small perturbation will cause solidification. Tiller et al. [255] derived a relationship for the onset of instability by estimating the change in the melting temperature as

$$\frac{dT_m}{dz} = m \frac{dc}{dz}$$  \hspace{1cm} (2.8)$$

where the concentration is given by the one-dimensional solution derived above: Eq. (2.4). The criterion for the stability of the interface is expressed as

$$\frac{G}{V_g} > -m \frac{c_0}{D\kappa} (1 - \kappa)$$  \hspace{1cm} (2.9)$$

The above relationship indicates that lowering the temperature gradient, increasing the growth rate, or increasing the inlet concentration might lead to morphological instabilities. The critical concentration $c_{0c}$ for the onset of instability is given by

$$c_{0c} = \frac{D\kappa G}{V_g (\kappa - 1)m}$$  \hspace{1cm} (2.10)$$

Although the constitutional supercooling model is a good physical explanation for the source of morphological instabilities of the interface, it ignores many important phenomena, such as the effects of surface energy, and of heat and species diffusion on the evolution of the disturbances. Furthermore, Eq. (2.9) does not predict a spatial wavelength for the cellular structure of the unstable disturbances because it is one-dimensional in all respects.

Mullins and Sekerka [203] introduced a rigorous method for the linear stability
2.1 Introduction

analysis of binary alloy solidification by considering species and heat transport and the constraint of equilibrium thermodynamics at the interface. These investigators studied theoretically the time dependence of the amplitude of sinusoidal perturbations introduced at the planar interface. They used the gradients of the thermal and concentration fields which satisfied the perturbed boundary conditions in order to determine the velocity of each element of the interface, thereby finding the unstable wavenumbers for which the amplitude of the disturbances grew, and also the stable wavenumbers for which the amplitude of the disturbances decayed. An expression was derived for the neutral stability of a disturbance, i.e. one that neither grows nor decays with time, as a function of the parameters. This formula determined the onset of the what is called morphological instability because the instabilities affect the interface morphology. Mullins and Sekerka’s neutral stability curve is put into the form of critical concentration as a function of the wavenumber to give

\[
c_{0c} = \frac{\kappa}{\kappa - 1 V_g} \frac{D \left[ 2 T_m \Gamma \omega^2 + (G_s + G) \right] \left[ \omega^* \frac{V_g}{2D} \right]}{2m \left[ \omega^* \frac{V_g}{2D} \right]}
\]

(2.11)

where \( c_{0c} \) is the inlet concentration at which the planar interface becomes unstable, \( \kappa \) is the distribution coefficient, \( D \) is the diffusion coefficient for the solute in the liquid, \( V_g \) is the growth rate, \( T_m \Gamma \) is the capillary parameter, \( \omega \) is the wavenumber of the disturbance, \( m \) is the slope of the liquidus curve, and the other parameters are

\[
G_s \equiv k_s \frac{k}{\bar{k}} G_s \quad (2.12)
\]
\[
G \equiv k \frac{k}{\bar{k}} G \quad (2.13)
\]
\[
\bar{k} \equiv \frac{k + k_s}{2} \quad (2.14)
\]
\[
\omega^* \equiv \frac{V_g}{2D} + \left[ \frac{V_g}{2D} \right]^2 + \omega^2 \]^{1/2} \quad (2.15)

where \( G \) is the temperature gradient at the interface, \( k \) is the thermal conductivity,
<table>
<thead>
<tr>
<th>$V_g$</th>
<th>$1 \times 10^{-4}$ cm/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_m \Gamma$</td>
<td>$9 \times 10^{-6}$ cm.K</td>
</tr>
<tr>
<td>$D$</td>
<td>$2.1 \times 10^{-5}$ cm$^2$/s</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>0.12</td>
</tr>
<tr>
<td>$k$</td>
<td>0.3 W/cm.K</td>
</tr>
<tr>
<td>$k_s$</td>
<td>0.15 W/cm.K</td>
</tr>
<tr>
<td>$\omega$</td>
<td>1-1000 cm$^{-1}$</td>
</tr>
<tr>
<td>$G$</td>
<td>200 K/cm</td>
</tr>
<tr>
<td>$G_s$</td>
<td>400 K/cm</td>
</tr>
<tr>
<td>$m$</td>
<td>-1.6 K/wt%</td>
</tr>
</tbody>
</table>

Table 2.1: Parameters for the tin-lead alloy [60] used for the linear stability calculations presented in Fig. 2-4.

and the subscript $s$ signifies the solid phase.

A typical morphological neutral stability curve is shown in Fig. 2-4 and leads to two immediate observations. The minimum value of the stability curve occurs at high wavenumbers, and the bottom of the curve is very flat. These two characteristics invariably exist for the morphological instability curves for melt/crystal systems [59, 65]. They have important implications described below. A typical crystal growth configuration has a diameter of a few centimeters which corresponds to the very low wavenumbers in Fig. 2-4. Suppose the diameter is $\pi \simeq 3.14$ cm, so that $\omega_0 = 2\pi/\lambda = 2$ cm$^{-1}$. The permitted wavenumbers are $2n$ cm$^{-1}$, where $n = \{1, 2, 3, \ldots\}$, since $\omega_0 = 2$ and all its harmonics are allowed. Increasing the concentration corresponds to moving along the vertical lines in Fig. 2-4 for which $\omega = 2n$. For each fixed value of the spatial wavenumber $\omega$, a family of cellular interfaces evolves from the critical onset for the concentration $c_{0c}(\omega)$. Brown and colleagues have computed and discussed the evolutions, interactions, and dynamics of these solution families [33, 16, 17, 257, 260]. Even though the governing transport equations are linear in the absence of flow, the dependence of the melting temperature on the concentration and on the surface
Figure 2-4: Critical concentration as a function of wavenumber as predicted by Mullins and Sekerka's formula (Eq. (2.11)) for the tin-lead parameters shown in Table 2.1.
energy result in a highly nonlinear behavior for the model. The critical onsets for the bifurcation points are clustered in a small range of $c_0$ because the bottom of the morphological neutral stability curves are flat due to low surface energies. Because a system of fixed width admits the Fourier modes and its harmonics, many modes become unstable for small increases in $c_0$ near the flat bottom of the morphological neutral stability curve. Because many solution families bifurcate in a small range of $c_0$, they strongly interact.

While morphological instabilities in the melt/crystal interface are complex in their own right, they can still interact with other hydrodynamic instabilities due to the coupling of solute, flow, and temperature fields. Coriell et al. [59] first realized this and accordingly performed a linear stability analysis that allowed for morphological instability of a planar melt/crystal interface in the presence of thermosolutal convection in the melt. Coriell et al. [59] identified two modes of instability which were distinguishable by the spatial wavenumber for the disturbance to the interface. Morphological modes were found at high wavenumbers and corresponded to the interfacial instability first discovered by Mullins and Sekerka [203]. Introduction of flow did not modify the neutral stability curves for the high wavenumbers. However, the low wavenumber end of the morphological curve was modified; therefore, this end was called the convective branch. Convective modes were caused by thermosolutal instability in the density profile in the diffusion layer. Coriell et al. [59] discovered that convective instabilities might be oscillatory or steady, whereas morphological instabilities were invariably steady.

The work of Coriell and Sekerka [65] illustrates the reasons for labeling the various branches of the neutral stability curves. These authors studied the effects of forced and natural convection on morphological instability. They presented results for the very simplified case of the stagnant film model in which a thin layer of fluid near the interface is assumed to be stationary while the rest of the fluid is well-mixed. They also presented results for general two-dimensional flow in forced convection. Further-
more, they derived stability results for thermosolutal convection. They presented plots for a morphological instability case for which $g = 0$, for a convective instability case for which the interface is rigid, and for a general case including both natural convection and morphological deformation. Figures 2-5 to 2-7, which are the work of Coriell and Sekerka, show onsets for morphological, convective, and morphological plus convective instabilities, respectively. The morphological instability curve (Fig. 2-5) shows a shallow minimum at high wavenumbers $\omega = 690 \text{ cm}^{-1}$. The convective instability curve (Fig. 2-6) shows a sharper minimum which occurs at a much lower wavenumber $\omega = 45 \text{ cm}^{-1}$; furthermore, this curve rises sharply for higher wavenumbers. Combining both the convection and morphology (Fig. 2-7) does not modify the morphological instability curve at high wavenumbers; however, it does affect the convective instability at low wavenumbers. The convective instability interacts with morphological instability which results in steady- and oscillatory-convective instabilities.

The original work of Coriell et al. [59] did not separate the instabilities caused by density variations with concentration, from the ones caused by the dependence of the melting temperature on the concentration. Coriell and McFadden [60] later did study the instabilities introduced solely by variations of the melting point of the solution with the concentration of the solute. These investigators reported linear stability results for directional solidification of a tin containing lead alloy for which the solutal and temperature fields are stabilizing. They showed that the minima of the neutral stability curves occurred at much smaller wavenumbers than the minima of the purely morphological neutral stability. The corresponding large wavelengths are several centimeters which are of the same order as the ampoule diameter in an actual crystal growth configuration. A typical neutral stability diagram computed by Coriell and McFadden is shown in Fig. 2-8.

Relevant research abounds for linear stability of binary alloy solidification and weakly nonlinear analysis of the evolving states. Also, work has been done on non-
Figure 2-5: The concentration of tin in lead at the onset of instability during directional solidification at \( V = 40 \ \mu m/s \) as a function of the spatial frequency \( \omega \) of a sinusoidal perturbation. The gravitational acceleration \( g = 0 \) so that only morphological instability can occur. A shallow minimum occurs at \( \omega = 690 \ \text{cm}^{-1} \) [65].
Figure 2-6: The concentration of tin in lead at the onset of instability during directional solidification at $V = 40 \ \mu\text{m/s}$ as a function of the spatial frequency $\omega$ of a sinusoidal perturbation. The solid-liquid interface is rigid (planar) so that only convective instability can occur [65].
Figure 2-7: The concentration of tin in lead at the onset of instability during directional solidification at $V = 40 \, \mu m/s$ as a function of the spatial frequency $\omega$ of a sinusoidal perturbation. Both convective and morphological instabilities can occur; compare with Figs. 2-5 and 2-6. The solid curves show the onset of non-oscillatory instabilities ($\sigma_i = 0$); where the dashed curves show the onset of oscillatory instabilities (the value of $\sigma_i$ is give in the inset) [65].
2.1 Introduction

Figure 2-8: The concentration $c_0$ at the onset of instability during directional solidification at 1.0 $\mu$m/s, and a temperature gradient in the liquid of 200 K/cm as a function of the wavenumber of a sinusoidal perturbation for the density change with composition $\beta = 3$, and various values of the gravitational acceleration (in terms of the earth's gravitational acceleration $g_e$) [60].
linear calculations with simplifying assumptions such as a flat interface, in addition to the work of Brown and colleagues mentioned above. Some of these studies are briefly described below, first for the linear stability analysis, and then for the weakly nonlinear analysis and nonlinear computations. The discussion below is by no means an exhaustive account of the existing literature on directional solidification of binary alloys. Such a review would require a separate chapter. The reader is referred to the review article by Glicksman et al. [121] for more information on the subject.

Coriell et al. [62] studied the effects of forced Couette flow on coupled convective and morphological instabilities of binary alloy solidification. These investigators found that forced convection somewhat suppressed the onset of morphological instability, greatly suppressed the onset of thermosolutal convection, and decoupled the morphological and convective instabilities to some degree. Furthermore, they found that the onset of instabilities were oscillatory and corresponded to traveling waves. Hurle et al. [147] showed that the equations governing melt stability resemble those for the Bénard problem with permeable boundaries. The authors calculated critical conditions for the onset of stationary convection for a planar interface. Guérin et al. [129] studied the effect of rigid sidewalls and found that they had a stabilizing effect. Guérin et al. [130] calculated some results for the onset of instability of binary alloy solidification in a cylinder. They calculated the onset of instabilities for both axisymmetric and selected three-dimensional modes. They also indicated that the cylindrical configuration was more stable than the two-dimensional rectangular system. Öztékin and Pearlstein [212] studied the effects of non-monotonic dependence of the density on temperature. These authors found that for some combination of operating parameters there is a value of $c_0$ below which plane-front solidification is unstable for any growth velocity; conversely, there is a growth velocity beyond which plane-front solidification is unstable for any $c_0$.

McFadden et al. [189] solved the time-dependent nonlinear differential equations for flow, temperature, and concentration in a rectangular geometry, but they as-
2.1 Introduction

sumed a planar interface. Jenkins [159] extended the linear stability analysis and included weakly nonlinear analysis of the steady and oscillatory solutal convection in the flow to determine the planform of such flows: in these studies, it was found that the steady convection preferred a hexagonal planform and the oscillatory convection preferred two-dimensional bands. Also, Jenkins [158] performed weakly nonlinear analysis where the morphological and convective instabilities occurred simultaneously in order to study the interactions of the two modes, and concluded that it was weak. McFadden and Coriell [188] performed nonlinear calculations to study the effect of thermosolutal convection on solute segregation. These authors solved time-dependent flow, temperature, and concentration equations with the assumption of a flat (planar) interface. Jenkins [160] presented linear stability results and did some weakly nonlinear analysis on interaction of convective and morphological modes through perturbation analysis.

2.1.2 Mechanisms of Instability

The concentration field couples with the temperature and flow fields and to the shape of the melt/crystal interface through the dependence of the density on concentration [59], and through the dependence of the melting temperature on concentration [203, 60, 65]. These two mechanisms for instability have quite different origins and cause nonlinear states that are qualitatively different. The remainder of this Chapter will be spent analyzing these mechanisms.

The most important macroscopic transport phenomena are melt flow, transport of species in the r·elt, and transport of energy in the melt and solid. The flow is described by the incompressible Navier-Stokes equations with the Oberbeck-Boissinesq approximation [44, 116]. This approximation is a Taylor Series expansion of the density for the term that represents the buoyancy force as a function of temperature and concentration about a reference state \((c_0, T_0)\); only the linear terms are retained. The
fractional change in density with this approximation is

$$\frac{\Delta \rho}{\rho_0} \approx -\beta_T (T - T_0) - \beta_c (c - c_0)$$  \hspace{1cm} (2.16)

where the thermal ($\beta_T$) and solutal ($\beta_c$) expansion coefficients are defined as

$$\beta_T \equiv -\frac{1}{\rho} \frac{\partial \rho}{\partial T} \bigg|_{T=T_0}$$  \hspace{1cm} (2.17)

$$\beta_c \equiv -\frac{1}{\rho} \frac{\partial \rho}{\partial c} \bigg|_{c=c_0}$$  \hspace{1cm} (2.18)

The change in density enters the Navier-Stokes equations as a body force

$$\mathbf{F}_b = -\rho \mathbf{g}$$  \hspace{1cm} (2.19)

where $\mathbf{g}$ is the acceleration of gravity. The above approximation for buoyant forces causes the concentration field to couple with the flow field. This coupling gives rise to thermosolutal instabilities [59, 65]. For example, if the distribution coefficient is $0 < \kappa < 1$, solute is rejected upon solidification. As a result, solute would accumulate close to the interface because $c_s = \kappa c$ at the interface. Furthermore, if the coefficient of solutal expansion satisfies $\beta_c > 0$, the density decreases as the concentration increases, as expressed by Eq. (2.16). The higher concentration of the solute reduces the density close to the interface. The lighter fluid has the tendency to rise and leads to a hydrodynamic instability. The solutal field is destabilizing, so the basic state which is associated with the flat interface may become unstable for some combination of inlet concentration, growth rate, and thermophysical parameters. Normally the melt expands with increasing temperature; that is, $\beta_T > 0$ for most semiconductor systems. This means that the density decreases with increasing temperature. Therefore, the temperature field is always stabilizing because the system is heated from the top and cooled from the bottom. The instabilities which are driven by the dependence of the density on the concentration and temperature are called ther-
mosolutal or double-diffusive convective instabilities [259]. An interesting fact is that thermosolutal instabilities may occur when a light solute is rejected even if the overall density decreases with height; this is because of the different rates of diffusion of heat and species as explained by Coriell et al. [59].

The concentration field couples with the melt/crystal interface shape by virtue of the melting temperature. In a binary alloy, the melting temperature is a function of concentration as well as surface curvature. Usually, the Gibbs-Thomson relation [273] is used for approximating the dependence of the melting temperature on concentration and on surface curvature as

\[ T_m = T_{m0} + m(c - \frac{c_0}{\kappa}) - T_m \Gamma H \]

where \( T_{m0} \) is the melting temperature of a flat interface at the concentration \( c = c_0/\kappa \), \( m \) is the slope of the liquidus curve, \( T_m \Gamma \) is the capillary parameter, and \( H \) is the mean curvature of the interface. The dependence of the melting temperature on concentration causes instabilities when inequalities exist in temperature gradients at the melt/crystal interface. These inequalities are caused by unequal thermal conductivities for the solid and melt and by latent heat release. The basic, one-dimensional state loses its stability for some combination of parameters including the inlet concentration and growth rate. The origin of this instability is completely different from the thermosolutal instability discussed above; it exists even if the solutal field is stabilizing; for example if \( 0 < \kappa < 1 \) and \( \beta_c \leq 0 \), as shown by Coriell and McFadden [60]. These instabilities, called convective supercooling instabilities, appear in the following manner.

The lateral fluctuations in concentration field cause deviations in the planar shape of the melt/crystal interface which in turn cause lateral variations in the temperature gradients close to the interface. The lateral variations in the temperature gradient drive convective flow. This instability is morphological or interfacial in the sense that it starts at the interface. However, convection plays an important role in manifest-
ing the instability. Lateral disturbances in the concentration field cause the melting temperature to change and the interface to deviate from the planar shape. If an inequality exists in temperature gradients in the melt and solid, the non-planar interface results in lateral variations in the temperature gradient which drive flow. The flow may be may be stabilizing depending on the ratio of the thermal conductivities, as demonstrated by Coriell and McFadden [60]. These instabilities are not simply the constitutional supercooling discovered by Tiller et al. [255] or the morphological instabilities discovered by Mullins and Sekerka [203] because convection plays an important role. Even though the temperature field is stabilizing, including the effects of temperature on buoyancy may destabilize the system because of the coupling of temperature, concentration, and flow fields with the interface.

2.1.3 Objectives and Overview

The calculations presented here are meant to facilitate the study of the instabilities caused by the solutal field, and to examine the nonlinear behavior of solutions through linear stability, nonlinear computations, and bifurcation analysis.

Fully nonlinear bifurcation analysis and studies of the evolution of nonlinear states are lacking in the literature which otherwise includes all modes of transport and the melt/crystal interface as a free boundary in unidirectional solidification of a binary alloy. Previous models, which included flow, energy, and solute transport, treated the interface as planar [189, 188]. This is an inadequate assumption because the interface can deviate drastically from the planar state. Thus, treating the interface as flat can qualitatively change the behavior of the solutions. The directional solidification of dilute and non-dilute alloys have been studied through fully nonlinear simulations which account for the melt/crystal interface as a free surface but not for the surface energy of the interface [30, 47, 46, 22, 75, 2, 3, 29]; when the surface energy was considered, the convective flow field was ignored and the temperature field was assumed to be fixed [16, 17, 257, 258, 261].
2.1 Introduction

The results in this Chapter reveal dynamics of flow, temperature, and solutal fields and moving melt/crystal interface, wherein the surface energy is accounted for. A simple and efficient linear stability theory is used for determining the onsets of thermosolutal instability which mark the bifurcation points from the one-dimensional state with a flat interface. Families of solutions with secondary convective flows and nonplanar melt/crystal interfaces bifurcate at these onset points. It will be shown that the morphology of the melt/crystal interface and the convective flow can be highly coupled for both morphological and convective thermosolutal instabilities. This is contrary to the general belief that the interactions of convective and morphological modes are weak. Furthermore, truly transient solutions have been computed. The transient and steady solutions exhibit large couplings between the flow and morphology.

The case studies are for the organic alloy succinonitrile containing acetone and for the metallic alloy tin containing lead. The lead-tin alloy has been used extensively as a basis for linear stability analysis and experimental studies [59, 62, 129, 130, 266]. The low melting point of this alloy and the fact that its physical properties are well-known simplify experimental studies. The concentration of tin is lower than lead, and the tin which is rejected into the melt at the melt/crystal interface may destabilize the system [59]. The reverse system (tin containing lead) was studied by Coriell and McFadden [60] because the lead that is rejected into the melt at the interface is heavier, and is therefore stabilizing. Thus, the only possible instabilities are convective supercooling, as described in Section 2.1.2. The transparent succinonitrile-acetone alloy is also studied. This alloy, and the similar succinonitrile-ethanol alloy, have been used for theoretical and experimental studies in thermosolutal convection by many researchers [238, 238, 61, 63, 16, 170]. The popularity of this alloy is due in part to the transparency of succinonitrile, which allows observation of the convective flow by tracking the motion of small, neutrally buoyant particles [238, 237]. Furthermore, succinonitrile solidifies with an unfaceted melt/crystal interface and has well-known
physical properties.

The mathematical model is discussed in detail in Section 2.2 and the numerical methods in Section 2.3. The mathematical model includes solute and heat diffusions, thermosolutal convection in the melt, and heat diffusion in the solid. The interface is a free boundary described by the condition for thermal equilibrium and by the Gibbs-Thomson equation. The numerical methods are spectral Galerkin with modal expansion for linear stability calculations and Galerkin finite elements for fully nonlinear steady-state and transient calculations. Implicit time-stepping schemes are used for temporal discretization of the transient model. Newton's method is used to solve the nonlinear sets of equations that arise both in the steady-state and the transient calculations.

2.2 Mathematical Model

2.2.1 System

A diagram of the physical system is shown in Fig. 2-9. Without flow or morphological instability, the liquid/crystal interface is planar and no lateral variations occur in either the temperature or the concentration fields. This state is the basic state for linear stability analysis and is described quantitatively in Section 2.2.3. The melt is above the solid and the melt and the solid are joined by a flat interface which is infinite in both the \(x\)- and \(z\)-directions; the system is oriented such that gravity is perpendicular to the planar interface and is pointed toward the solid. The temperature increases in the \(+y\)-direction and decreases in the \(-y\)-direction. The coordinate frame of reference is attached to the interface. The concentration reaches the bulk value \(c_0\) far from the interface in either direction. There is a net downward growth velocity with a magnitude of \(V_g\).

The system may be subject to two-dimensional instabilities which are characterized by a wavelength \(\lambda\) or a wavenumber \(\omega \equiv 2\pi/\lambda\). The disturbance occurs in
an arbitrary direction in the \(xz\)-plane, and the wavenumber may have both \(x\)- and \(z\)-components: \(\omega_x\) and \(\omega_z\), where \(\omega = \sqrt{\omega_x^2 + \omega_z^2}\). An additional wavenumber is required along the vertical axis to characterize three-dimensional disturbances. Only two-dimensional disturbances are considered in the analysis presented here. Nonlinear states, which evolve from the point at which the state with a flat melt/crystal interface becomes unstable exhibit a wavy interface, convective flow with cellular character, and wavy temperature and concentration contours. The wavelength of the interface shape is the same as the wavelength of the disturbance which causes that instability; also, the number of cells in the flow field is twice the number of waves that appear in the system.

The system considered here is idealized such that no imposed lateral temperature gradients exist except where the effects of heat leakage through the boundaries is examined; heat leakage causes radial temperature gradients and appears as an imper-
fection in the system which ruptures the bifurcation points calculated here [131, 274].

The mathematical or computational domain, where the mathematical model is formulated, is two-dimensional (Fig. 2-10). To simplify calculations, the direction of the $x$-axis in Fig. 2-10 coincides with the direction of the disturbance, so $\omega_z = 0$ and $\omega = \omega_x$. The width of the domain is fixed and limits the permitted wavenumbers of the disturbances. Furthermore, the two-dimensional solutions are assumed to have a vertical plane of symmetry in the case of rectangular geometries, and a vertical center-line of symmetry in the case of cylindrical geometries. This assumption follows from the symmetry of the boundary conditions about the mid-plane for rectangular geometries and about the center-line for cylindrical geometries. Therefore, the computational domain in Fig. 2-10 is limited to half of the domain to reduce the number of unknowns in the computations. The width of the computational domain $W$ is half of the size of the wavelength $\lambda$ because of the reflective symmetry of the waves about the mid-plane, or the center-line in the case of cylindrical geometries. Only whole
2.2 Mathematical Model

waves are allowed in the whole domain, which is twice as wide as the computational domain. These waves include the principle wave and its harmonics: \( \{ \omega_0, 2\omega_0, 3\omega_0, \ldots \} \). The height of the computational domain \( H \) is chosen such that \( H \gg D/V_g \). More precisely, the value of \( H \) used in computations is about \( 10D/V_g \). Increasing \( H \) beyond this value does not affect the linear stability results; however, the evolving nonlinear states are affected.

2.2.2 Field Equations and Boundary Conditions

The model for thermosolutal convection in a melt/crystal system is written for a finite, two-dimensional domain, which is the same as the computational domain shown in Fig. 2-10. For the sake of consistency, the term computational domain will be used in this Section to refer to the domain for which the field equations and boundary conditions are formulated. In the following formulation, the geometrical shape of the computational domain is either rectangular or cylindrical, and the sidewall is either rigid (no-slip) or shear-free. The rectangular geometry with shear-free sidewalls corresponds to the infinite physical domain for which linear stability results often are computed [203, 59, 60], and the cylindrical geometry with rigid sidewall corresponds to the geometry of the vertical Bridgman crystal growth configuration. The rectangular geometry with shear-free sidewalls is a more reasonable framework for analyzing the two-dimensional instabilities whose wavelengths are much smaller than the ampoule dimensions in the vertical Bridgman crystal growth configuration. The physical domain is assumed to be long so that the top of the computational domain is a continuous inlet for fresh melt, and the bottom of the computational domain is a continuous outlet for the crystal. The densities of the solid and melt are assumed to be equal; i.e. the crystal will not contract or expand upon formation. The interface is represented as \( y_i = h(x, t) \), and the mesh is fixed to the interface. The time derivatives are zero for steady-state calculations, the interface remains stationary, and the crystal is grown at a continuous rate \( V_g \). The situation is similar for the
time-dependent case, except that the field variables and the interface position will change in time. A diagram of the computational domain is shown in Fig. 2-10.

The flow in the melt is governed by incompressible Boussinesq equations

\[
\nabla \cdot \mathbf{v} = 0
\]

\[
\rho \left( \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla P + \mu \nabla^2 \mathbf{v} + \rho g \left[ \beta_T (T - T_0) + \beta_c (c - c_0) \right]
\]

where \( \mathbf{v} = (u, v) \) is the velocity vector, \( P \) is the pressure, \( T \) is the temperature, \( c \) is the concentration, \( \rho \) is the density, \( \mu \) is the viscosity, \( g \) is the acceleration of the gravity, \( \beta_T \) is the coefficient of thermal expansion, \( \beta_c \) is the coefficient of solutal expansion, and the subscript 0 refers to some reference values for the temperature and concentration. The reference value \( c_0 \) is the inlet concentration and \( T_0 \) is the maximum imposed temperature. The boundary conditions for the velocity are no-slip at the interface and at the inlet

\[
u = 0, \quad v = -V_g \quad \text{on} \quad \partial D_i \cup \partial D_t
\]

and shear-free at the mid-plane, or center-line in the case of cylindrical geometry,

\[
u = 0, \quad \frac{\partial v}{\partial x} = 0 \quad \text{on} \quad \partial D_c
\]

The sidewall is either rigid

\[
u = 0, \quad v = -V_g \quad \text{on} \quad \partial D_w
\]

or shear-free

\[
u = 0, \quad \frac{\partial v}{\partial x} = 0 \quad \text{on} \quad \partial D_w
\]
2.2 Mathematical Model

The energy equation in the melt is

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \alpha \nabla^2 T$$  \hfill (2.27)

where $\alpha \equiv \rho/kC_p$ is the thermal diffusivity, $k$ is the thermal conductivity, and $C_p$ is the heat capacity. Only energy transport is assumed in the solid; it is described by

$$\frac{\partial T_s}{\partial t} - V_g \mathbf{e}_y \cdot \nabla T_s = \alpha_s \nabla^2 T_s$$  \hfill (2.28)

where the subscript $s$ stands for solid. The reflective symmetry for the temperature at the mid-plane or center-line translates to

$$\frac{\partial T}{\partial x} = \frac{\partial T_s}{\partial x} = 0 \quad \text{on} \quad \partial D_e$$  \hfill (2.29)

The infinite physical system represented by the rectangular geometry and shear-free sidewall has reflective symmetry about the sidewall; mathematically

$$\frac{\partial T}{\partial x} = \frac{\partial T_s}{\partial x} = 0 \quad \text{on} \quad \partial D_w$$  \hfill (2.30)

The above condition at the sidewall also represents a physical system of finite width which has adiabatic sidewalls. Heat leakages through the sidewall are approximated by Newton’s law of cooling as

$$\frac{\partial T}{\partial x} = h \left( T_a(y) - T \right), \quad \frac{\partial T_s}{\partial x} = h \left( T_a(y) - T_s \right) \quad \text{on} \quad \partial D_w$$  \hfill (2.31)

where $h$ is the heat transfer coefficient between the ampoule and the ambient which is itself at the temperature of $T_a$. The heat transfer coefficient is assumed to be constant, and the profile for $T_a$ is assumed to be linear in the vertical direction. The value of $T_a$ is $T_h$ at the top and $T_c$ at the bottom of the computational domain. The temperatures at the top and bottom of the computational domain are equal to the
set values

\[ T = T_h \quad \text{on} \quad \partial D_t \]  \hspace{1cm} (2.32)

\[ T_s = T_c \quad \text{on} \quad \partial D_b \]  \hspace{1cm} (2.33)

The temperatures in the solid and melt are equal to the melting temperature at the interface \( \partial D_i \) which is given by the Gibbs-Thomson equation [273]

\[ T_m = T_{m0} + m(c - \frac{c_0}{\kappa}) - T_m \Gamma H \]  \hspace{1cm} (2.34)

where \( T_{m0} \) is the melting temperature of a flat interface at a concentration \( c_0/\kappa \), \( m \) is the slope of the liquidus curve, \( T_m \Gamma \) is the capillary parameter, and \( H \) is the surface curvature. The energy balance at the melt/crystal interface gives

\[ -k\mathbf{n} \cdot T + k_s \mathbf{n} \cdot T_s = \left( -V_g + \frac{\partial y_i}{\partial t} \right) \Delta H_s \mathbf{n} \cdot \mathbf{e}_y \quad \text{on} \quad \partial D_i \]  \hspace{1cm} (2.35)

where \( \mathbf{n} \) is the normal unit vector at the interface pointing in the liquid, \( k \) is the thermal conductivity, \( \Delta H_s \) is the latent heat of solidification, and the subscript \( s \) refers to the solid phase. The species transport in the melt is modeled as

\[ \frac{\partial c}{\partial t} + \mathbf{v} \cdot \nabla c = D \nabla^2 c \]  \hspace{1cm} (2.36)

where \( D \) is the solute diffusion coefficient in the melt. The solute does not penetrate the sidewall, and the concentration profile is reflectively symmetric about the midplane or the center-line of the computational domain. These conditions are written as

\[ \frac{\partial c}{\partial x} = 0 \quad \text{on} \quad \partial D_w \cup \partial D_c \]  \hspace{1cm} (2.37)

The top of the computational domain is taken as a pseudo-inlet for the solute, where
2.2 Mathematical Model

<table>
<thead>
<tr>
<th>variable</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>(x, y) W</td>
</tr>
<tr>
<td>velocity</td>
<td>(u, v) ν/W</td>
</tr>
<tr>
<td>time</td>
<td>t W²/ν</td>
</tr>
<tr>
<td>pressure</td>
<td>P ρν²/W²</td>
</tr>
<tr>
<td>temperature</td>
<td>T, Tₜ Tₜ - Tₜ</td>
</tr>
<tr>
<td>concentration</td>
<td>c c₀</td>
</tr>
</tbody>
</table>

Table 2.2: Scales used for putting the variables in dimensionless form.

the solute entering balances the solute transported in the melt by diffusion and convection [70] as

\[ D \frac{\partial c}{\partial t} + V_y c = V_y c₀ \quad \text{on} \quad \partial D_i \quad (2.38) \]

The above condition for solute balance is more rigorous than setting the inlet concentration equal to the bulk value of c₀ [163]. Nonetheless, setting the inlet concentration to the bulk value would not cause large differences in the results because the height of the computational domain is much larger than the length for the exponential decay of the concentration (H \(\gg\) lₙ = D/Vₚ); this implies that \(\partial c/\partial y \approx 0\) at \(\partial D_i\), and condition (2.38) reduces to \(c \approx c₀\). The solute balance at the melt/crystal interface gives

\[ -D \mathbf{n} \cdot \nabla c = \left( V_y - \frac{\partial y}{\partial t} \right) (1 - k)c \quad \text{on} \quad \partial D_i \quad (2.39) \]

The initial conditions for the field variables and the interface shape are taken from the basic state, which is described in Section 2.2.3.

The scales used to put the variables in the dimensionless form are summarized in Table 2.2. The dimensionless variables are defined in Table 2.3 and and the dimensionless numbers, parameters, and ratios are listed in Table 2.4. Substituting the
\[
\begin{align*}
(x', y') &\equiv \left( \frac{x}{W}, \frac{y}{W} \right) \\
(u', v') &\equiv \left( \frac{u}{W}, \frac{v}{W} \right) \\
t' &\equiv \frac{\nu}{W^2} t \\
P' &\equiv \frac{W^2}{\varrho \mu} P \\
T' &\equiv \frac{T - T_c}{T_h - T_c} \\
\xi' &\equiv \frac{\xi}{\xi_0} - 1 \\
H' &\equiv WH
\end{align*}
\]

Table 2.3: The definition of dimensionless variables as produced by using the scales which are summarized in Table 2.2.

\[
\begin{align*}
\text{Ra}_T &\equiv \frac{\beta gD \Delta TW^3}{\alpha \nu} \\
\text{Ra}_v &\equiv \frac{\beta g \varrho_0 W^3}{\alpha \nu} \\
\text{Pe} &\equiv \frac{\nu_2 W}{\alpha} \\
\text{Pe}_s &\equiv \frac{\nu_2 W}{\alpha_s} \\
\text{Pr} &\equiv \frac{\nu}{\alpha} \\
\text{Sc} &\equiv \frac{\nu}{D} \\
\text{St} &\equiv \frac{\Delta H_s}{C_p \Delta T} \\
\text{Bi} &\equiv \frac{bW}{k} \\
\gamma_s &\equiv \frac{\alpha_s}{\alpha} \\
K_s &\equiv \frac{k_s}{k} \\
T_{m0}' &\equiv \frac{T_{m0} - T_c}{T_h - T_c} \\
m' &\equiv \frac{mc}{\Delta T} \\
T_{m1}' &\equiv \frac{T_{m1} \Gamma'}{\Delta TW} \\
A &\equiv \frac{H}{W}
\end{align*}
\]

Table 2.4: The definition of dimensionless parameters as produced by using the scales which are summarized in Table 2.2.
dimensionless variables into the transport equations and dropping the prime symbols for simplicity give the dimensionless differential equations for momentum and continuity as

\[
\nabla \cdot \mathbf{v} = 0
\]

\[
\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla P + \nabla^2 \mathbf{v} + \frac{1}{\text{Pr}} [\text{Ra}_c c + \text{Ra}_T (T - 1)] e_y
\]

for energy as

\[
\text{Pr} \left( \frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T \right) = \nabla^2 T
\]

\[
\text{Pr} \frac{\partial T_s}{\partial t} - \text{Pe} e_y \cdot \nabla T_s = \gamma_s \nabla^2 T_s
\]

and for the species concentration as

\[
\text{Sc} \left( \frac{\partial c}{\partial t} + \mathbf{v} \cdot \nabla c \right) = \nabla^2 c
\]

The dimensionless boundary conditions on the velocity field in the melt are

\[
u = 0, \quad \frac{\partial u}{\partial x} = 0 \quad \text{on} \quad \partial D_i \cup \partial D_t
\]

\[
u = 0, \quad \frac{\partial u}{\partial x} = 0 \quad \text{on} \quad \partial D_c \cup \partial D_w
\]

or

\[
v = -\text{Pe} \quad \text{on} \quad \partial D_w
\]

when the sidewall is rigid, for the temperature

\[
T = 1 \quad \text{on} \quad \partial D_t
\]
$T = 0$ on $\partial D_b$ \hspace{1cm} (2.49)

$\frac{\partial T}{\partial x} = 0$ on $\partial D_c \cup \partial D_w$ \hspace{1cm} (2.50)

$T = T_s = T_{m0} + mc - T_m \Gamma H$ on $\partial D_i$ \hspace{1cm} (2.51)

$-n \cdot \nabla T + K \sigma_n \cdot \nabla T_s = \text{St} \left( -\text{Pe} + \text{Pr} \frac{\partial y_i}{\partial t} \right) n \cdot e_y$ on $\partial D_i$ \hspace{1cm} (2.52)

or

$\frac{\partial T}{\partial x} = \text{Bi} \left( T_s(y) - T \right)$, $\frac{\partial T_s}{\partial x} = \text{Bi} \left( T_s(y) - T_s \right)$ on $\partial D_w$ \hspace{1cm} (2.53)

when heat leakage is allowed through the sidewall, and for the concentration

$\frac{\partial c}{\partial x} = 0$ on $\partial D_c \cup \partial D_w$ \hspace{1cm} (2.54)

$\frac{\partial c}{\partial y} = \frac{\text{Pe} \text{Sc}}{\text{Pr}} c$ \hspace{1cm} \text{or} $\partial D_i$ \hspace{1cm} (2.55)

$-n \cdot \nabla c = \text{Sc} \left( -\frac{\text{Pe}}{\text{Pr}} + \frac{\partial y_i}{\partial t} \right) n \cdot e_y(1 - \kappa)(c + 1)$ on $\partial D_i$ \hspace{1cm} (2.56)

### 2.2.3 Convectionless, Unidirectional Solidification

Equations (2.40)-(2.56) admit a steady-state one-dimensional solution which serves as the basic state for the analysis which follows. The basic state is convectionless and has a planar interface, which is written mathematically as

\[ u = 0 \] \hspace{1cm} (2.57)

\[ v = -\text{Pe} \] \hspace{1cm} (2.58)

\[ y_i = y_{i0} \] \hspace{1cm} (2.59)
2.2 Mathematical Model

The continuity equation (2.40) is automatically satisfied with the above form for the velocity. The $x$-momentum equation gives

$$\frac{\partial P}{\partial x} = 0 \quad (2.60)$$

which implies that the pressure satisfies $P = P(y)$. The $y$-momentum equation becomes

$$\frac{dP}{dy} = Pr \ \text{Ra}_T (T - 1) \quad (2.61)$$

which determines the pressure field given the temperature profile in the melt. The temperature equations for the melt and for the solid are

$$-Pe \ \frac{dT}{dy} = \frac{d^2 T}{dy^2} \quad (2.62)$$

$$-Pe_s \ \frac{dT_s}{dy} = \frac{d^2 T_s}{dy^2} \quad (2.63)$$

The boundary conditions on the temperature field are

$$T = 1 \ \text{at} \ y = A \quad (2.64)$$

$$T_s = 0 \ \text{at} \ y = 0 \quad (2.65)$$

$$T = T_s = T_{m0} \ \text{at} \ y = y_i \quad (2.66)$$

where $A$ is the dimensionless height of the domain. Integrating the equations for the temperature and applying the boundary conditions (2.64)-(2.66) give

$$T' = 1 + (T_{m0} - 1) \frac{1 - e^{-Pe(A-y)}}{1 - e^{-Pe(A-y_i)}} \quad (2.67)$$

$$T_s = T_{m0} \frac{1 - e^{-Pe_s y_i}}{1 - e^{-Pe_s y_i}} \quad (2.68)$$

Because $Pe$ and $Pe_s$ are usually small due to large thermal diffusivities of semiconductor melts and the low growth rates used, the length scales for the exponential decay
are large for the temperature; this is in contrast to the length scale for the concentration which is much smaller; i.e. \( l_g \ll \text{Pe}^{-1} \). Furthermore, the exponential profiles in Eqs. (2.67)-(2.68) appear nearly linear for the temperature because the domain height is usually much smaller than the exponential decay length for the temperature; i.e. \( A \ll \text{Pe}^{-1} \).

The position of the interface \( y_i \) is found from the heat flux balance at the interface

\[ -\frac{dT}{dy} = -K_s \frac{dT_s}{dy} + \text{Pe \ St} \quad (2.69) \]

which gives a nonlinear equation in \( y_i \)

\[ (T_{m0} - 1) \left( 1 - e^{-\text{Pe} y_i} \right) = \frac{K_s}{\gamma_s} T_{m0} \left( 1 - e^{-\text{Pe}(y_i - A)} \right) - \text{St} \quad (2.70) \]

Also, using Eq. (2.67) for the temperature gives the pressure field as

\[ P = \frac{\text{Pr} \, \text{Ra}_T}{1 - e^{-\text{Pe}(A-y_i)}} \left[ y - \frac{1}{\text{Pe}} \left( 1 - e^{-\text{Pe}(A-y_i)} \right) \right] \quad (2.71) \]

Because the length scale is small for the exponential decay of the concentration field, i.e. \( l_g \ll A \), the domain appears infinite for the concentration field, and Eq. (2.4) derived in Section 2.1 is a valid description of the concentration field here as well.

The above equations simplify for the limit of small Peclet numbers, i.e. \( \text{Pe} \ll 1 \) and \( \text{Pe}_s \ll 1 \). The exponentials are expanded in Taylor Series as: \( e^x \approx 1 + x + x^2/2 + \ldots \) to yield

\[ y_i \approx \frac{1}{1 + K_s \frac{T_{m0}}{T_{m0} - 1}} \quad (2.72) \]

\[ T \approx 1 + (T_{m0} - 1) \frac{A - y}{A - y_i} \quad (2.73) \]

\[ T_s \approx T_{m0} \frac{y}{y_i} \quad (2.74) \]

\[ P \approx \text{Pr} \, \text{Ra}_T \left( 1 - T_{m0} \right) \frac{(A - y)^2}{2(A - y_i)} \quad (2.75) \]
2.2 Mathematical Model

The above derivation assumes a finite height for the domain. The equations for the temperature can be derived alternatively, assuming that the height is infinite and the origin of the coordinate frame of reference is at the interface. Linear stability analysis usually uses assumes these conditions [203, 59, 65, 213].

It is not possible to set essential boundary conditions for the temperature at infinity for both the solid and the liquid, so the temperature gradient is set at the interface instead. The equations for the temperature become

\[
T = T_{m0} - \frac{G}{\text{Pe}} \left( e^{-\text{Pe} \gamma y} - 1 \right) \tag{2.76}
\]

\[
T_s = T_{m0} - \frac{G_s}{\text{Pe}_s} \left( e^{-\text{Pe}_s \gamma y} - 1 \right) \tag{2.77}
\]

where \( G \) is the dimensionless temperature gradient in the liquid at the interface, and \( G_s = \frac{\kappa}{\kappa_s} (G + \text{Pe} \text{St}) \) is the temperature gradient in the solid at the interface. Equation (2.76) is valid in the melt, i.e. for \( y \geq 0 \), and Eq. (2.77) is valid in the crystal, i.e. for \( y \leq 0 \).

2.2.4 Disturbance Equations and Boundary Conditions for Linear Stability Analysis

The presentation of linear stability analysis given here closely follows those of Coriell et al. [59] and by Öztékin [213]. Perturbations are introduced in all the field variables and in the interface position. The perturbed variables are

\[
v = v_{bs}(y) + v_r(x, y, z, t) \tag{2.78}
\]

\[
P = P_{bs}(y) + P_r(x, y, z, t) \tag{2.79}
\]

\[
T = T_{bs}(y) + T_r(x, y, z, t) \tag{2.80}
\]

\[
T_s = T_{s,bs}(y) + T_{s,r}(x, y, z, t) \tag{2.81}
\]

\[
c = c_{bs}(y) + c_r(x, y, z, t) \tag{2.82}
\]
where the subscript $bs$ refers to the basic state solution, and the subscript $\epsilon$ refers to the perturbation. Substituting the forms of variables given by Eqs. (2.78)-(2.83) into the field equations, and keeping only the terms that are first-order in the disturbance give the linear stability equations as

\[
\nabla \cdot \mathbf{v}_\epsilon = 0 \tag{2.84}
\]

\[
\frac{\partial \mathbf{v}_\epsilon}{\partial t} - \text{Pe} \frac{\partial \mathbf{v}_\epsilon}{\partial y} = -\nabla P_\epsilon + \nabla^2 \mathbf{v}_\epsilon + \frac{1}{\text{Pr}} (\text{Ra}_c \, c_\epsilon + \text{Ra}_T \, T_\epsilon) \mathbf{e}_y \tag{2.85}
\]

\[
\frac{\partial T_\epsilon}{\partial t} - \text{Pe} \frac{\partial T_\epsilon}{\partial y} + v_\epsilon \frac{\partial T_\epsilon}{\partial y} = \frac{1}{\text{Pr}} \nabla^2 T_\epsilon \tag{2.86}
\]

\[
\text{Pr} \frac{\partial T_{s,\epsilon}}{\partial t} - \text{Pe} \frac{\partial T_{s,\epsilon}}{\partial y} = \gamma_s \nabla^2 T_{s,\epsilon} \tag{2.87}
\]

\[
\frac{\partial c_\epsilon}{\partial t} - \frac{\text{Pe}}{\text{Pr}} \frac{\partial c_\epsilon}{\partial y} + v_\epsilon \frac{\partial c_\epsilon}{\partial y} = \frac{1}{\text{Sc}} \nabla^2 c_\epsilon \tag{2.88}
\]

Eliminating pressure in the momentum equation and using continuity give an equation for the $y$-component of the velocity as

\[
\frac{\partial \nabla^2 v_\epsilon}{\partial t} - \frac{\partial \nabla^2 v_\epsilon}{\partial y} = \nabla^2 v_\epsilon + \frac{\text{Ra}_T}{\text{Pr}} \nabla^2 T_\epsilon + \frac{\text{Ra}_c}{\text{Pr}} \nabla^2 c_\epsilon \tag{2.89}
\]

where

\[
\nabla^2 \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2} \tag{2.90}
\]

The boundary conditions for the disturbances at the interface are

\[
v_\epsilon = -\epsilon \frac{\partial y_i,\epsilon}{\partial t} \tag{2.91}
\]

\[
w_\epsilon = \epsilon \text{Pe} \frac{\partial y_{i,\epsilon}}{\partial z} \tag{2.92}
\]

\[
u_\epsilon = \epsilon \text{Pe} \frac{\partial y_{i,\epsilon}}{\partial x} \tag{2.93}
\]
\[ \text{St Pr } \frac{\partial y_{i,c}}{\partial t} = \left[ \text{Pe } G y_{i,c} - \frac{\partial T_c}{\partial y} \right] + K_s \left[ -\text{Pe } G_s y_{i,c} + \frac{\partial T_{s,c}}{\partial y} \right] \quad (2.94) \]

\[ \frac{1 - \kappa}{\kappa} \frac{\partial y_{i,c}}{\partial t} = \frac{\kappa}{\text{Pr}} \frac{\text{Pe } G e y_{i,c} - (1 - \kappa) \text{Pr } c_e}{\text{Sc } \frac{\partial c_e}{\partial y}} \quad (2.95) \]

\[ G_s y_{i,c} + T_{s,c} = G y_{i,c} + T_c \quad (2.96) \]

\[ T_c + G y_{i,c} = m G e y_{i,c} + m c_e + T_m \Gamma \nabla^2 \nabla \cdot y_{i,c} \quad (2.97) \]

where \( w \) is the \( z \)-component of the velocity vector and \( G_e \equiv -\frac{\text{Pe } \text{Sc}}{\text{Pr}} \frac{1 - \kappa}{\kappa} \) is the dimensionless solute gradient at the interface. The boundary conditions (2.92) and (2.93) are combined to yield

\[ \frac{\partial v_i}{\partial y} = -\varepsilon \text{Pe } \nabla^2 \nabla \cdot y_{i,c} \quad (2.98) \]

wherein the continuity equation is used, thereby eliminating the \( x \)- and \( z \)-components of the disturbance velocity from the boundary conditions.

The dimensionless field variables and interface position are written in the Fourier series form, \( \Sigma_k A_k(y) \exp(\sigma_k t + i\omega_{x,k} x + i\omega_{z,k} z) \), where \( A_k \) is the amplitude of the disturbance, \( \sigma_k \) the temporal growth rate of the disturbance, \( \omega_{x,k} \) the \( x \)-component of the wavenumber of the disturbance, and \( \omega_{z,k} \) the \( z \)-component of the wavenumber of the disturbance. The normal mode representation is possible because the above equations and boundary conditions are separable in \( x \), \( y \), and \( t \); this approach is known as either modal expansion, separation of variables, or Fourier transform. The general solution is a superposition of all possible wavenumbers (\( \{ \omega_{x,k}, \omega_{y,k}, \sigma_k \} \) for \( k = \{0, 1, ..., \infty\} \)). Usually, the subscript \( k \) is dropped for convenience.

Substituting the disturbances of the form

\[ v_i = \hat{v}(y)e^{\sigma t + i\omega_{x,i} x + i\omega_{z,i} z} \quad (2.99) \]

\[ T_i = \hat{T}(y)e^{\sigma t + i\omega_{x,i} x + i\omega_{z,i} z} \quad (2.100) \]

\[ T_{s,c} = \hat{T}_s(y)e^{\sigma t + i\omega_{x,i} x + i\omega_{z,i} z} \quad (2.101) \]

\[ c_e = \hat{c}(y)e^{\sigma t + i\omega_{x,i} x + i\omega_{z,i} z} \quad (2.102) \]
\[ y_{i,c} = y_t e^{\sigma t + i \omega_c x + i \omega_z z} \]  

(2.103)

into the differential equations gives

\[
\sigma (\dot{\phi}^{(2)} - \omega^2 \dot{\phi}) + \dot{\phi}^{(3)} - \omega^2 \dot{\phi}^{(1)} = \dot{\phi}^{(4)} - 2\omega^2 \dot{\phi}^{(2)} + \omega^2 \ddot{\varrho} - \frac{\text{Ra}_T}{\text{Pr}} \omega^2 \hat{T} - \frac{\text{Ra}_c}{\text{Pr}} \omega^2 \ddot{c} 
\]  

(2.104)

\[
\sigma \hat{T} - \text{Pe} \hat{T}^{(1)} + \dot{\varrho} G \varrho_{e} \text{Pe} y = \frac{1}{\text{Pr}} (-\omega^2 \hat{T} + \hat{T}^{(2)}) 
\]  

(2.105)

\[
\sigma \hat{T}_s - \text{Pe} \hat{T}_s^{(1)} = \frac{\gamma_s}{\text{Pr}} (-\omega^2 \hat{T}_s + \hat{T}_s^{(2)}) 
\]  

(2.106)

\[
\sigma \ddot{c} - \frac{\text{Pe}}{\text{Pr}} \dot{c}^{(1)} + \dot{\varrho} G_{c} e^{- \frac{\text{Pe} \text{Sc}}{\text{Pr}}} y = \frac{1}{\text{Sc}} (-\omega^2 \ddot{c} + \ddot{c}^{(2)}) 
\]  

(2.107)

where superscripts in parentheses signify derivatives with respect to \( y \). Substituting the forms for the disturbance given by Eqs. (2.99)-(2.103) into the boundary conditions at the interface yields

\[
\dot{\varrho} = -\epsilon \sigma \hat{y}_i 
\]  

(2.108)

\[
\dot{\varrho}^{(1)} = \epsilon \text{Pe} \omega^2 \hat{y}_i 
\]  

(2.109)

\[
\sigma \text{St} \text{Pr} \hat{y}_i = \text{Pe} \varrho \hat{y}_i - \hat{T}^{(1)} + K_s (-\text{Pe}_{s} \varrho \hat{y}_i + \hat{T}_s^{(1)}) 
\]  

(2.110)

\[
\frac{1 - \kappa}{\kappa} \dot{\hat{y}}_i = \frac{\text{Pe}}{\text{Pr}} \varrho \hat{y}_i - (1 - \kappa) \frac{\text{Pe}}{\text{Pr}} \dot{c} - \frac{1}{\text{Sc}} \dot{c}^{(1)} 
\]  

(2.111)

\[
G_s \hat{y}_i + \hat{T}_s = G \hat{y}_i + \hat{T} 
\]  

(2.112)

\[
\hat{T} + G \hat{y}_i = m G_{c} \hat{y}_i + m \dot{c} - \omega^2 T_{m} \varrho \hat{y}_i 
\]  

(2.113)

The five differential equations and six boundary conditions, Eqs. (2.104)-(2.113), are linear and homogeneous. Four more boundary conditions are required because the differential equations are fourth order in \( \dot{\varrho} \), second order in \( \hat{T}, \hat{T}_s \), and \( \dot{c} \), and zeroth order in \( y_i \). The far-field boundary conditions \( (y \rightarrow +\infty) \) are set for the disturbances as

\[
\dot{\varrho} = \dot{\varrho}^{(1)} = \hat{T} = \dot{c}^{(1)} = 0 
\]  

(2.114)
which require that the disturbances decay far from the interface.

# 2.3 Numerical Methods

## 2.3.1 Linear Stability

The numerical method used to solve Eqs. (2.104)-(2.113) is described by Öztékín [213]; it is based on a spectral Galerkin representation developed by Zebib [280, 279]. The highest derivatives are approximated with truncated Chebyshev polynomials for the amplitude of the disturbances as

\[
\hat{\psi}^{(4)}(y) \simeq \sum_{j=0}^{J} v_j T_j(y) \\
\hat{T}^{(2)}(y) \simeq \sum_{j=0}^{J} t_j T_j(y) \\
\hat{T}_s^{(2)}(y) \simeq \sum_{j=0}^{J} t_{s,j} T_j(y) \\
\hat{c}^{(2)}(y) \simeq \sum_{j=0}^{J} c_j T_j(y)
\]  

(2.115) (2.116) (2.117) (2.118)

where \( T_j \) are the Chebyshev polynomials and \( (v_j, t_j, t_{s,j}, c_j) \) are the unknown coefficients. Chebyshev polynomials satisfy the properties

\[
2T_n(y) = \frac{\alpha_n}{n+1} \frac{dT_{n+1}(y)}{dy} - \frac{\lambda_{n-2}}{n-1} \frac{dT_{n-1}(y)}{dy}
\]

(2.119)

\[
2yT_n(y) = \alpha_n T_{n+1}(y) - \lambda_{n-1} T_{n-1}(y)
\]

(2.120)
where the constants $\alpha_n$ and $\lambda_n$ are

$$
\alpha_n = \begin{cases} 
0 & \text{for } n < 0 \\
2 & \text{for } n = 0 \\
1 & \text{for } n > 0 
\end{cases} 
$$  (2.121)

$$
\lambda_n = \begin{cases} 
0 & \text{for } n < 0 \\
1 & \text{for } n = 0 \\
1 & \text{for } n > 0 
\end{cases} 
$$  (2.122)

The lower-order derivatives follow from integrating the Chebyshev approximations for the amplitudes, using the properties of the Chebyshev polynomials, and utilizing the boundary conditions to determine the integration constants. For example, integrating Eq. (2.115) and using the properties of the Chebyshev polynomials give

$$
\hat{v}^{(3)} = \sum_{j=0}^{J+1} \sum_{i=0}^{J} f_{ji}^{(3)} v_i T_j + C_1 T_0 
$$  (2.123)

$$
\hat{v}^{(2)} = \sum_{j=0}^{J+2} \sum_{i=0}^{J} f_{ji}^{(2)} v_i T_j + C_1 T_1 + C_2 T_0 
$$  (2.124)

$$
\hat{v}^{(1)} = \sum_{j=0}^{J+3} \sum_{i=0}^{J} f_{ji}^{(1)} v_i T_j + \frac{C_1}{4} T_2 + C_2 T_1 + C_3 T_0 
$$  (2.125)

$$
\hat{v} = \sum_{j=0}^{J+4} \sum_{i=0}^{J} f_{ji}^{(0)} v_i T_j + \frac{C_1}{24} T_3 + \frac{C_2}{4} T_2 + \left( C_3 - \frac{C_1}{8} \right) T_1 + C_4 T_0 
$$  (2.126)

where the constants $f_{ji}^{(\beta)}$ for $\beta = \{0, 1, 2, 3\}$ are obtained from properties (2.119) and (2.120), and are given by Zebib [279]; the constants of integration are determined by using the boundary conditions. The general form for $\hat{\nu}^{(\beta)}$, $\beta = 0, 1, 2, 3$ is

$$
\hat{\nu}^{(\beta)} = \sum_{j=0}^{J+4-\beta} \sum_{i=0}^{J} v_{ji}^{(\beta)} v_i T_j(y) 
$$  (2.127)
where the constants \( \{C_k\} \) and \( \{f_j^{(\beta)}\} \) are combined into \( \{v_{ji}^{(\beta)}\} \). The Chebyshev expansions are substituted into the governing equations, and to avoid spurious eigenvalues, inner products are formed with

\[
\sum_{j=0}^{J+4} v_{ji}^{(0)} T_j, \quad \sum_{j=0}^{J+2} t_{ji}^{(0)} T_j, \quad \sum_{j=0}^{J+2} t_{sj,i}^{(0)} T_j, \quad \sum_{j=0}^{J+2} c_{ji}^{(0)} T_j
\]  

(2.128)

rather than with the Chebyshev polynomials [213, 280]. The spurious eigenvalues are avoided because the set of functions (2.128) satisfy the boundary conditions, as shown by Chandrasekhar [44].

The above procedure reduces the solution of the differential equations and boundary conditions to a matrix eigenvalue problem

\[
As + \sigma Bs = 0
\]  

(2.129)

where \( A \) and \( B \) are square matrices, \( \sigma \) is the temporal eigenvalue, and

\[
s \equiv [v_j \ t_j \ t_{s,j} \ c_j \ \hat{y}_i]^T
\]

is the eigenvector.

The eigenvalues \( \{\sigma_k\} \) determine the stability of the system. If \( \text{Re}(\sigma_k) < 0 \) for all \( k \), the system is stable, but if \( \text{Re}(\sigma_k) > 0 \) for a single \( k \), the system is unstable. The condition \( \text{Re}(\sigma_k) = 0 \) signifies a point of neutral stability where the amplitudes of the disturbances neither grow nor decay temporally. This point also is called the onset of instability; if \( \text{Im}(\sigma_k) = 0 \), the onset of instability is steady, and if \( \text{Im}(\sigma_k) \neq 0 \) the onset is oscillatory in time.

The neutral stability curves which are presented in this Chapter show the critical concentration \( c_0 \) as a function of the spatial wavenumber \( \omega = \sqrt{\omega_x^2 + \omega_z^2} \) of the instability. The curves are computed point-by-point through vertical sweeps in concentration: the concentration is increased at a fixed spatial wavenumber, and the
spectrum is checked by computing \{\sigma_k\} from Eq. (2.129) for each value of the concentration. A bisection method is used to focus on the neutral stability point [213] after a change in sign is detected in Re(\sigma_k) for some eigenvalue \sigma_k.

2.3.2 Nonlinear Simulations

The discretization and solution methods used to solve nonlinear problems are summarized in this Section. More details can be found in the theses by Chang [45], Adornato [1] and Kim [163]. The time-dependent nonlinear differential equations and boundary conditions governing thermosolutal and morphological instabilities are discretized with Galerkin finite elements. The mesh is fixed to the interface, so it moves with it. The representation for the interface is \( y_i = h(x, t) \), making it a function of the lateral coordinate only, and enabling it to move in time. The unknown variables are the two components of the velocity \((u, v)\) in the melt, the temperature in the melt \( T \), the temperature in the solid \( T_s \), the concentration of the solute \( c \), and the interface position \( h \). The field variables are approximated by biquadratic Lagrangian basis functions \{\Phi^j\} except for the pressure, which is approximated by the discontinuous piecewise-linear functions \{\Psi^j\} in order to satisfy the \( LBB \) condition [168, 12, 25]. The approximate forms of the field variables are

\[
\begin{bmatrix}
  u(x, y, t) \\
  v(x, y, t) \\
  T(x, y, t) \\
  c(x, y, t)
\end{bmatrix}
\approx
\begin{bmatrix}
  u_j(t) \\
  v_j(t) \\
  T_j(t) \\
  c_j(t)
\end{bmatrix}
\Phi^j(x(t), y(t)) = \sum_{j=1}^{N} \Phi^j(x(t), y(t))
\]

\[
T_s(x, y, t) = \sum_{j=1}^{N_s} T_{s,j}(t)\Phi^j(x(t), y(t))
\]

\[
P(x, y, t) = \sum_{j=1}^{3} P_j^p(t)\Psi^j(\xi, \eta)
\]
where $N$ is the number of bi-quadratic nodes in the melt, $N_s$ is the number of bi-quadratic nodes in the solid, the superscript $e$ denotes the element number in the pressure approximation, and the coefficients (such as $T_j(t)$) are the values of the field variable at some node. The interface position is approximated by quadratic basis functions $\{\phi^j\}$ as

$$ h \simeq \sum_{j=1}^{n} h_j(t) \phi^j(x(t)) \quad (2.133) $$

where $n$ is the number of quadratic nodes along the interface. The $y$-coordinates of the nodes move in time because the mesh is fixed to the interface. Therefore, the position is approximated as

$$ x \simeq \sum_{j=1}^{9} x_j \Phi^j(\xi, \eta) \quad (2.134) $$

$$ y \simeq \sum_{j=1}^{9} y_j(h_j(t)) \Phi^j(\xi, \eta) \quad (2.135) $$

where $\{x_j, y_j\}$ are the coordinates of the mesh nodal points.

The weighting functions in the Galerkin finite element method are $\{\Psi^i\}$ for the continuity equation, $\{\Phi^i\}$ for the momentum, energy, and solute equations, and $\{\phi^j\}$ for the interface position. Requiring the weighting functions to be orthogonal to the corresponding residual equations results in the weak forms for the continuity

$$ \int_{D(t)} \Psi^i(\nabla \cdot \mathbf{v}) \, dA = 0 \quad (2.136) $$

momentum

$$ \int_{D(t)} \Phi^i \left( \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} + \nabla P - \nabla^2 \mathbf{v} - \frac{1}{\text{Pr}} [\text{Ra}_c c + \text{Ra}_T T] e_y \right) \, dA = 0 \quad (2.137) $$
energy

\[
\int_{D(t)} \Phi^i \left( \Pr \left( \frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T \right) - \nabla^2 T \right) \, dA = 0 \tag{2.138}
\]

\[
\int_{D_s(t)} \Phi^i \left( \Pr \frac{\partial T_s}{\partial t} - \text{Pe} \, e_y \cdot \nabla T_s - \gamma_s \nabla^2 T_s \right) \, dA = 0 \tag{2.139}
\]

and species conservation

\[
\int_{D(t)} \Phi^i \left( \text{Sc} \left( \frac{\partial c}{\partial t} + \mathbf{v} \cdot \nabla c \right) - \nabla^2 c \right) \, dA = 0 \tag{2.140}
\]

equations, where \( D(t) \) refers to the melt region and \( D_s(t) \) refers to the solid region. The weak form for the interface position comes from the melting point condition at the interface, which is

\[
\int_{d(t)} \phi^i \left( T(x, h) - T_m \right) \, dS = 0 \tag{2.141}
\]

wherein \( d(t) \) is the interface, and \( dS \) is defined as

\[
dS \equiv \sqrt{1 + (\partial h/\partial x)^2} \, dx \tag{2.142}
\]

Integration by parts to eliminate the second derivatives, application of the boundary conditions, and substitution of the approximate forms for the field variables result in a set of nonlinear ordinary differential equations referred to as a differential algebraic set, which has the form of

\[
\sum_{j=1}^{N_i} M_{ij} \frac{du_j}{dt} = F_i(u_j) \tag{2.143}
\]

where \( N_i \) is the total number of unknowns and the indices \( i \) and \( j \in \{1, 2, \ldots, N_i\} \) refer to equation and unknown, respectively. The coefficient matrix \( M \), usually called the mass matrix, is singular since the continuity equation and the melt/crystal interface do not explicitly depend on time; therefore, the differential algebraic set is stiff [104, 217], and explicit time integration methods are not efficient. The differential algebraic
set is integrated by low-order implicit schemes, such as the Backward Euler

\[ y'_{n+1} \approx \frac{y_{n+1} - y_n}{\Delta t} \]  

or the Trapezoidal rule

\[ y'_{n+1} \approx \frac{2}{\Delta t} (y_{n+1} - y_n) - y'_n \]

which have large stability regions. Discretization in time produces a set of nonlinear algebraic equations at each time step

\[ R_i(u_j) = 0 \]

which is solved with either Newton’s method as shown in Fig. 3-5, modified Newton’s method as shown in Fig. 2-11, or a hybrid of these two methods.

Steady-state solutions of the solidification problem are found by setting the time derivatives in Eq. (2.143) equal to zero. This results in the nonlinear set of equations

\[ F_i(u_j; c_0) = 0 \]

for a given set of parameter values. The bulk concentration is included in Eq. (2.147) as a variable because families of solutions are calculated with the bulk concentration as the variable parameter. The value of \( c_0 \) is incremented or decremented and successive solutions are computed as a function of \( c_0 \). Tracking of a family of solutions as a function of the parameter \( c_0 \) is achieved by using the converged solution at the previous value of \( c_0 \) as the initial guess to the Newton iterations at the new value of \( c_0 \). More elaborate continuation methods can be used for more efficient solution tracking, such as the methods described by Kubicek [166] and Keller [162] and used by Yamaguchi et al. [274] and Ungar and brown [261].
form the coefficient matrix (Jacobian) for the linear system:

\[ J_{ij} = \left( \frac{\partial f_i}{\partial u_j} \right) \]

do \( LU \)-decomposition on the Jacobian \( J \):

\[ J_{ij} = L_{im}U_{mj} \quad \text{where} \quad L_{ii} = 1 \quad \text{and} \quad L_{im} = U_{mj} = 0 \quad \text{for} \quad m > i, j \]

for \( k = 1 \rightarrow k_{\text{max}} \) or convergence

solve the system:

\[ L_{im}U_{mj}\delta u_j^k = -R_i^k \quad \text{by:} \]

forward elimination (FE)

\[ L_{im}\delta z_m^k = -R_i^k \quad \text{where} \quad \delta z_m^k \equiv U_{mj}\delta u_j^k \]

back substitution (BS)

\[ U_{mj}\delta u_j^k = z_m^k \]

update solution:

\[ u_j^{k+1} = u_j^k + \delta u_j^k \]

test for convergence:

\[ \max_j |\delta u_j^k| < \epsilon_{a,\infty} \quad \text{or} \quad \max_j \frac{|\delta n_j^k|}{|n_j^k|} < \epsilon_{r,\infty} \]

Figure 2-11: Modified (or quasi-) Newton’s method. A variation of Einstein’s summation notation is used for convenience: when the same subscript appears in a term on two or more variables, the summation is implied over that index.
2.3 Numerical Methods

2.3.3 Bifurcating Solutions and Solution Tracking

Continuous families of the solutions are calculated by continuation as long as the solutions are specified uniquely by the parameter $c_0$ and no bifurcations to other families occur. The non-uniqueness of a single family with $c_0$ occurs near a limit point $c_{0f}$ where the family reverses direction in $c_0$ [152]. Classical perturbation methods have been adapted for numerical analysis of the solutions near bifurcation $(c_{0r})$ and limit points by many investigators [162, 32, 274]; the description below is based on these various methods. The methods employed here are quite simple; however, for the sake of completeness, more rigorous methods for solution tracking around limit points and near bifurcation points also are discussed.

Near limit points and bifurcation points, an amplitude parameter $\epsilon$ is introduced so that each family of the solutions is specified uniquely as $(u(\epsilon), c_0(\epsilon))$ in the close proximity of the singular point at $\epsilon = 0$. The solution vector $u$ and the parameter $c_0$ are expanded in terms of $\epsilon$ as [274]

$$
\begin{bmatrix}
    u(\epsilon) \\
    c_0(\epsilon)
\end{bmatrix}
= \sum_{k=0}^{\infty} \frac{\epsilon^k}{k!}
\begin{bmatrix}
    u(k) \\
    c_0(k)
\end{bmatrix}
$$

(2.148)

where the coefficients of the expansion are determined from Eq. (2.147), which is expanded to the appropriate order of $\epsilon$. For example, the coefficients of the first two terms $(u(1), c_0(1))$ and $(u(2), c_0(2))$ are

$$
F_u(u(0); c_0(0))u(1) = -F_{c_0}(u(0); c_0(0))c_0(1)
$$

(2.149)

$$
F_u(u(0); c_0(0))u(2) = -F_{uu}u(1)u(1) - 2F_{ur}(u(0); c_0(0))u(1)c_0(1)
- F_{c_0}(u(0); c_0(0))c_0(2) - F_{r_2}c_0^2
$$

(2.150)

as given by Yamaguchi et al. [274]. The subscripts on $F$ in Eqs. (2.149)-(2.150) denote derivatives with respect to the indicated variables; for example the components of the
tensor $F_{uu}$ are $\{F_{uu}\}_{ijk} \equiv \partial F_i / \partial u_j \partial u_k$ and the components of the vector $F_{u\nu u}$ are $\{F_{u\nu u}\}_i \equiv \partial^2 F_i / \partial \nu^2$. The matrix $F_u$ is actually the Jacobian matrix evaluated at the singular point. The terms in Eq. (2.148) can be computed from Eqs. (2.149)-(2.150) and higher order expansions of Eq. (2.147), once a definition for the parameter $\epsilon$ is chosen.

The pseudo-arc length is chosen for the definition of $\epsilon$ to track a family of the solutions around limit points [225, 162, 274]. The amplitude parameter $\epsilon$ is defined based on the pseudo-arc length given by the equation

$$S \equiv (s - s_0)^2 + \| u(s) - u(s_0) \|_2^2 + |c_0(s) - c_0(s_0)|^2 \quad (2.151)$$

where $\| u \|_2$ is the $L_2$-norm of $u$, i.e.

$$\| u \|_2 = \sqrt{\sum_j u_j^2} \quad (2.152)$$

Equation (2.149) is combined with the linearization of Eq. (2.151) about the known solution

$$s - s_0 + S u_{(1)} + c_{0(1)} = 0 \quad (2.153)$$

to determine the tangent vector $(u_{(1)}, c_{0(1)})$ which is used for constructing an initial guess for the Newton iterations. The Newton iterations are performed on the Nonlinear equation set (2.147) augmented with Eq. (2.151). The limit point $c_{0\epsilon}$ is determined from the criterion $c_{0(1)} \equiv (dc_0/ds)_{s=0} = 0$. The right side of Eq. (2.149) vanishes and the Jacobian matrix becomes singular at a limit point. Thus, Eq. (2.149) admits a homogeneous solution in the null space of $J$ whose magnitude is fixed by the orthogonalization condition (2.153). The pseudo-arc length approach was not used here since only one limit point was encountered in the computations. This family of solutions was computed close to the limit point by approaching the limit point from both sides.
2.3 Numerical Methods

The relative stability of the solutions in an individual family to small perturbations can change only at bifurcation (including Hopf) and limit points [250, 152, 274]. The linear stability analysis as presented in Section 2.2.4 was used to detect bifurcations from the one-dimensional, basic state solution; the sign of the determinant of the Jacobian matrix, \( \det(J) \), was monitored for secondary bifurcations from the two-dimensional, non-base solutions. The Jacobian matrix \( J \) is singular at the values of \( c_0 \) for bifurcation points, where the equations admit multiple solutions. Only one real eigenvalue of the linearized problem goes through zero at simple bifurcation points, whereas a complex conjugate pair pass through the real axis at Hopf bifurcation points. As a real eigenvalue of the linearized problem goes to zero, \( \det(J) \) changes sign signifying a bifurcation point to a steady solution. However, the Hopf bifurcation points cannot be determined by monitoring the sign of \( \det(J) \) because the Jacobian matrix has real components, and the eigenvalues for this matrix come in complex conjugate pairs. Therefore, a complex conjugate pair of eigenvalues cross the real axis together at a Hopf point, and \( \det(J) \) does not change sign. Instead, linear stability analysis is used to determine the Hopf bifurcation points on the basic state solution. Moreover, linear stability analysis is an efficient method for determining the steady bifurcation points on the basic state solution. However, detection of bifurcation points on the non-base solutions was performed by monitoring the sign of \( \det(J) \), since using linear stability analysis would require fundamental changes to the presentation in Section 2.2.4, including re-derivation of the disturbance equations and boundary conditions (Eqs. (2.104)-(2.113)) for disturbances about a general, two-dimensional solution.

At a bifurcation point, the null vector \( z \) corresponding to the zero eigenvalue of \( J \) and its adjoint vector \( y \) satisfy

\[
J(u(c_0r); c_0r) z = 0 \quad (2.154)
\]

\[
J^T(u(c_0r); c_0r) y = 0 \quad (2.155)
\]
\[ z^T y = 1 \]  

(2.156)

where \( J^T \) is the transpose of the Jacobian matrix \( J \). The null vectors are computed by direct solution of Eq. (2.154). The non-uniqueness of the null vectors and the singularity of the Jacobian matrix are accounted for by ignoring the last row of the Jacobian matrix corresponding to the zero pivot, and by setting the last entry of \( z \) equal to one. The null vector \( z \) may be normalized as is customary so that \( \| z \|_2 = 1 \), but this step is not necessary for the computations and so was omitted.

The continuation vector \( u_{(1)} \), given by Eq. (2.149), is decomposed into the components in the null space and range of \( J \) as

\[ u_{(1)} = z_0' + \alpha z \]  

(2.157)

where \( \alpha \) is a constant to be determined after \( \epsilon \) is defined, and \( z_0' \) is the particular solution of Eq. (2.149) which has no component in the direction of the null vector; i.e. \( y^T z_0' = 0 \). Since the right side of Eq. (2.149) is homogeneous in \( c_{0(1)} \), the particular solution is written as \( z_0' = c_{0(1)} z_0 \) where \( z_0 \) is a particular solution of the system

\[ F_u(u(c_{0c}); c_{0c}) z_0 = -F_{c_0}(u(c_{0c}); c_{0c}) \]  

(2.158)

Equation (2.158) is solvable only if

\[ y^T F_{c_0}(u(c_{0c}); c_{0c}) = 0 \]  

(2.159)

In other words, the vector \( F_{c_0} \) must be in the range of \( J \). Bifurcation between the two solution families is guaranteed if condition (2.159) is satisfied for a simple eigenvalue at \( c_{0c} \).

The amplitude parameter definition used here is similar to the one used by Yam-
aguchi et al. [274] as

$$\epsilon \equiv \alpha y^T(u(\epsilon) - u(0)) + c_{0(1)}(c_0(\epsilon) - c_0(0))$$  \hspace{1cm} (2.160)

where the first term is proportional to the magnitude of the solution along the null vector, and the second term is proportional to the change in the parameter value. At the first order in $\epsilon$, $u(\epsilon) - u(0) = u(1)$ and $c_0(\epsilon) - c_0(0) = c_{0(1)}$, so

$$1 = \alpha^2 + c_{0(1)}^2$$  \hspace{1cm} (2.161)

The solvability condition for the second-order problem (Eq. (2.150)), given by Yamaguchi et al. [274] and Keller [162], sets a second relationship between $\alpha$ and $c_{0(1)}$ as

$$\alpha^2 C_1 + 2\alpha c_{0(1)} C_2 + c_{0(1)}^2 C_3 = 0$$  \hspace{1cm} (2.162)

where

$$C_1 \equiv y^T F_{uu}zz$$  \hspace{1cm} (2.163)
$$C_2 \equiv y^T F_{uu}z_0 z + y^T F_{uu}z$$  \hspace{1cm} (2.164)
$$C_3 \equiv 2y^T F_{uu}z_0 z_0 + y^T F_{uu}z_0 z_0$$  \hspace{1cm} (2.165)

The pair $(\alpha, c_{0(1)})$ is computed as roots of Eqs. (2.161)-(2.162) once the coefficients $\{C_i\}$ are computed at the bifurcation point $c_{0c}$ by Eqs. (2.163)-(2.165). The solution families for the bifurcating branch are computed by using Eqs. (2.148) and by choosing a large enough $\epsilon$ to converge to the desired bifurcating family.

The above procedure for calculating a solution on a bifurcating branch is rigorous, but laborious and complicated; it involves calculation of many tensor, matrix, and vector components, and formation of several inner products. Here, the following, more practical and simpler, method is used for calculation of a solution on the bifurcating
family. If $u$ is a solution to Eq. (2.147), the linearized system

$$\mathbf{J}\delta u = 0$$  \hspace{1cm} (2.166)

has a trivial solution $\delta u = 0$ since $\mathbf{F}(u) = 0$. At a bifurcation point, the solution to Eq. (2.166) has a homogeneous component in the null space of $\mathbf{J}$ as well as the particular solution of zero in the range of $\mathbf{J}$, similar to Eq. (2.157). The solution to Eq. (2.166) at a bifurcation point is written as

$$\delta u = \alpha z$$  \hspace{1cm} (2.167)

where $z$ is the null vector as before. Therefore, the solution at the bifurcation point is approximated to the first-order as

$$u = u^{(0)} + \alpha z$$  \hspace{1cm} (2.168)

A solution on the bifurcating family is computed by varying the value of $\alpha$ and by using Eq. (2.168) to compute an initial guess for the Newton iterations. An appropriate value of $\alpha$ results in convergence to the desired bifurcating family, whereas an inappropriate value of $\alpha$ results in divergence of Newton iterations, or convergence to an undesired family. In practice, this procedure is used near a bifurcation point, rather than at the bifurcation point. The values of $c_0$ at which the “null vector” is computed, as well as the value of $c_{0(1)}$, are important parameters for the Newton iterations to converge to a solution on the desired family. These values are determined by trial and error. The procedure is shown graphically in Fig. 2-12.
Figure 2-12: Computing a solution on a bifurcating family from the basic state solution. $\varepsilon$ is a measure of deviation from the basic state. The failed attempts are shown in dashed lines. One failed attempt results in convergence to base and the other to divergence.

### 2.4 Instabilities Caused by the Dependence of Melting Temperature on the Concentration

#### 2.4.1 Systems

The physical properties of the binary alloy tin containing lead was used for the first part of the studies here. The reverse system (lead containing tin) has been used extensively as a basis for linear stability analysis and experimental studies [59, 62, 129, 130, 266]. The low melting point of this alloy simplifies experimental studies, and its physical properties are well-known. The concentration of tin is lower than lead, and the tin which is rejected into the melt at the melt/crystal interface may destabilize the system.

The system tin containing lead was studied by Coriell and McFadden [60] because the lead that is rejected into the melt at the interface is heavier, and hence the solu-
tal field is stabilizing. These authors discovered that the system may be unstable to small perturbations even though both the solutal and temperature fields are stabilizing. These instabilities are caused by the inequality of the temperature gradients at the interface which drives flow as the interface is perturbed by lateral variations in concentration; this was described in more detail in Section 2.1.2. In the analysis below, the change in the density with concentration is neglected so that the solutal field has no effect on buoyancy. The effects of buoyancy are studied separately in Section 2.5.

The transparent succinonitrile-acetone alloy also is studied. This alloy and the similar succinonitrile-ethanol alloy have been used for theoretical and experimental studies in thermostolutal convection by many researchers [238, 238, 61, 63, 16, 170]. One reason for using this system is the transparency of succinonitrile which allows observation of the convective flow by tracking the motion of small, neutrally buoyant particles [238, 237]. Furthermore, succinonitrile solidifies with an unfaceted melt/crystal interface and has well-characterized physical properties.

Coriell and McFadden [60] first calculated the onset of instabilities that are explored here for the tin-lead alloy. The minimum neutral point of the stability curves occur at relatively large wavelengths for these instabilities (on the order of several centimeters). These wavelengths are comparable to the size of the ampoule in a vertical Bridgman crystal growth configuration. Linear stability analysis, as used by Coriell and McFadden [60] and others [203, 59, 65], involves the assumption of an infinite system (Fig. 2-9), or a physical system that is large compared to the wavelength of the perturbations. Because the wavelengths of the instabilities studied here are similar to the size of the ampoule in a Bridgman crystal growth configuration, the infinite system is not a good approximation. A cylindrical geometry with rigid boundaries is more appropriate. The infinite systems are studied because linear stability results are available for these systems, while cylindrical geometries with rigid boundaries are studied because these systems are better approximations for the real
2.4 Convective Supercooling

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kinematic viscosity</td>
<td>$\nu$</td>
<td>$2.6 \times 10^{-3} \text{ cm}^2/\text{s}$</td>
</tr>
<tr>
<td>molecular diffusivity</td>
<td>$D$</td>
<td>$2.1 \times 10^{-5} \text{ cm}^2/\text{s}$</td>
</tr>
<tr>
<td>coefficient of thermal expansion</td>
<td>$\beta_T$</td>
<td>$1 \times 10^{-4} \text{ K}$</td>
</tr>
<tr>
<td>thermal conductivity of the liquid</td>
<td>$k$</td>
<td>$0.3 \text{ J/cmKs}$</td>
</tr>
<tr>
<td>thermal conductivity of the solid</td>
<td>$k_s$</td>
<td>$0.6 \text{ J/cmKs}$</td>
</tr>
<tr>
<td>segregation coefficient of the solute</td>
<td>$\kappa$</td>
<td>0.12</td>
</tr>
<tr>
<td>slope of the liquidus curve</td>
<td>$m$</td>
<td>$-1.6 \text{ K/wt%}$</td>
</tr>
<tr>
<td>latent heat of fusion</td>
<td>$\Delta H_s$</td>
<td>$59 \text{ J/g}$</td>
</tr>
<tr>
<td>density of the liquid and the solid</td>
<td>$\rho$</td>
<td>$6.95 \text{ g/cm}^3$</td>
</tr>
<tr>
<td>heat capacity of the liquid</td>
<td>$C_p$</td>
<td>$0.254 \text{ J/gK}$</td>
</tr>
<tr>
<td>heat capacity of the solid</td>
<td>$C_{p,s}$</td>
<td>$0.270 \text{ J/gK}$</td>
</tr>
</tbody>
</table>

| Melting point of pure tin             | $T_m$ at $c_0 = 0$ | 505.1 K |

Table 2.5: Physical properties used for the tin-lead alloy [60, 66].

Crystal growth configurations. For a fixed wavelength of the disturbances $\lambda$, the infinitely wide system is modeled by a system of finite width equal to half the size of this wavelength $\lambda/2$ or any number of half wavelengths $\lambda/2n$, where the sidewall boundaries are chosen so that these boundaries are planes of reflective symmetry, as described in Section 2.2. This rectangular geometry with reflective sidewalls is studied in lieu of infinite systems.

The physical properties of the alloys studied are summarized in Tables 2.5 and 2.6. Coefficient of solutal expansion is set to zero in all cases to eliminate the effects of density dependence on the concentration. Therefore, the only coupling of the concentration field with other field equations is by virtue of the dependence of the melting temperature on the concentration. Also, the melting temperature of the solid succinonitrile-acetone system is modified to 0.00446 J/cmKs so that sufficient inequality in the temperature gradients exist for this system at the interface to cause instabilities at low concentrations ($< 0.1 \text{ mole\%}$).
Figure 2-13: Tin-lead stability diagram for rectangular geometries where the sidewalls are planes of reflective symmetry, based on the work by Coriell and McFadden [60]; \( V_g = 1 \mu/s \) and \( G = 200 \) K/cm. The number in parentheses associated with each curve denotes the number of waves for the disturbances which are introduced in the computational domain of fixed width as determined by the abscissa.
2.4 Convective Supercooling

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kinematic viscosity</td>
<td>$\nu$</td>
</tr>
<tr>
<td>molecular diffusivity</td>
<td>$D$</td>
</tr>
<tr>
<td>coefficient of thermal expansion</td>
<td>$\beta_T$</td>
</tr>
<tr>
<td>thermal conductivity of the liquid</td>
<td>$k$</td>
</tr>
<tr>
<td>thermal conductivity of the solid*</td>
<td>$k_s$</td>
</tr>
<tr>
<td>segregation coefficient of the solute</td>
<td>$\kappa$</td>
</tr>
<tr>
<td>slope of the liquidus curve</td>
<td>$m$</td>
</tr>
<tr>
<td>latent heat of fusion</td>
<td>$\Delta H_s$</td>
</tr>
<tr>
<td>density of the liquid and the solid</td>
<td>$\rho$</td>
</tr>
<tr>
<td>heat capacity of the liquid</td>
<td>$C_p$</td>
</tr>
<tr>
<td>heat capacity of the solid</td>
<td>$C_{p,s}$</td>
</tr>
<tr>
<td>melting point of pure tin</td>
<td>$T_m$ at $c_0 = 0$</td>
</tr>
</tbody>
</table>

* A value of 0.00446 J/cmKs is used for the thermal conductivity of the solid in most of the calculations; this is explained in the text.

Table 2.6: Physical properties used for the succinonitrile-acetone alloy [238, 16].

The wavelengths and wavenumbers in the discussion that follows are expressed in the dimensional units of cm and cm$^{-1}$, respectively, since it is more conventional [60, 59, 65], and also so that the absolute size of the disturbances is obvious.

2.4.2 Tin-Lead

Coriell and McFadden [60] present the neutral stability curves for the tin-lead alloy in the infinite domain shown in Fig. 2-9. This infinite system is equivalent to a series of systems of finite width equal to half the wavelengths ($\lambda/2$) of the disturbances studied, wherein the boundary conditions are chosen so that the sidewalls are planes of reflective symmetry. The appropriate boundary conditions which make the sidewalls planes of reflective symmetry are shear-free for the flow, adiabatic for the energy, and no-flux for the solute. Therefore, the neutral stability curves by Coriell and McFadden [60] can be applied to domains with rectangular geometries and with shear-
free, adiabatic walls.

The neutral stability diagram shows the critical concentration $c_{0_c}$ as a function of the wavenumber $\omega$. The basic state loses stability for a value of $c_0$ slightly larger than the critical onset ($c_{0_c}$) to a solution with a wavelength given by $\lambda = 2\pi / \omega$. A new solution family bifurcates at the critical onset. This family exhibits cellular character in its flow field, and has a wavy interface corresponding the wavelength of the disturbance $\lambda$. The system of finite width admits Fourier modes with spatial frequency $\omega$ and the harmonics $\{2\omega, 3\omega, \ldots\}$. Therefore, the critical concentrations on a neutral stability diagram for wavenumbers $n\omega$ where $n = \{1, 2, \ldots\}$ are the bifurcation points which correspond to solutions with interface shapes given by $n$ waves in the low-amplitude limit.

The neutral stability curves for systems of finite width are constructed by superimposing the curves $c_{0_c}(n\omega)$ on the same plot, where $n = \{1, 2, \ldots\}$ and $c_{0_c}(\omega)$ is the neutral stability curve for the infinitely wide system. This construction makes an easier framework for examining the stability of a system of finite width. The different curves $c_{0_c}(n\omega)$ determine the critical concentrations at the onsets of instability to disturbances with wavenumbers of $n\omega$. These onsets of instability also mark bifurcation points from the basic state to solution families whose interface shapes have $n$ waves in the low-amplitude limit, where $n$ is the same as in the $c_{0_c}(n\omega)$ curve. As the bulk concentration $c_0$ is increased for a system of fixed width, the order in which the $c_{0_c}(n\omega)$ curves are encountered determines the order in which the solutions whose interface shapes have $n$ waves bifurcate from the basic state solution. Furthermore, this superposition of $c_{0_c}(n\omega)$ curves reveals the points where two families bifurcate simultaneously from the base solution. These points correspond to two real eigenvalues passing through zero simultaneously and are called *codimension-two* points [128]. For the remainder of the discussion, the term "one-wave solution" is used to indicate the bifurcating solution family which has one wave in the melt/crystal interface in the low-amplitude limit; this family has one wave in the temperature and in the con-
2.4 Convective Supercooling

centration contours, and its convective flow consists of two cells. The terms two-wave solutions, etc. are defined similarly.

The first three curves \( \{c_{0c}(\lambda), c_{0c}(\lambda/2), c_{0c}(\lambda/3)\} \) for the tin-lead alloy are plotted in Fig. 2-13. As \( c_0 \) is increased, the curves in Fig. 2-13 mark the points of bifurcation from the basic state of one-wave, two-wave, and three-wave solutions; the number of waves is indicated by the curve. A codimension two bifurcation point occurs in Fig. 2-13 for the one-wave curve \((c_{0c}(\lambda))\) crossing the two-wave curve \((c_{3c}(\lambda/2)\) at \( \lambda_{12} \approx 5.4 \) cm. The one-wave solution bifurcates at a lower \( c_0 \) for \( \lambda < \lambda_{12} \) and the two-wave solution bifurcates at a lower \( c_0 \) for \( \lambda > \lambda_{12} \).

The bifurcation diagrams which follow indicate a measure of deviation from the base \( \varepsilon \) as a function of the inlet concentration \( c_0 \) such that

\[
| \varepsilon | \equiv \pm \| u \|_\infty
\]

(2.169)

where \( u \) is the lateral component of the velocity, and \( \| u \|_\infty \) indicates the maximum value of \( u \). The sign for \( \varepsilon \) is arbitrarily chosen to be positive for one-wave solutions whose convective flow rotates counter-clockwise, and also for two-wave solutions for which the flow is downward at the sidewall; the sign for \( \varepsilon \) is chosen to be negative for one-wave solutions whose convective flow rotates clockwise and for two-wave solutions for which the flow is upward at the sidewall.

Contours of the concentration and flow fields are plotted for sample solutions on the bifurcating families. The temperature fields show minute deviations from the state of unidirectional conduction; therefore, temperature fields are not shown. The reasons are weak convective flows and the low Prandtl number of the tin-lead alloy. Even though the nonlinearity of the temperature fields here is insignificant, for all practical purposes, the more pronounced nonlinear behavior in the concentration and the flow fields are important, and manifest themselves as lateral variations in the concentration of the solute in the crystal.

A typical mesh used in the tin-lead calculations is illustrated in Fig. 2-14; this
mesh has 20 elements in the vertical direction divided equally between the melt and solid and 20 elements in the horizontal direction resulting in about 5,000 unknowns. The location of the bifurcation points agree with the predictions by Coriel and McFadden [60] to better than 1% for $\lambda = 3$ cm. This excellent agreement attests to the adequacy of the mesh size. The agreement is better than 1% for the first bifurcation point of $\lambda = 6$ cm, and is to within 6% for the second bifurcation point of that wavelength. The lower accuracy is due to inadequate lateral discretization for the wider wavelength. It is not useful to refine the mesh any further since doing so only raises the computation time without adding more insight to the problem. The quality of the solution is clearly captured by this mesh; the comparison with the results in Section 2.4.3 show similar qualitative behavior for the succinonitrile-acetone alloy, for which the mesh is more rigorously refined.

Bifurcation diagrams are shown for $\lambda = 3$ and 6 cm in Figs. 2-15 and 2-16, respectively. These figures show the one-wave and two-wave solution families which
bifurcate from the basic state at the critical values of the concentration. As shown by Fig. 2-13, the distance between the bifurcation points to one-wave or to two-wave solutions is nearly an order-of-magnitude larger for \( \lambda = 3 \) than for \( \lambda = 6 \). The first bifurcating solution for \( \lambda = 3 \) cm (Fig. 2-15) has one wave and the first solution for \( \lambda = 6 \) cm (Fig. 2-16) has two waves, because \( 3 \text{ cm} < \lambda_{12} < 6 \text{ cm} \), as shown in Fig. 2-13.

Bifurcation in both cases (\( \lambda = 3 \) and \( 6 \) cm) is super-critical, as determined by symmetry of a rectangular geometry with shear-free, adiabatic sidewalls. The steady-state differential equations have no preferred spatial direction. The boundary conditions for this system are symmetric about a vertical line which bisects the domain. Transposing the domain about this vertical line does not change the boundary conditions at all. To see why, consider any solution that satisfies the equations and boundary conditions. A mirror image of any such solution has to satisfy the equations and boundary conditions because the inversion does not change the boundary conditions. Thus, every solution has a mirror image counterpart which deviates from the basic state by the same amount but in the opposite direction laterally. This implies a sub- or super-critical bifurcation.

No secondary interactions occur for the point \( \lambda = 3 \) cm (Fig. 2-15) up to a concentration of \( 0.1 \text{ wt}\% \), which is nearly two orders-of-magnitude beyond the critical concentration.

In contrast, the \( \lambda = 6 \) cm case shows interesting nonlinear interactions between the first and the second branches, as shown in Fig. 2-16. The second family of one-wave solutions turns around and connects to the family of two-wave solutions through a sub-critical bifurcation point.

Relaxing the insulated sidewall boundary condition causes an imperfection \([152]\) in the system, as shown by Hall and Walton \([131]\) for the Bénard convection. A linear ambient temperature is imposed on the sidewall based on Newton's law of cooling through a fixed Biot number, as described in Section 2.2. The imperfection breaks
Figure 2-15: The first two bifurcating solution families for the tin-lead alloy in a rectangular geometry with shear-free sidewalls; $\lambda = 3$ cm.
Figure 2-16: The first two bifurcating solution families for tin-lead alloy in a rectangular geometry with shear-free sidewalls; $\lambda = 6$ cm. The lower one-wave family connects to the upper two-wave family, as indicated by an open circle (no connection), or a closed circle (connection).
the bifurcation diagram (see Fig. 2-15). Even the small Biot number $\text{Bi} = 0.01$ breaks
the bifurcation dramatically. Therefore, even small heat leakages through the walls
cause much larger convection than that which the thermostolulal instabilities produce.

The effects of introducing no-slip boundary conditions at the sidewall were con-
sidered for the domain with $\lambda = 3$ cm; the computed bifurcation diagram is shown in
Fig. 2-17. As shown in the figure, only the one-wave solution family is tracked for this
case. The first bifurcation is trans-critical with the base state. However, the negative
branch reversed direction at a slightly lower concentration than the critical onset and
continues in the direction of increasing $c_0$. The whole one-wave bifurcating family,
including the critical onset point $c_{0c}$, seems to differ from the case for the shear-free
sidewall slightly with respect to the basic state solution.

Cylindrical systems with shear-free and rigid sidewalls were examined next, and
the computed bifurcation diagrams are shown in Figs. 2-18 and 2-19. Both cases
show trans-critical bifurcations. However, the system with no-slip boundaries shows
a connection between the first and the second branches through a limit point which
appears at a concentration of about 0.055 wt%. No such interactions exist for the
shear-free case up to concentrations of 0.08 wt%.

Examining the value of the critical concentration for the onset of instabilities for
various geometries and boundary conditions leads to an interesting observation. The
effects of rigid boundaries are slightly stabilizing; about 1%. The effects of geometry
are more pronounced; the critical onset for the cylindrical geometry is larger by about
10%. These observations are consistent with the conclusion of Guérin et al. [129, 130]
that the rigid sidewalls and cylindrical geometries have stabilizing effects; moreover,
a comparison of the results between the rigid sidewalls and cylindrical geometries
indicates that the effects of geometry are more pronounced.

All systems except the rectangular geometry with shear-free sidewalls show trans-
critical bifurcations. The symmetry of the rectangular geometry with shear-free side-
walls implies a sub- or super-critical behavior, as discussed above. Because the other
Figure 2-17: The first bifurcating solution family (positive side) for the tin-lead alloy in a rectangular geometry with rigid sidewalls; $\lambda = 3$ cm.
Figure 2-18: The first bifurcating family (negative side) for tin-lead alloy in the cylindrical geometry with a shear-free sidewall; $R = 1.5$ cm.
systems do not have this symmetry, the above symmetry argument does not apply. The asymmetry of the boundary conditions or of the geometry about the vertical bisecting line implies that the mirror image of a solution is not a solution; therefore, similar solutions with equal but opposite lateral variations are not guaranteed. It is not possible to draw conclusions about whether the bifurcation will be trans-critical. In an attempt to eliminate the possibility of mesh effects being an artificial source of trans-criticality, the number of elements was increased by 20% and solutions close to the first bifurcating point for the cylindrical geometry with rigid sidewalls were recomputed. The bifurcation remained trans-critical.
2.4.3 Succinonitrile-Acetone

In this Section, the succinonitrile-acetone alloy is studied. The thermal conductivities of the solid and liquid for this alloy are nearly equal, as shown in Table. 2.6. Also, because the effect of latent heat release is small, so is the inequality in the temperature gradients at the melt/crystal interface. Because this inequality is a required driving force for the instabilities studied in this Section, no instabilities of the base solution up to 0.1 mol% concentrations are observed. However, if the thermal conductivity of the solid is modified to twice that of the liquid, convective supercooling instabilities appear, and secondary nonlinear interactions occur. Coriell and McFadden [60] concluded that the ratio of the thermal conductivities of the tin-lead alloy plays an important role in the linear stability of the system. The results presented in this Section are for the modified succinonitrile-acetone alloy with the thermal conductivity of the solid set to twice the thermal conductivity of the liquid, in order to increase the inequality of the temperature gradients at the interface. Increasing the thermal conductivity ratio between the solid and the liquid increases this driving force. The results for the equal values of the thermal conductivities are presented in Section 2.5.

The growth rate and the temperature gradient in the liquid are set to $V_g = 1 \, \mu/s$, and $G = 200 \, K/cm$, as for the tin-lead alloy in Section 2.4.2.

The mesh used in the tin-lead calculations (Fig. 2-14) was refined for the succinonitrile-acetone alloy because of the smaller length of the exponential decay of the concentration due to lower solutal diffusivity. Two finer meshes were considered; these meshes are shown in Fig. 2-20. The medium mesh in Fig. 2-20 has 24 lateral elements and 14 vertical elements in the melt and 12 vertical elements in the crystal resulting in 8,000 unknowns. The finer mesh in Fig. 2-20 has 24 lateral elements and 18 vertical elements in the melt and 12 vertical elements in the crystal resulting in 10,000 unknowns. The connectivity of the bifurcation diagram did not change with mesh refinement even though the bifurcation points moved somewhat. For example, the location of the first bifurcation moved by 5% from the coarse (Fig. 2-14) to the medium
Figure 2-20: The medium and fine meshes used in calculations for the succinonitrile-acetone alloy; the coarse mesh is the one used for tin-lead; see Fig. 2-14.

mesh, and by 1% from the medium to the finer mesh. The results which are presented here are from the finer mesh.

The bifurcation diagram for $\lambda = 3 \text{ cm}$ is shown in Fig. 2-21. As shown by the neutral stability diagram of this system (Fig. 2-23), the sample width of $\lambda = 3 \text{ cm}$ is close to the codimension two point $\lambda = \lambda_{12}$. The family of one-wave solutions for $\lambda = 3 \text{ cm}$ bifurcates at a lower $c_0$ than the family of the two-wave solutions. The one-wave bifurcating family connects to the two-wave family through a sub-critical secondary bifurcation point. This structure is similar to the solution structure for the tin-lead alloy in the rectangular geometry with $\lambda = 6 \text{ cm}$, as shown in Fig. 2-16. In both systems, the one-wave solution family connects to the two-wave family through a sub-critical bifurcation point; however, this bifurcation point lies on the positive or negative side of the two-wave family depending on which family (the one-wave or two-wave) bifurcates from the base at the lower value of $c_0$. The two-wave solution family bifurcates at a lower value of $c_0$ than the one-wave family for the tin-lead alloy where $\lambda = 6 \text{ cm}$, as shown in Fig. 2-13. The one-wave solution family bifurcates before
Figure 2-21: Connectivity of the first two bifurcating families of the succinonitrile-acetone alloy in a rectangular geometry with shear-free sidewalls; $\lambda = 3$ cm. The upper one-wave family connects to the lower two-wave family as indicated by an open circle (no connection), and a closed circle (connection).
2.4 Convective Supercooling

![Diagram](image)

**Figure 2-22:** Interaction of the one-wave (1) and two-wave (2) solution families close to the codimension two point.

The two-wave family for the succinonitrile-acetone alloy in the rectangular, shear-free system with $\lambda = 3 \text{ cm}$, as shown in Fig. 2-23. The neutral stability curves for the two alloys look similar in that the one-wave solution family bifurcates at a lower $c_0$ than the two-wave family for $\lambda < \lambda_{12}$, and at a higher $c_0$ for $\lambda > \lambda_{12}$ in the neighborhood of the codimension two point. The following scenario explains the behavior close to the codimension two point, depicted in Fig. 2-22.

The codimension two point is approached from the left as $\lambda$ increases; $\lambda < \lambda_{12}$ and $\lambda \to \lambda_{12}$. The secondary bifurcation point on the two-wave solution moves closer to the base. Also, the onsets for the bifurcations to the one-wave and the two-wave solution families move closer to each other. At the codimension two point, the bifurcation points from the base and the secondary bifurcation point coincide. Only the two-wave solution family branches away from the codimension two bifurcation point; the one-wave family disappears. As $\lambda$ is increased beyond the codimension two point, $\lambda > \lambda_{12}$ and $\lambda - \lambda_{12}$ increases. The secondary bifurcation moves away from the base. However, in this instance the secondary bifurcation point is on the opposite side, with respect to the base, of the two-wave solution family on the bifurcation diagram.
Figure 2-23: Neutral stability curves for succinonitrile-acetone alloy in rectangular geometries with shear-free sidewalls. These curves were calculated by monitoring the sign of det($J$) in nonlinear calculations rather than by linear stability analysis. The number in parentheses associated with each curve denotes the number of waves for the disturbances which are introduced in the computational domain of fixed width as determined by the abscissa.
2.5 Thermosolutal Instabilities

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kinematic viscosity ( \nu )</td>
<td>( 2.6 \times 10^{-2} \text{ cm}^2/\text{s} )</td>
</tr>
<tr>
<td>molecular diffusivity ( D )</td>
<td>( 1.3 \times 10^{-5} \text{ cm}^2/\text{s} )</td>
</tr>
<tr>
<td>coefficient of thermal expansion ( \beta_l )</td>
<td>( 1.07 \times 10^{-3} \text{ K}^{-1} )</td>
</tr>
<tr>
<td>coefficient of solutal expansion ( \beta_s )</td>
<td>( 2.5 \times 10^{-3} \text{ wt}% )</td>
</tr>
<tr>
<td>thermal conductivity of the liquid ( k )</td>
<td>0.00223 J/cmKs</td>
</tr>
<tr>
<td>thermal conductivity of the solid ( k_s )</td>
<td>0.00223 J/cmKs</td>
</tr>
<tr>
<td>thermal diffusivity of the liquid ( \alpha )</td>
<td>( 1.14 \times 10^{-3} \text{ J/gK} )</td>
</tr>
<tr>
<td>thermal diffusivity of the solid ( \alpha_s )</td>
<td>( 1.14 \times 10^{-3} \text{ J/gK} )</td>
</tr>
<tr>
<td>segregation coefficient of the solute ( \kappa )</td>
<td>0.1</td>
</tr>
<tr>
<td>slope of the liquidus curve ( m )</td>
<td>(-2.8 \text{ K/wt}% )</td>
</tr>
<tr>
<td>capillary parameter ( T_m \Gamma )</td>
<td>( 6.2 \times 10^{-6} \text{ cmK} )</td>
</tr>
<tr>
<td>latent heat of fusion ( \Delta H_s )</td>
<td>48.38 J/g</td>
</tr>
<tr>
<td>density of the liquid and the solid ( \rho )</td>
<td>0.988 g/cm(^3)</td>
</tr>
<tr>
<td>melting point of pure tin ( T_m )</td>
<td>( c_0 = 0 \text{ 331.2 K} )</td>
</tr>
</tbody>
</table>

Table 2.7: Physical properties used for the succinonitrile-acetone alloy [63].

2.5 Instabilities Caused by the Dependence of Density on the Solutal Field

2.5.1 System

The system studied is succinonitrile containing acetone. Table 2.7 shows the thermophysical parameters used. These parameters are adopted from Coriell et al. [63] with the exception of the thermal conductivity and diffusivity of the solid which differ by 1% from the liquid values and are set equal to the liquid values. Therefore, there is inequality of temperature gradients at the interface only because of latent heat release \( \Delta H_s \). Instabilities are caused purely by the dependence of density on concentration for \( \Delta H_s = 0 \). Below, results are presented both for \( \Delta H_s = 0 \) and \( \Delta H_s \neq 0 \).
2.5.2 Steady-State Solutions

The calculations presented here are for the parameters given in Table 2.7 with one exception; the latent heat is set to zero. Other parameters are \( V_g = 4.02 \mu s \), and \( G = G_s = 20 \) K/cm. Because \( k = k_s \) and \( \Delta H_s = 0 \), there is no inequality at the interface for the gradients of the temperature in the liquid and in the solid; therefore, the instabilities occur only because of buoyancy. Furthermore, the temperature field is fixed as a result of these approximations, except for the contributions of convective heat transfer driven by the solidification velocity and by thermosolutal motion.

The neutral stability curves are shown in Fig. 2-24 for the critical acetone concentration as a function of the wavenumber \( \omega \). Continuous lines in Fig. 2-24 are results of the linear stability analysis, and discrete points are results of fully nonlinear simulations found by monitoring the sign of \( \det(J) \). The predictions of the linear stability and nonlinear calculations agree very well for the onsets of instability. Linear stability is easier to use and much more economical for predicting the onsets of instability; therefore, the onsets are found by linear stability analysis rather than by monitoring the sign of \( \det(J) \) as in nonlinear calculations. Moreover, monitoring the sign of \( \det(J) \) does not predict instabilities which correspond to oscillatory onsets, because the complex eigenvalues of the real matrix \( J \) appear in complex conjugate pairs which cross the real axis together.

The curves in Fig. 2-24 show the features described by Coriell et al. [59]. The convective portion of linear stability curve at low \( \omega \) consists of a branch for the oscillatory onset, which is depicted in dashed line, and of a closed loop for the steady onset, which is drawn in solid line. An oscillatory branch connects the closed convective loop with the morphological branch. The morphological portion of the curve at high \( \omega \) is the onset for steady instabilities and asymptotically approaches the prediction by Mullins and Sekerka [203] as \( \omega \) increases (Eq. (2.11)).

Nonlinear solutions in Fig. 2-25 are shown for bifurcation from the base at \( \omega = 25 \) cm\(^{-1}\) for \( c_0 = 0.0126 \) to 0.0134 wt\%. These parameters correspond to a very small,
almost indiscernible, vertical line which is indicated on the convective loop in Fig. 2-24. Even though the nonlinear states in Fig. 2-25 bifurcate off the convective branch very far away from the onset for the morphological instabilities, these states develop large interface deflections with only a small increase in $c_0$ above the critical value. The range of $c_0$ covered in Fig. 2-25 is from 2% below to 5% above the critical onset. States for higher values of $c_0$ could not be computed because of the steepness of the melt/crystal interface. The flow and concentration fields computed with each state also are shown in Fig. 2-25; the cellular structure of the flow has been emphasized by subtracting the uniaxial streamlines that correspond to the growth velocity.

The calculations shown in Fig. 2-25 demonstrate that the thermosolutally induced convection in the melt cannot be thought of as being decoupled from the interface shape. In fact, the interface deformation shown here is as large as those traditionally associated with finite amplitude solidification cells. The following is a possible scenario for formation of deep cells.

Nonlinear mode coupling between morphological and convective branches manifests itself through the large lateral segregations of the solute across the interface which are induced by the flow. The lateral segregation of the solute (acetone) increases with increasing $c_0$; the percentage differences in concentration across the interface for the four cases shown in Fig. 2-25 are: 0, 49, 72, and 93 percent respectively. It seems likely that a weak cellular flow causes uneven mixing of the solute, and leads to a high concentration as the flow moves away from the interface. This higher concentration makes the interface morphologically unstable locally, and leads to large interface deformation, which enhances the lateral solute segregation. Accordingly, convection and morphology couple to cause large interface deflection and solute segregation.

2.5.3 Oscillatory Instabilities

The calculations here are for the nonzero heat of solidification: $\Delta H_s = 48.38 \text{ J/g}$; other parameters are as shown in Table 2.7, $V_g = 4.02 \mu \text{s}$, and $G = 20 \text{ K/cm}$. The
Figure 2-24: Neutral stability curves computed for the succinonitrile-acetone alloy with vanishing heat of solidification.
$\frac{c_0}{c_{o,crit}} = 0.98 \quad 1.01 \quad 1.02 \quad 1.05$

Figure 2-25: Interface shapes, concentration and flow fields computed at $\omega = 25 \text{ cm}^{-1}$. 
Figure 2-26: Neutral stability curves computed for succinonitrile-acetone alloy.
neutral stability curves are shown in Fig. 2-26 for these parameters. The curves are similar to the ones for $\Delta H_s = 0$ (Fig. 2-24), but the convective loop has shrunk to a much smaller size. As a result, the oscillatory-convective branch, which connects the steady-convective loop to the morphological branch, lies below the morphological branch for a range of wavenumbers. The flat bottom of the morphological branch can be avoided by choosing the wavenumber such that the oscillatory onset occurs at a lower concentration than the minimum for the morphological branch. As the inlet concentration is increased slightly beyond the critical onset value $c_{0e}$, only one mode, the oscillatory, becomes unstable. For wavenumbers where the oscillatory branch lies above the minimum of the morphological branch, many morphological modes become unstable at a lower concentration because of the flat bottom of the morphological branch of the neutral stability curve.

A nonlinear oscillatory solution is computed at $\omega = 35$ cm$^{-1}$ for the inlet concentration at 1% above the critical value. The initial condition was chosen as the basic state solution with an addition of some random noise. Average interface concentration $<c>_i$ is shown as a function of dimensionless time in Fig. 2-27. The frequency of the oscillations ($f = 0.0031 - 2$ s$^{-1}$) agrees very well with the prediction by linear stability theory ($f_{LS} = 0.0032 - 4$ s$^{-1}$). The interface and flow field are shown in Fig. 2-28, in which the basic state flow field is subtracted to better show the cellular character of the flow. The interface and flow oscillate during a period. The flow is strongest when the interface is flat and stops when the interface reaches maximum deflection. The maximum interface deflection is large even though the inlet concentration is only 1% above the critical onset.

2.5.4 Interaction of Oscillatory and Morphological Modes

In this Section, the interaction of nonlinear solution families which evolve from the oscillatory and morphological instabilities are studied. The oscillatory branch in Fig. 2-26 joins the morphological at a higher concentration than the minimum of
Figure 2-27: Average interface concentration as a function of dimensionless time for an oscillatory solution.
Figure 2-28: Interface and flow field in a period for an oscillatory solution. The dashed contours signify counter-clockwise flow.
the morphological branch. Because the morphological branch of the neutral stability
curve has a flat bottom, many morphological modes of varying wavelengths become
unstable at lower concentrations than would oscillatory modes; and separating the
various modes and solution tracking becomes difficult. Furthermore, resolving all the
modes is not practical with a fine mesh because of the largely varying wavelengths.
Therefore, the thermocapillary parameter has been increased to 0.062 cmK to sharpen
the flat bottom of the morphological branch. The resulting neutral stability curves
are shown in Fig. 2-29. Bifurcating solutions are calculated at \( \omega = 24.5 \text{ cm}^{-1} \) for a
range of concentrations, which encompasses \( c_0 = 0.019 \) to \( c_0 = 0.069 \text{ wt\%} \), as shown
by the vertical dashed line in Fig. 2-29. An enlargement of Fig. 2-29 is shown in
Fig. 2-30. The instabilities for the wavenumber of \( \omega = 24.5 \text{ cm}^{-1} \) with increasing
inlet concentration occur as follows. First, a one-wave morphological mode becomes
unstable at \( c_0 \approx 0.031 \text{ wt}\%, \) which is point \( m_1 \) in Fig. 2-30. Second, another one-
wave morphological mode becomes unstable at \( c_0 = 0.043 \text{ wt}\%, \) which is point \( m'_1 \)
in Fig. 2-30. Third, at a slightly higher concentration, an oscillatory mode becomes
unstable, which is point \( o \) in Fig. 2-30. Fourth, a two-wave morphological mode
becomes unstable, which is point \( m_2 \) in Fig. 2-30.

The bifurcation diagram in Fig. 2-31 shows lateral segregation as a function of the
inlet concentration for \( \omega = 24.5 \text{ cm}^{-1} \) and \( c_0 = 0.019 - 0.069 \text{ wt}\%. \) The concentration
field and the flow field with the base flow subtracted are shown at selected points on
the diagram. The bifurcating solution family from the first one-wave morphological
mode is stable in Fig. 2-31. Calculations of the oscillatory bifurcating solution family
are unsuccessful because these solutions are not stable. Time integrations at concen-
trations slightly larger than the oscillatory onset converge to a solution on the stable
one-wave family rather than on the oscillatory family, as shown in Fig. 2-31. Average
interface concentration \( \langle c \rangle_i \) is shown in Fig. 2-31 as a function of dimensionless
time for time integration; \( \langle c \rangle_i \) begins by oscillating in time, but the amplitude
of the oscillations grows until finally the integration converges to a solution on the
2.6 Summary

Fully nonlinear simulations were combined with linear stability theory in this Chapter to study directional solidification of binary alloys. Nonlinear calculations illuminate the nature of the bifurcating solutions, the nonlinear interactions of the solutions, and the relative importance of various modes of transport. Linear stability analysis is used for predicting the onset of instabilities which mark points at which solution families with oscillatory and steady convective flows bifurcate. It was shown that labeling the various branches of neutral stability curve as convective or morphological based on what causes the instability may be misleading. The evolving nonlinear states look mutually indistinguishable for the presentations in Section 2.5; typically, they involve flow and show large interface deflections for bifurcations from both the so-called convective and morphological branches.

In the first part of this Chapter, instabilities and nonlinear interactions, which were due solely to the dependence of the melting temperature on concentration at the interface, were investigated. These instabilities typically corresponded to wavelengths which are similar to the ampoule dimensions in a crystal growth configuration. A tin-lead alloy was examined in rectangular and cylindrical geometries with rigid and shear-free sidewalls. Succinonitrile-acetone alloy also is studied. No instabilities were observed for a dilute \( c_0 < 0.1 \text{ mole}\%) \) succinonitrile-acetone alloy for which \( k \approx k_s \); however, this alloy behaves similarly to the tin-lead alloy when the melting point of the solid is modified so that \( k_s = 2k \), because the ratio of thermal conductivities between the liquid and solid is the strongest driving force for these instabilities. The systems show mildly nonlinear behavior in general. The effects of geometry and boundary conditions are small when \( c_0 - c_{0c} \) is small, but these effects become large
Figure 2-29: The onset of instability for succinonitrile-acetone with a capillary parameter of 0.062 cmK. The vertical dashed line shows the range of $c_0$ for which bifurcating solutions are shown in Fig. 2-31.
Figure 2-30: An enlargement of Fig. 2-29 qualitatively showing the behavior at the onsets of instability which are studied in the bifurcation diagram of Fig. 2-31 for $\omega_0 = 24.5 \text{ cm}^{-1}$.

as $c_0 - c_{0c}$ grows, and the behavior changes qualitatively. Because small boundary imperfections in the form of heat leakage break the bifurcations dramatically, it is not likely that these nonlinear states would be directly observable in experiments.

In the second part of this Chapter, instabilities and nonlinear interactions, which are caused by the dependence of the density on the concentration, were investigated. These instabilities typically corresponded to wavelengths which were smaller than the ampoule dimensions in a crystal growth configuration. The evolving states show highly nonlinear couplings between the interface morphology and the flow, which cause large interface deformations: as the interface deforms and a groove forms, the flow sweeps solute into the groove: large solute concentrations in the groove then lower the melting point, and as a result, the groove gets deeper; the deepening of the groove causes larger lateral variations in the concentration and in the temperature which drive a stronger flow. At this point, the interface deflection becomes so steep
Figure 2-31: The Bifurcation diagram for the range of $c_0$ corresponding to the vertical dashed line in Fig. 2-29.
that the single-valued representation \( y_i = h(x, t) \) breaks down even for concentrations only a few percent above \( c_{0c} \).

The temperature field does not deviate significantly from the base for any of the nonlinear states caused by the concentration field instabilities. The departure of the temperature field from the base is minimal, even for a succinonitrile-acetone alloy with a high Prandtl number. While imperfect temperature boundary conditions are the primary source for the lateral variations in the temperature field, large lateral segregations may occur because of flow. These segregations manifest themselves in the finished product as significant lateral variations in the crystal composition.
Chapter 3

Parallel Processing

The development of parallel and distributed processing marks a revolutionary change in the field of computations. There are physical boundaries (such as the speed of light) which impede performance by limiting the absolute speed of a single processor. More importantly, microfabrication technology has, and will continue, to limit the speed of available processors. As a result, the performance of single processor machines is limited, and the cost of their improvement skyrockets as the physical limitations are approached. With parallel or distributed computing, where several tasks are performed simultaneously, total time of the execution is reduced even though each processor may not be operating extremely fast. Although parallel processing offers enormous potential for improving computation speed, at the same time, it introduces many complications particular to it. The question of what the best algorithm is for application to a serial computer is compounded enormously in the case of parallel computations. One can envision many algorithms being designed to achieve the same purpose. Parallelization is highly dependent upon the machine and the algorithm, and since there is no simple way of determining the best architecture and the best algorithm combination, an enormous amount of literature has been produced in recent years on various aspects and uses of distributed computing.

The purpose of this chapter is to lay a foundation for solution of partial differential
equations by parallel computing. Economic motivations for parallel processing and its potential power are described in Section 3.1. The choices available for parallel solution of partial differential equations and the merits and demerits of each are discussed in Section 3.2. Finally, the concepts, measures, and desired goals of parallelism are presented in Section 3.3.

3.1 Parallel Processors

3.1.1 Overview and history

The available technology has shaped the science of computing throughout history [233]. The performance of the computers has been limited by the state of the art and the cost of the technology. Furthermore, the designers of computers are restricted in their designs by components and techniques that are available to them [278, 233]. For example, Charles Babbage conceived his Analytical Engine, which contained many features of a von Neumann computer. The Analytical Engine required mechanical parts but was never built because of the large cost required to make its precisely machined components.

Two features of electronics component technology have motivated parallel processing. One is the rapid decrease in cost of producing components in mass quantities once the technology has been established; it is much more expensive to advance the state of the art than it is to mass produce what is currently available. The second motivation for parallel processing is the ever increasing demand for speed, which requires improvements beyond what circuit speeds allow [239]. Recently, it has become clear to many that parallel computing is the key to meeting future demands for computations, both because it reduces costs, and by virtue of the new possibilities it affords.

The improvements in computing power have come about both by improving computer architectures, e.g. by the development of pipelining and vectorization, and by
improving algorithms, e.g. by the development of fast Fourier transforms (FFT) [57]. The mere use of distributed computing does not reduce the need for efficient algorithms as one might hope. In fact, the computational engineer has a much greater responsibility for efficient use of parallelism since a bad algorithm will degrade the performance of the computer much more seriously than it would for a serial computer! On the positive side, the gains of a good algorithm can be tremendous. In fact, algorithm improvements can surpass architecture improvements. The reason is that gains made by the algorithm can increase as the size of the problem is increased, whereas the gains by the architecture are fixed. An algorithm which reduces the computational complexity by \( f(N) \), where \( f \) is an increasing function of the problem size \( N \), reduces the computation proportionally to \( f(N) \), as \( N \) is increased. However, a computer that runs \( n \) times faster reduces the computation time by \( n \) times, regardless of the problem size [54]. For example, fast Fourier transforms (FFT) [57] improved the computational complexity by an order of \( N/\log N \) over the previous methods for transforming \( N \) data points; the savings by FFT grow with the increasing \( N \). Implementing FFT on a SUN SPARCstation 2 which runs 2.5 times faster than a DECstation 3100 reduces the computation time by 2.5, independently of \( N \).

Although practical use of parallel computers and distributed systems has only been realized over the last two decades, the idea of parallel processing is not very recent. Menabrea in 1842 [196] mentioned that several results could be calculated simultaneously in his correspondence with Babbage, even though he did not make it clear how this idea could be incorporated into the Analytical Engine.

The work of Babbage was not surpassed until the advent of electronics a century later. The first stored program computer was the Manchester Mark 1 in 1948 [169, 269]. Vacuum tubes were the active components of this computer, and it used a cathode ray tube (CRT) for its storage medium [88]. Although von Neumann suggested the idea of a square array of processors as a general automaton [268], the first hardware design of an electronic parallel processor is attributed to Unger [262]. Unger's
computer required state-of-the-art technology which was in the process of being developed. His machine was never built, but is nonetheless considered the ancestor of all subsequent two-dimensional grid architectures [233]. ILLIAC IV was the first parallel supercomputer of its time. It was delivered to NASA and was fully operational by 1975 [91]. With a speed of about 50 MFLOPS, it was among the fastest supercomputers then available; the Cray 1 was first delivered in 1976 with an estimated speed of 100 MFLOPS [138].

Since the time of ILLIAC IV, a variety of parallel computers have been produced. The first fine-grain parallel machine was the Massively Parallel Processor (MPP) with 16,834 processors [154] designed mainly for image processing applications for NASA. Today, there are a large number of parallel computers available with different architectures and designs. Numerical analysts use a variety of distributed memory machines such as the more popular Intel hypercubes, Paragon, and Touchstone Delta with tens to hundreds of processors; the Thinking Machines' CM-2 and CM-5 with tens to thousands of processors; and other parallel machines such as Alliant, Fujitsu, and MasPar. Some of these computers boast speeds of tens of GFLOPS, and manufacturers promise speeds measured in tera FLOPS by the end of the decade. Although some architectures are more suited to certain applications, the optimal architecture for a problem is not clear in many cases.

This difficulty is compounded by many factors. For instance, there are usually several possible algorithms for the same problem, and careful implementation of an algorithm will make large differences in savings. The fraction of peak speed possible to achieve depends on the availability of fine-tuned numerical routines to take advantage of processor architecture, and on the availability of good compilers to produce efficient object codes.
3.1.2 Computer Architectures

Parallel computers can be categorized in many ways; for example, according to the network for interconnection of the processors, memory access, processor operation, and number of processors. This section touches upon some of these classifications and describes various methods for parallelism. For more thorough discussions the reader is referred to Desrochers [76] and Dongarra et al. [79].

Parallelism within a Single Processor

Usually one associates a parallel architecture with a network of several self-contained computing units, each comprising a central processing unit (CPU) and memory. A broader definition describes parallelism as a method of using replicated processing elements including parallelism within a single processor, such as is achieved by multiple functional units, pipelining, overlapping, and vectorization. Such a broader definition is described below.

The use of *Multiple functional units* was one of the first approaches to parallelism. Here, the arithmetic logic unit (ALU) is split up into sections, for example a floating point multiplication unit and an addition unit, which work concurrently. The software, e.g. compiler, is responsible for taking advantage of the multiple functional units effectively.

*Pipelining* is parallelism taken a step further and involves division of a functional unit into smaller segments. Each segment is responsible for partial decoding or execution of the process. Pipelining is similar to an assembly line process where each segment is processed in parallel with others. This process is demonstrated in Figs. 3-1 and 3-2. Once the first set of available data has passed through the first stage of the pipeline, having been successfully operated upon there, it passes to the next stage of the pipeline, freeing the first stage to operate on the next available set of data. *Overlapping* is similar to pipelining but somewhat more sophisticated in the sense that one of the following conditions may occur: function evaluations may have dependencies,
they may require a different sequence of stages, the stages may be relatively distinct in their purpose, or the time per stage may not be necessarily constant. In *Chaining*, the results of a pipelined process are directed into a second pipelined process before the entire first operation is complete. Therefore, the second operation starts just as the results from the first operation are leaving the unit.

*RISC*, the acronym for a *Reduced Instruction Set Computer*, was developed to augment efficiency by simplifying the processor [74]. A RISC processor can execute instructions at the rate of one per clock cycle with only basic instructions available at the processor level. RISC processors are naturally suited to the use of pipelining since pipelining breaks the functional units into stages which take one clock cycle to execute. *VLIW* (Very Long Instruction Word) architectures are RISC architectures with a large number of parallel pipelined functional units controlled by a single thread. The compiler schedules the tasks. This fine-grained control of the many functional units requires very large instructions, hence the name of the architecture.

*Vectorization* is an operation that is carried out on a set of operands. The operands are an array of scalar values which are loaded into a pipe and operated on simultaneously. Vectors too large to fit into vector registers require software fragmentation, or *stripmining* [79]. The compiler introduces an outerloop to break the vector operations
Figure 3-2: Piplined execution of an $N$-step process \[79\].

into segments which can be accommodated by the vector registers. There is a startup
time for vectorization in addition to the operation time which depends on the length
of the vector. Because of the overhead caused by the startup-time, vectorization in-
creases the speed more when the vectors are longer. The speed for vector operation
is usually

$$R = \frac{R_\infty}{N_{1/2}/N + 1}$$  \hspace{1cm} (3.1)

where $R$ is the speed for vector operation of length $N$, $R_\infty$ is the maximum vector
speed for a very long vector, and $N_{1/2}$ is the vector length at which half of the
maximum speed is reached \[137\]. Equation (3.1) produces a good approximation
when stripmining effects and memory bank conflicts are negligible.

Parallelism with Multiple Processors

Multiple processor parallel machines are categorized according to the memory con-
nection and hierarchy, the control of the processing elements, the interconnection
topology, and the grain size. These classifications are described below.
The terms *Distributed memory* or *shared memory* signify whether each processor has its own memory or shares a global memory with other processors. Shared memory processors, like those made by Cray, access the global memory through a data bus; see Fig. 3-3. The bus configuration is simple to construct and allows easy expansion of the memory, but memory access is shared and bus contention can degrade the performance because of limited bandwidth. Local memory and cache are used in most instances to relieve bus contention. Memory access is hierarchical in *distributed memory architectures* and therefore involves no contention. The processors must nonetheless communicate to exchange data because of the aforementioned memory structure hierarchy. The communication, which includes startup time and the time needed to send the message, is usually slow compared to computation, a factor which introduces a bottleneck which must be minimized by the use of distributed algorithms.

*Interconnection topologies* subdivide the class of distributed memory concurrent
3.1 Parallel Processors

systems. Processing elements need to be connected in some way for the exchange of information since the memory is not global. The network consists of physical connections and switches, and a router directs the message passing. If a physical connection exists between the processor originating the message and its destination, the path of the message is direct. If a physical connection does not exist, the message has to be routed through other processors. Usually, the hardware design minimizes the interruption of the processors along the message path for indirect messages. The possibilities for the interconnection network increase tremendously as the number of processing elements increases. Various interconnection architectures include mesh, three-dimensional mesh, ring, and hypercube; the processing elements also can be connected through high speed switches; see Fig. 3-4 for schematic representations of these topologies.

The ideal network is one in which each processing element is connected to all other processing elements. However, a large number of processors requires an even larger number of connections; this, in addition to hardware limitations and mounting expense, make such design highly impractical. Hypercube architectures minimize the number of connections while maximizing connectivity, theoretically an ideal network. For a while, hypercube architectures were considered the optimal choice. However, many manufacturers who started by designing hypercube architectures decided to build other architectures instead. Intel's new products, Paragon and Touchstone Delta, have mesh topologies, and Thinking Machines CM-5 has a fat-tree interconnecting network. The earlier products of both companies, Intel iPSC's and Thinking Machines' CM-2, had hypercube structures. The choice of optimal architecture is not an easy one and depends on many factors. The interconnection architecture is important to the programmer because the algorithms need to take advantage of the machine topology to minimize communication. For example, a binary reflected gray code is used to map the subdomains of the decomposed problem onto adjacent processors on the hypercube architecture used in Chapter 4 of this thesis. This algorithm
Figure 3-4: Common interconnecting networks of parallel computers.
is optimized for a hypercube topology.

The level of autonomy of each processor, called *Processor control*, is another way of classifying multiprocessor computers. Two popular classes are single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD). The processing elements of a SIMD machine can either perform the same instruction, each on its own set of data, or remain idle. The processing elements of a MIMD machine can perform independently of each other, and therefore, can execute different instructions simultaneously. Usually, the processing elements of SIMD machines are simpler, slower, and have less memory; therefore, each single processor is cheaper. MIMD processors are more sophisticated and more expensive because each processor must be able to operate independently.

Programming environments are radically different for these two cases. All processors are controlled by a single program in a SIMD environment, whereas each processor runs its own program in a MIMD environment. Usually, MIMD programs are replicas of a single program, which are loaded onto each processor, and operate independently. Messages are passed and processors are synchronized as needed.

A MIMD structure may seem more attractive because it has fewer limitations, but practical issues need to be considered: two examples are message passing among the processing elements, and synchronization of various tasks. A SIMD machine is obviously much more restrictive, but both programming and debugging are easier on it. On the contrary, debugging in a MIMD environment is quite complicated. Parallel debuggers simplify debugging to some degree by allowing examination and control of individual processes during a debug session.

*Grain size* categorizes a parallel computer based on the number of processing elements. *Coarse grain* refers to the machines with less than ten processors; *medium grain* refers to ones with between ten and several hundred processors; and *fine grain* refers to machines with hundreds to thousands of processors. These terms are used loosely, and the boundaries are fuzzy for the definitions.
Grain size is important for several reasons. Solving a problem in parallel requires division of the tasks for the solution among the processors. Usually, the processors need to communicate to exchange data in various stages of their work. An algorithm consists of a finite number of tasks. Independent tasks are identified based on the order in which they need to be executed. Maximum parallelism can be achieved by executing all independent tasks concurrently. In practice, this may be impossible because of physical limitations. The most restrictive physical limitation is the speed of communication. Finer grains require more communication for a given problem, and communication is undesirable because it is slow. At some point, the delays caused by communication offset the speed gained by parallelism. Problems which are inherently more parallel usually require less communication and can be implemented on finer grain architectures.

Another practical consideration is the level of control and sophistication of individual processing elements for architectures of various grain sizes. Coarse grain architectures can afford more expensive processors which are faster and have more memory. The processors of fine grain machines are usually not as fast and have less memory. The level of processor sophistication, control, and power directly affects programming and algorithm construction.

3.1.3 Potentials and Economics of Parallel Processing

One very important motivation for developing parallel processing is its potential for expanding the limits of feasible simulations. Another equally significant motivation is reduction of the costs of computations. The demand for complex simulations is pressing, and limited computer power has been an obstacle for realistic simulations of many processes. New technology that enables fast, efficient networks to connect several processing elements has led to direct increases in computing power. While, unfortunately, the improvement of single processing elements has been difficult; and expensive, mass producing the existing processing elements has become cheaper. All of
these factors cause computational scientists to look to parallel processing as the future standard in the field. The trend toward parallel processing is expected to continue regardless of improvements in single processors because improved single processors will be viewed by the proponents of parallel processing as potential components for parallel machines; machines which can directly increase the peak speed of any single processor counterpart. Concurrent machines are faster and cheaper in theory, but to use them efficiently is far from trivial and requires extensive effort; nevertheless, it is only a matter of time until capable parallel algorithms and methods are widespread.

Fischer and Patera [96] discuss the motivations for parallel processing through computational costs. The direct computational cost $C_{\text{dir}}$ is the cost of purchase and maintenance of a computer. The indirect computational cost $C_{\text{ind}}$ is the cost associated with the turnaround time; $C_{\text{ind}}$ is inversely proportional to the elapsed time for a computation. Suppose one has a processing element and makes machine $i$ with one replica of this processing element and parallel machine $ii$ with a network connecting $P$ replicas of this processing element. Then, consider a problem which can be solved with either machine $i$ or $ii$. The direct costs are lower for solving the problem on machine $i$. The reasons are that machine $ii$ can at most be $P$ times as fast as machine $i$, but it costs at least $P$ times as much as machine $i$. On the other hand, the indirect costs of solving the problem on machine $ii$ are lower by inverse speedup achieved on machine $ii$; e.g. if machine $ii$ speeds up the solution by a factor of $P$, as would be ideal, $C_{\text{ind}}$ is reduced by a factor of $P$. The above discussion assumes that it is possible to fit the given problem on machine $i$. In practice, machine $ii$ has $P$ times the memory of machine $i$ and can fit problems which machine $i$ cannot fit. The ability to fit a problem into computer memory is not usually emphasized enough in discussions of parallel processing, but it is extremely important since it is an indispensable prerequisite for solving a problem.

Now, suppose the processing element in machine $i$ is replaced with a more powerful processing element to bring the peak speed of machine $i$ up to the peak speed of
machine $ii$. Call this new computer machine $I$. In that case, the indirect costs are similar for both machines $I$ and $ii$, if the speedup on machine $ii$ is nearly optimal; in other words, suppose it takes nearly the same amount of time to solve the problem on either machine provided that the code on machine $ii$ utilizes all $P$ processors effectively. The indirect cost of machine $ii$ would be higher for speedups lower than the optimal $P$. Nevertheless, the direct cost is higher for machine $I$ especially if the processors of machine $ii$ are using state-of-the-art technology because improving them will be particularly costly to yield the processor of machine $I$.

Thus, if the major concern is only to reduce the direct costs of solving a problem, the problem should be solved on a low-cost single-processor machine. The only remaining concern would be fitting the problem into the memory. To reduce the indirect costs of solving a problem requires use of a faster, more expensive machine. A parallel processor is the best choice because its direct costs are lower than those of a single processor machine of equivalent speed.

In practice, varying the machine would be impractical and overly costly. Parallel computers allow the user to vary the number of processors used for a computation because only a subsection of the whole machine need be used. This is a choice, albeit limited, somewhat similar to choosing a different machine for different problems or for various purposes. For example, in a parametric study typical of engineering and scientific calculations, the problem size may vary considerably. The variation in problem size could be large, from the coarse meshes used to capture the behavior of solutions with smooth variations, to very fine meshes used to resolve sharp changes and boundary layers. As the problem size grows, more processors are employed to fit the problem and to reduce the time for computation. Indirect costs are not thereby significantly increased. Moreover, direct costs are reduced compared to a single processor machine with similar indirect costs. Therefore, the real advantages of parallel processing are that it enables solution of larger problems in reasonable times, i.e. without significantly increasing the indirect costs, while minimizing the
increases in direct costs.

In addition, the increased processing power of parallel computers, including greater memory and speed, enables more realistic computations which include more degrees-of-freedom, spatiotemporal dimensions, and complicated physical phenomena. Therefore, parallel processing can reduce the costs of currently feasible computations by allowing more simulations, and more thorough parametric studies; it can also make possible the solution of more complicated and realistic mathematical models with greater resolutions. The only development needed to obtain these goals is a set of algorithms capable of solving a wide variety of problems efficiently on parallel computers.

3.2 Solving Partial Differential Equations in Parallel

Construction of a parallel solution procedure for a mathematical model involves the selection of a parallel computer architecture, a method of discretization in space and time, methods for solution of nonlinear and linear systems of equations, schemes for division of various tasks among individual processors, and the management of data traffic among the processors. Several options for various parts of the solution procedure are discussed below. These options affect each other. For example, a computer architecture must be adopted as a model, and the general task distribution and data management of the algorithm must be addressed. The choice of computer architecture influences the selection of task distribution and data management. Conversely, a given task and data distribution scheme is usually more suitable for a particular computer architecture. It is not feasible to systematically explore all of the options because the number of choices is enormous and analyzing each set of options is an arduous task. Because the nonlinear coupling of the choices is not clearly understood, a mathematical model does not exist for performing rigorous global optimization to
Figure 3-5: An efficient serial solution procedure for solving transport equations in directional solidification.

arrive at a set of best choices; a situation which, were it achievable, would obviate the need to explore all possible options.

A mathematical model for a physical process is usually a set of Partial differential equations expressed in the general form of

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{F}(\mathbf{u}(\mathbf{x}, t)) \quad (3.2)$$

where \( \mathbf{u}(\mathbf{x}, t) \) is the vector of field variables, \( \mathbf{x} \) is the position vector, \( t \) is the time, and \( \mathbf{F} \) is a nonlinear differential operator. This equation is defined on a domain \( \Omega \) for some time, with initial conditions at \( t = 0 \) and boundary conditions on the boundaries of the domain \( \partial \Omega \). Solution of (3.2) requires a full simulation in space and time. Sometimes only the \textit{steady-state} solutions are required; in that case, the time derivatives are zero and a spatial simulation for \( \mathbf{F}(\mathbf{u}(\mathbf{x})) = 0 \) suffices.
A serial steady-state solution procedure for Eq. (3.2) using a finite element/Newton algorithm is shown in Fig. 3-5. Extensions to time-dependent equations are possible by temporal discretization and by adding an overall time integration loop in the solution procedure which is shown Fig. 3-5. This procedure has been used successfully for solution of transport equations for two-dimensional and axisymmetric transport phenomena in directional solidification (Chapter 2) and other complicated simulations including viscoelastic fluid flow (King [164]).

As shown in Fig. 3-5, partial differential equations are discretized and boundary conditions are applied with finite elements. For steady-state problems, these steps result in a nonlinear system of algebraic equations, written as

$$F(u(x)) = 0$$ (3.3)

where the same notation $F$ is used for the now discretized nonlinear differential operator. Newton's method is used to solve this nonlinear set through iteration as

for $k = 1 \rightarrow$ convergence or $k_{max}$

$$\text{solve } F^k_u \delta u^k = -F^k$$

$$u^{k+1} = u^k + \delta u^k$$

where $F^k_u$ is the Jacobian matrix at step $k$ with the entries $\{F^k_u\}_{ij} \equiv (\partial F_i/\partial u_j)^k$, and $\delta u^k$ is the correction vector to the solution at step $k$. Newton’s method is simply derived from a Taylor series expansion of the residual by keeping only the linear terms

$$F^{k+1} \simeq F^k + F^k_u \delta u^k = 0$$ (3.4)

This method converges quadratically when the initial guess $u_j^0$ is close to the actual solution. A linear system of equations needs to be solved at each Newton iteration step

$$F^k_u \delta u^k = -F^k$$ (3.5)

Serial frontal techniques [153, 139, 140] are used with auxiliary storage methods for
formulation and solution of the linear equation sets on vector supercomputers. These methods are amenable to efficient vectorization and are applicable to large problems because of the use of auxiliary storage. Convergence of the Newton iteration results in a solution which converges with mesh refinement to the actual solution provided that the numerical discretization is consistent and stable for the problem. Designing an efficient parallel counterpart for the finite element/Newton algorithm requires careful reevaluation of this scheme, which necessitates major revisions to various parts of the sequential algorithm.

3.2.1 Hierarchical Algorithms

Hierarchical algorithms and computer architectures are favorable for parallel solution environments because of the hierarchical nature of differential equations describing physical phenomena [43]. Equations (3.2) are local in nature. Evolutions of field variables at a point depend on the values of variables at the near proximity because of the local nature of the derivatives. This stems from the fact that the transport equations are continuous asymptotic models of discrete, locally interacting particles such as molecules and ions. On the other hand, information propagates in the whole domain with various time scales which are usually widely different. This information propagation is instantaneous for elliptic partial differential equations, such as the steady-state version of Eq. (3.2), wherein the solution depends at each point on the whole domain. Therefore, the solution of the transport equations are globally connected even though the equations themselves are local. This suggests a hierarchical nature for the transport equations in space. The hierarchical nature has direct implications for parallel solution of the transport equations. Physical domain can be divided into smaller sections, the equations can be solved partially in each small domain, and the partial solutions can be patched up (quilted) to complete the global solution. The above hierarchical approach is the essence of domain decomposition, which provides parallelism

---

1 Long range interactions such as radiative transport and viscoelastic behavior are neglected.
from a global viewpoint. There are other factors which affect domain decomposition besides the hierarchical nature of differential equations, namely problem geometry and discretization techniques. The former cannot be changed, but there are various possibilities for the latter as discussed in Section 3.2.2.

Two issues are vital for efficient division of the domain among processors regardless of the numerical method being used. The first issue is load balancing. If the processor loads are not similar, some processors will be idle for some time; this deteriorates the efficiency of the computation. The second issue is minimizing communication. Since processors need communicate in order to quilt their local solutions, and since communication is slow, quilting deteriorates efficiency and should be minimized as much as possible. Optimal domain decomposition is not an easy problem to solve. In fact, it could dominate the solution time. Because of these difficulties, various heuristic methods are used. For example, a heuristic algorithm is used in Section 4.3.3 for partitioning rectangular domains by nested dissection. Heuristic algorithms have been developed for automatic partitioning of more complicated two- and three-dimensional domains, for example by Vidwans and Kallinderis [267] and by Vavasis [264].

3.2.2 Discretization

Whereas numerical solution of Eq. (3.2) requires discretization in space and time, only discretization in space is needed for steady-state problems. If the numerical discretization is consistent and stable, the numerical solution converges to the actual solution as the mesh is refined, that is, by increasing the number of the grid points or unknowns. Typically, a convergent discretization technique is selected based on various considerations including the rate of convergence, the computational complexity for a given accuracy, parallelizability, area of stability, adaptability, generality, and ease of implementation. Various convergent methods exist for spatial and temporal discretization. For detailed discussion of these methods, the reader is referred to: for finite difference methods Dahlquist et al. [69], for finite element methods Carey and
Oden [36] and Finlayson [93], for finite volume methods Wirz and Smolderen [271], and for spectral element methods Gottlieb and Orszag [125].

Usually, spatial discretization is tightly coupled with domain decomposition for parallel implementation. Time integration is sequential because of the hyperbolic nature of the time derivatives in Eq. (3.2). Therefore, the geometric domain is discretized and decomposed among the processors. The Spatial part of simulation is performed in parallel, and time integration is performed sequentially.

The following simple example illustrates domain decomposition, time integration, and a parallel solution algorithm. This example is highly parallel, interesting, and educational, but it is also very problem specific and impractical for most real-world applications.

Consider the heat equation in three dimensions defined in a unit cube $D$ as

$$
\frac{\partial u}{\partial t} - \nabla^2 u = f(x, y, z, t) \quad (3.6)
$$

$$
u(x, y, z, t = 0) = u_o(x, y, z) \quad (3.7)
$$

$$
u(x, y, z, t) = 0 \text{ on } \partial D \quad (3.8)
$$

where $\partial D$ is the surface of the cube, as shown in Fig. 3-6. Consider the second-order finite difference scheme to approximate the Laplacian, and the forward Euler method to discretize the time derivative. Taking the spatial increment $h (= 1/N)$ to be the same in all three directions results in the following discrete equations

$$
u_{ijk}^{n+1} = \nu_{ijk}^n + \frac{k}{h^2} \left[ 6\nu_{ijk}^n - (\nu_{i+1,jk}^n + \nu_{i-1,jk}^n + \nu_{ij+1,k}^n \
+ \nu_{ij-1,k}^n + \nu_{ij,k+1}^n + \nu_{ij,k-1}^n) + h^2 f_{ijk}^n \right] \quad (3.9)
$$

$$
u_{ijk}^n = 0 \text{ for } i, j, \text{ or } k = 1 \text{ or } N + 1 \quad (3.10)
$$

$$
u_{ijk}^0 = u_{o,ijk} \quad (3.11)
$$

where $k$ is the temporal increment, $x = (i - 1)h$, $y = (j - 1)h$, $z = (k - 1)h$, $t = nk$,,
Figure 3. Schematic of the three-dimensional heat equation in a unit cube.
and \( u_{ijk}^n \) are the values of the unknown \( u \) at the time step \( n \) and at the point in space whose position is determined by the \( ijk \) indices.

Consider the application of a SIMD machine with fast data transfer between the nearest neighbor processors and a three-dimensional mesh interconnectivity of processors to solve this problem. To begin, each point in space is assigned to either one corresponding processor in the three-dimensional mesh, or, if the number of points is large, to a virtual processor, and the dependent variable \( u \) is initialized. With the low-order finite difference scheme for spatial discretization and the explicit time integrator, the solution at the next time step is written explicitly in terms of the solution at the present step, and no systems of equations need be solved to evaluate the solution at the next time step. Furthermore, the solution at each point depends on its nearest neighbors, or its nearest neighbors and the next nearest neighbors, depending on the order of accuracy of the finite difference scheme. Thus, if the communication between the nearest neighbors is fast, time integration can be performed very efficiently and quickly.

The solutions at successive time steps are computed using Eq. (3.9), while the values of \( u \) along the boundaries are kept equal to the prescribed boundary conditions. The solutions at different points for each time step are found in parallel with the following calculations and communications. Each processor gets the value of \( u \) from its six neighbors. The forcing function \( f \) is evaluated. Then, nine additions and three multiplications are performed to evaluate the solution at the next time step. The computation time needed to evaluate \( u \) at the next time step is the time needed for evaluation of \( f \) at one point in space and time, plus the time for six communications, three multiplications, and nine additions. Assuming that each communication, addition and multiplication take the same amount of time, the time to reach the next time step will only be the time one processor takes to do 18 floating point operations plus however many operations are needed to evaluate the function \( f \).

Even though this method fails to generalize and becomes more inefficient as the
problem, and hence the mesh, becomes more irregular, it is useful because of its vast potential to parallelize. Note that this method would still be applicable if there were a nonlinearity in $u$ in the original differential equation as long as the time derivative were not affected. For example, adding a $u^2$ term on the left side of Eq. (3.6) would add a corresponding term to the right side of Eq. (3.9), requiring two more operations per time step.

Investigators have used methods similar to the one above, their degree of success depending on the problem. Saati et al. [235] solved time-dependent compressible Navier-Stokes and energy equations in two dimensions on a Connection Machine (CM-2), and reported a performance rate of 1.42 GFLOPS. These authors used a finite difference scheme for spatial discretization and a predictor/corrector scheme for time integration. Farhat et al. [89] performed transient finite element calculations on a CM-2 with an explicit central difference algorithm for time integration; however, a large degree of mesh irregularities reduced the performance to about 400 MFLOPS.

Unfortunately, explicit methods similar to the one described above are not efficient for most applications. Explicit time integration does not work well for solution of transport problems because of the largely varying time scales involved in the physical problems. For example, the continuity equation for an incompressible fluid has no time derivative; therefore, it has an infinitely short time scale. The velocity, temperature, and species equations each have time scales which vary largely from problem to problem; these problems are numerically stiff [104]. Numerical stability limits the size of the time step to prohibitively small increments for stiff transport problems. Low-order implicit time stepping schemes, such as backward Euler and trapezoidal rule, have larger stability regions and are used instead and with better results for these problems. Implicit methods necessitate the solution of either linear or nonlinear systems of equations.

It is easier to implement finite or spectral elements than finite differences for problems with arbitrary geometries and moving boundaries. Moreover, boundary condi-
tions are thereby more smoothly incorporated. Finally, finite and spectral elements are much better suited to parallel implementation. For finite and spectral element methods, the coupling across elements occurs only through element borders which do not increase in width with the increasing order of interpolating polynomials. Increasing the interpolation order for finite differences increases the coupling throughout the domain. More local coupling has direct implications for parallel implementation and domain decomposition. If the domain is divided along element boundaries, the width of the quilting segments, where the processors need to patch their solutions, is minimal. On the other hand, the width of the quilting segments is a function of the order of interpolation in the case of finite difference discretizations; see Fig. 3-7. In general, then, finite differences require more communication to quilt the subdomains than do finite elements for a solution with the same accuracy. Therefore, finite differences are less desirable for parallel implementation.

The parallel method presented in Chapters 4 and 5 uses finite and spectral element methods. These methods are closely related. Finite element methods increase the number of elements, and spectral elements increase the order of the interpolating polynomial, as shown in Fig. 3-8. Both refinements increase the accuracy of the solution. Increasing the number of elements is called $h$-refinement, and increasing the polynomial order is called $p$-refinement. Spectral elements are particularly suited to parallel implementation because they maximize connectivity within the subdomains of the decomposed domain while minimizing the size of the quilting segments. Spectral elements are also advantageous when high accuracies are desirable because these discretizations lower the computational complexities over the finite element discretizations. If refining the polynomial order does not improve the quality of the solution because the solution is not smooth, it is necessary to increase the number of elements for increasing accuracy. The best results in such a case are obtained by a combination of $h$- and $p$-refinements. This combination corresponds to both increasing the number of elements and the order of interpolating polynomials for increasing the accuracy.
Figure 3-7: The width of the two subdomains’ shared boundary with the increasing order of interpolation for finite differences and finite elements.
of the solution. A yet more powerful algorithm is obtained by adding refinement which refers to moving the mesh nodal points. The advantages of multi-refinement are discussed thoroughly by Oden [211].

### 3.2.3 Linear Systems of Equations

**Data Structure and Sparsity**

In most instances, numerical solution of partial differential equations requires solution of linear systems of equations of the form

\[ Ax = b \]  \hspace{1cm} (3.12)

where \( A \) is the coefficient matrix, \( x \) is the vector of unknowns, and \( b \) is the vector of knowns (the right side vector). Storage of these linear systems uses most of the available memory, and their solution takes up most of the computational time. Therefore,
3.2 Solution of PDE’S

A good PDE solver must have a fast and robust solver of linear systems. The linear solver must use the properties of the linear system to manage the memory usage efficiently and to reduce the computational requirement. Furthermore, the linear solver needs to be robust and must apply to a large class of real problems.

Properties of linear systems result from problem physics and geometry (equations and boundary conditions), discretization technique, and the ordering used for arranging the equations and unknowns in the linear equation set. One of the most important properties of linear systems is their sparsity. Sparsity refers to the zero and nonzero pattern of A in Eq. (3.12). Generally, A has a large number of zeros which have a known pattern. Avoiding unnecessary storage of the zeros in A reduces the memory requirement, as well as the amount of computation and communication; if the zeros are not stored, they do not participate in any computation or communication. Depending on the method of solution, the sparsity pattern of A may or may not change during the solution of the linear system; for example, direct solution by LU-decomposition of A introduces fill-in; i.e. some zero entries become nonzero. The extent to which the sparsity is destroyed varies depending on the ordering of the equations. Ordering refers to the sequence in which the nodal or grid points, hence equations and unknowns, are numbered. The following example shows the sparsity pattern of the linear equation set which results from a finite element discretization.

Consider the two-dimensional Poisson’s problem in a square domain discretized with bilinear finite elements, as shown in Fig. 3-9. The weak form results from integrating the equation over the domain with weighting functions \( \{ \Phi^i \} \), which are the finite element basis functions used in approximating the solution. This weak form is

\[
\int_D \Phi^i (\nabla^2 u - f) dA = 0 \tag{3.13}
\]

Integrating by parts and applying the boundary conditions yields

\[
\int_D (\nabla \Phi^i \cdot \nabla u - \Phi^i f) dA = 0 \tag{3.14}
\]
Approximating the solution with the bilinear basis functions gives

$$u \simeq \sum_j \Phi^j u_j$$  \hspace{1cm} (3.15)

Substituting the approximation for $u$ into the weak form yields a set of linear system of equations as

$$\sum_{j=1}^{N} \left[ \int_D \nabla \Phi^i \cdot \nabla \Phi^j dA \right] u_j = \int_D \Phi^i f dA$$  \hspace{1cm} (3.16)

The index $i$ in Eq. (3.16) refers to the equation and index $j$ to the unknown. Comparing Eq. (3.16) with Eq. (3.12), the elements of the coefficient matrix $a_{ij}$, the unknown vector $x_j$, and the right side vector $b_i$ are identified as

$$a_{ij} \equiv \int_D \nabla \Phi^i \cdot \nabla \Phi^j dA$$  \hspace{1cm} (3.17)

$$x_j \equiv u_j$$  \hspace{1cm} (3.18)

$$b_i \equiv \int_D \Phi^i f dA$$  \hspace{1cm} (3.19)
for \( i \) and \( j = \{1, 2, ..., N\} \).

The sparsity pattern of \( A \), shown in Fig. 3-10, is for the equations and unknowns numbered according to the band ordering. There are nine nonzero coefficients for each equation, except for the degrees-of-freedom on the boundaries of the domain which have only one nonzero coefficient due to the essential boundary conditions imposed there. The nine nonzeros appear in \( A \) as the main diagonal with immediate super- and sub-diagonals and with three super- and three sub-diagonals which lie at a distance of about \( \sqrt{N} \) from the main diagonal. The sparsity of \( A \) follows from the shape of the basis and weighting functions \( \{\Phi^i\} \) which are nonzero only over four elements, as shown in Fig. 3-9. There are nine nodes in each set of four adjacent elements in Fig. 3-9. These nine nodes correspond to the nine nonzero entries in a row of \( A \). The term \( \int_D \nabla \Phi^i \cdot \nabla \Phi^j \, dA \) is nonzero only if \( \Phi^i \) and \( \Phi^j \) are the basis functions for one of the nine nodes in the four adjacent elements. The diagonal entry in \( A \) is for \( i = j \), the immediate super- and sub-diagonals are for \( j \) located directly at the right or directly at the left of \( i \), and the six farther super- and sub-diagonals are for \( j \) located at the top or bottom of \( i \), assuming that the numbering is done from left to right in the band ordering.

Different ordering schemes result in different sparsity patterns for \( A \). For example, Fig. 3-11 shows the sparsity pattern of \( A \) ordered with nested dissection. In nested dissection ordering, the domain is divided recursively into subdomains and separators, which are the borders of the subdomains. First, the subdomains are numbered, and then the separators are numbered. The separator which was created first in the dividing of the domain into two is numbered last. Note that between the band (Fig. 3-10) and the nested dissection (Fig. 3-11) orderings, only the pattern of nonzeros is different, not the number of them. However, the fill-in is different for the two cases if Gaussian elimination (\( LU \)-decomposition) is performed on \( A \). Figures 3-12 and 3-13 show the sparsity of \( A \) after \( LU \)-decomposition.\(^2\) The required amounts of storage

\(^2\)See Fig. 3-16 for a description of the \( LU \)-decomposition algorithm.
Figure 3-10: The sparsity of the coefficient matrix for Fig. 3-9 ordered with a band method.
3.2 Solution of PDE'S

Figure 3-11: The sparsity of the coefficient matrix for Fig. 3-9 ordered with nested dissection.
Figure 3-12: The sparsity of the coefficient matrix for Fig. 3-9 with band ordering after Gaussian elimination. Darker regions are the fill-in introduced by elimination and pivoting within the band.
Figure 3-13: The sparsity of the coefficient matrix for Fig. 3-9 ordered with nested dissection after Gaussian elimination. Darker regions are the fill-in introduced by elimination and pivoting within subdomains and separators.
are different for the band and nested dissection orderings after \( LU \)-decomposition of \( A \), and the computational complexities are different for the two cases. These differences are quantified in Chapter 4.

**Algorithms for Solution of Linear Systems of Equations**

Algorithms for the solution of linear systems of equations fall into two broad classes known as *iterative* and *direct* methods. Direct methods, such as \( LU \)-decomposition with forward and backward elimination, solve the linear system exactly, whereas iterative methods, such as the Jacobi's method, rely on iteration to approximate the solution. Direct methods give the exact solution in a finite number of steps if the arithmetic is exact. Iterative methods correct the approximate solution successively to improve it; however, improvement is not always guaranteed. Each class has both advantages and disadvantages, and the choice is not obvious in many instances.

Generally, direct methods are preferred because of their robustness, but they become too costly for very large problems. As a result, many researchers have investigated iterative schemes to reduce the computational time and memory usage. However, the convergence of iterative solvers is not guaranteed for general linear systems. Both direct and iterative methods for solution of linear systems of equations are evaluated below.

Iterative (indirect) solvers have been much more popular than direct solvers that use domain decomposition methods because of their simplicity and low storage requirements; for example, see Fischer et al. [97] and Christara [51]. These methods start from an approximation and attempt to successively improve it until a sufficiently accurate solution is obtained. Some examples for iterative methods are Jacobi's method, the Gauss-Seidel method, and the successive overrelaxation (SOR) method. These examples are described in many standard textbooks [69]. The most popular class of iterative methods has emerged from the conjugate gradient method devised by Hestenes and Stiefel (CGHS) [135]. The CGHS method is written as
3.2 Solution of PDE'S

\[ r^0 = b - Ax^0 \text{ and } p^0 = r^0 \]

for \( k = 1, 2, 3, \ldots \)

\[ \alpha^k = \frac{||r^k||^2}{p^k^TAp^k} \]

\[ x^{k+1} = x^k + \alpha^k p^k \]

\[ r^{k+1} = r^k - \alpha^k Ap^k \]

\[ \beta^k = \frac{||r^{k+1}||^2}{||r^k||^2} \]

\[ p^{k+1} = r^{k+1} + \beta^k p^k \]

where \( x^0 \) is the initial guess. The vectors \( p^k \) are called the search directions and \( r^k \) the residuals. The CGHS method minimizes the function \( \frac{1}{2}x^TAx - b^Tx \). It can be shown that \( p^k^T r^k = ||r^k||^2 \) and that \( p^k^T Ap^l = 0 \) for \( k \neq l \) [224]. The latter is the an \( N \)-dimensional generalization of the condition for conjugacy in the theory of conic sections and quadric surfaces [69]. When \( A \) is symmetric and positive-definite, CGHS is a direct method; that is, \( x^N \) will be the exact solution when the arithmetic is exact. \( x^{k+1} \), which is often a good approximation, is obtained by stopping the above procedure at some point at which \( k < N - 1 \). The CGHS method is popular because it is optimal in the sense that it minimizes the \( A \)-norm (defined below) of the error over some subspace. Moreover, CGHS does not require a priori parameter estimates and may be implemented via a three-term recursion [7].

The general class of preconditioned conjugate gradient methods are defined and categorized by Ashby et al. [7]. Briefly described, the broader class of gradient methods consists of schemes which produce a sequence of approximations to \( x = A^{-1}b \) by

\[ x^{k+1} = x^k + d^k \quad (3.20) \]

starting from an initial guess \( x^0 \), where \( d^k \) belongs to the Krylov space of dimension at most \( k + 1 \) given as

\[ V^{k+1}(r^0, A) \equiv \text{span}\{r^0, Ar^0, A^2r^0, \ldots, A^k r^0\} \quad (3.21) \]
The choice of $d^k$ from the Krylov space results in a particular gradient method. A conjugate gradient method results if the $B$-norm of the error is minimized. The $B$-norm is $<B, \cdot>$, where $<\cdot, \cdot>$ is the usual Euclidean inner product. The matrix $B$ is called the inner product matrix. The error at the step $k$ is $e^k = x - x^k$, and the error at the step $k + 1$ follows from Eq. (3.20) as

$$e^{k+1} = e^k - d^k$$

(3.22)

The vector $d^k \in V^{k+1}$ is chosen to minimize $||e^{k+1}||_B \equiv <Be^{k+1}, e^{k+1}>^{1/2}$.

In order to speed up the convergence rate, the iterative methods are usually applied to the preconditioned problem

$$QAP\tilde{x} = Qb$$

(3.23)

where $Q$ and $P$ are nonsingular linear transformations and $P\tilde{x} = x$. Ideally, the matrix $\tilde{A} = QAP$ has a smaller condition number than $A$, and the iterative scheme converges faster for the preconditioned problem than for the original problem. The preconditioned conjugate gradient algorithms are described by Ashby et al. [7].

Despite their popularity, iterative methods are not as robust as direct methods, and convergence problems may arise for asymmetric, indefinite matrices, for example, due to convective terms in transport equations. Experience shows that indirect solvers need to be preconditioned to be practical [243], but the choice of good preconditioners is usually problematic, especially for asymmetric, indefinite systems, for example in problems involving fluid mechanics [39]. One way to make partial use of the robustness of direct solvers with only some sacrifice of storage is to use incomplete $LU$-factorization for preconditioning [198, 11].

Nachtilgal et al. [204] have reviewed three of the best known iterative methods for asymmetric matrices: CGNR, the conjugate gradient applied to normal equations [135], GMRES, residual minimization in a Krylov space [234], and CGS, a biorthogonalization algorithm adapted from the biconjugate gradient iteration [245].
$\beta^o := 0; \ p^o := 0$

for $k = 1, 2, \ldots$

\begin{align*}
p^k &:= A^* r^{k-1} + \beta^{k-1} p^{k-1} \\
\alpha^n &:= ||A^* r^{k-1}||^2/||Ap^k||^2 \\
x^k &:= x^{k-1} + \alpha^k p^k \\
r^k &:= r^{k-1} - \alpha^k Ap^k \\
\beta^k &:= ||A^* r^k||^2/||A^* r^{k-1}||^2
\end{align*}

$\| \cdot \|$ is the two-norm, $r^k$ is the $k$th residual, and $A^*$ the Hermitian transpose of $A$.

Figure 3-14: CGNR algorithm [204].

Tests with various matrices show that convergence properties of each method are different. Moreover, convergence depends on the type of the matrix considered. Nachtigal et al. tested eight different kinds of matrices and noted convergence in 1 to $N$ steps where $N$ was the size of the matrix. CGS diverged in two cases.

These results may seem somewhat discouraging for iterative methods, but even the worst results of these methods may be acceptable in some cases. Consider a spectral element discretization, so that the resulting $N \times N$ system is full. Suppose one of the above iterative methods is used which converges at $N$ steps, i.e. one of the worse cases. As an example, consider the CGNR algorithm (Fig. 3-14). At each step, two matrix vector multiplications, two vector dot products, two scalar vector multiplications, and three vector additions are performed. On a parallel computer with at least $N$ processors, two vectors are multiplied in one computational time step for a floating point operation; the other operations can be performed in $O(N)$ time steps, ignoring communication costs, and assuming that the matrix is partitioned column-wise among the processors, and that the vector is divided among the processors element-by-element, as shown in Fig. 3-15. Convergence is reached after $O(N^2)$ time steps. This is remarkable compared to serial Gaussian elimination which
takes $O(N^3)$ computational time steps. Some practical limitations were overlooked in reaching this conclusion, for example the size of memory of each processor and the cost of communication.

Now consider the most versatile and popular direct method: Gaussian elimination ($LU$-decomposition) with pivoting. Figure 3-16 shows $LU$-decomposition with pivoting. Forward elimination (Fig. 3-17) and back substitution (Fig. 3-18) complete the solution and store it in the same storage location as the right side vector $b$. Forward elimination and back substitution have computational complexities which are lower by $O(N)$ than the ones for $LU$-decomposition for the serial algorithm. Since they do not contribute to the asymptotic discussions below, only $LU$-decomposition will be considered.

Factorization of sparse positive definite systems has been extensively analyzed using graph-theoretic approaches and the concept of elimination trees [111, 105, 9]. These systems do not require pivoting, so the work in this area relies on preservation of the structure of the matrix. The discussion here will include pivoting, because it is necessary for numerical stability of factorization of general asymmetric matrices. Many of the concepts and ideas for factorization of symmetric, positive-definite systems generalize to algorithms for $LU$-factorization with pivoting.

Parallelizing $LU$-factorization is not as straightforward as parallelizing conjugate gradient iteration. It helps to identify the independent tasks and organize them in a systematic diagram such as a task graph for better division of the tasks among individ-
3.2 Solution of PDE'S

for $k = 1$ to $N - 1$
    find $p$ such that $|a_{pk}| = \max\{a_{kk}, \ldots, a_{nk}\}$
    $piv(k) := p$
    interchange $a_{piv(k),k}$ and $a_{kk}$
    $t := 1/a_{kk}$
    for $i = k + 1$ to $N$
        $a_{ik} := a_{ik} \ast t$
    for $j = k + 1$ to $N$
        interchange $a_{piv(k),j}$ and $a_{kj}$
    for $i = k + 1$ to $N$
        $a_{ij} := a_{ij} - a_{ik} \ast a_{kj}$

Figure 3-16: Algorithm for $LU$-decomposition with pivoting.

for $k = 1$ to $N - 1$
    interchange $b_{piv(k)}$ and $b_k$
    for $i = k + 1$ to $N$
        $b_i := b_i - a_{ik} \ast b_k$

Figure 3-17: Forward elimination algorithm.

for $k = N$ down to $1$
    $b_k := b_k/a_{kk}$
    for $i = 1$ to $k - 1$
        $b_i := b_i - b_k \ast a_{ik}$

Figure 3-18: Back substitution algorithm.
for $k = 1$ to $N - 1$
    execute $T_{kk}$
    for $j = k + 1$ to $N$
        execute $T_{kj}$

where $T_{kk}$ and $T_{kj}$ are the following tasks:

$T_{kk}$:
• Find the pivot (active entry with largest absolute value).
• Interchange the pivot with the entry in the current row.
• Scale the entries in the column below the pivot with the pivot.

$T_{kj}$:
• Modify the number in the columns to the right and the rows below the pivot appropriately.

Figure 3-19: Gaussian elimination algorithm organized in tasks.

al processors. Marrakchi and Robert [185] used a graph-theoretic approach to derive asymptotically optimal algorithms for parallel $LU$-decomposition with partial pivoting on a MIMD computer. Their results demonstrate the high degrees of parallelism inherent in this algorithm; see also Lord et al. [179]. The algorithm for Gaussian elimination with partial pivoting [179] is written as in Fig. 3-19. The precedence constraints in the algorithm are

1. $T_{kk}$ should be done before $T_{kj}$

2. $T_{kj}$ should be done before $T_{k+1j}$

where $1 \leq k \leq N - 1$ and $k + 1 \leq j \leq N$. Constraint (1) means that the pivot column should be prepared before the other columns are modified, and constraint (2) means that the update of column $j$ at step $k$ must be completed before its update at step $k + 1$ can be started. The task graph for $N = 5$ is shown in Fig. 3-20 [179]. Assuming that a computational time step consists of one multiply and one compare, or one multiply and one subtract, it takes $N + 1 - k$ time steps for each task $T_{kk}$.
Figure 3-20: Gaussian elimination task graph for $N = 5$. 
for $p = 1$ to $N - 1$
  find pivot
  broadcast pivot row down
  form pivot column
  broadcast pivot column to the right
  for $q = p + 1$ to $N$
    update column $q$ using pivot row and column

Figure 3-21: An algorithm for Gaussian elimination in parallel on a SIMD machine.

and $N - k$ time steps for task $T_{kj}[179]$. The number of sequential execution time steps is $T_s = \frac{N^3}{3} + O(N^2)$. The parallel time for the above algorithm is the longest path in the task graph. This path contains $T_{11}$, $T_{12}$, $T_{22}$, ..., $T_{kk}$, $T_{kk+1}$, ..., $T_{n-1n}$. So the number of parallel time steps is $T = N^2 - 1$. This number is based on the assumption that enough processors are available to perform the desired tasks in parallel. Therefore, Gaussian elimination can have a computational complexity of $O(N^2)$ as did the CGNR algorithm which converged in $N$ steps. More detailed discussion of the optimal number of processing elements and other issues for parallel Gaussian elimination are discussed by Marrakchi and Robert [185].

The above method is not the only way to parallelize Gaussian elimination. Compare, for example, a similar algorithm for a SIMD parallel computer, as shown in Fig. 3-21. Each computational operation in the loops of Fig. 3-21 is performed in parallel. Suppose the elements of the matrix are assigned to processors of a SIMD computer with a two-dimensional mesh architecture so that there is a one to one correspondence between each element of the matrix and each processor. Also, assume that communication and computation take the same amount of time. Computational complexities are: $O(p)$ for finding the pivot column, $O(p)$ for broadcasting the pivot row, $O(1)$ for forming the pivot column since this is done in parallel, $O(p)$ for broadcasting the pivot column, and $O(1)$ for updating the $p + 1$ to $N$ columns since
these computations are done simultaneously. Therefore, as in the previous cases for a MIMD machine, the total time is $O\left(\sum_{p=1}^{N-1} p\right) = O(N^2)$.

The above analyses are asymptotic cases with simplifying assumptions. In a real situation, the limited number of processors, slow speed of communication, variable coefficients of the asymptotic terms which depend on the speed of computation, and complicated sparsity patterns would prevent ideal improvements in the solution time. More clever, but more complicated, algorithms would reduce the communication and computational complexities and help reduce the solution time.

While the choice between an iterative and a direct solver is not obvious, several distinctions are clear. Generally, if the problem is too large to fit in memory with direct solvers, there is no choice; an iterative solver is the only path, no matter how fast it converges. If the problem can fit in memory, usually direct solvers are advantageous. This is because direct solvers will solve the linear system of equations for a much larger class of problems. Alternatively, if a good preconditioner is available for a problem, the iterative solver may be faster and thus more economical to use.

Another practical consideration is the fact that iterative algorithms are much easier to program. For direct solvers, efficient use of sparsity without sacrificing speed is not easy to achieve. However, both the robustness and the general applicability of direct solvers make the larger programming efforts worthwhile. A direct solver was developed in this thesis. Details of the particular direct solver are discussed in Chapter 4.

Formulation and Solution with Direct Methods

Frontal and band formulations have been very popular for single processor environments because of their simplicity, robustness, and ease of vectorization; however, the non-hierarchical nature of these direct solvers makes them less attractive for parallel implementation. Multifrontal, domain decomposition, and nested dissection methods with direct solvers are hierarchical, and therefore, much better suited for paralleliza-
tion [80, 15]. Some of these hierarchical algorithms have lower computational complexities and could therefore be superior in the serial case. For example, for a 2-D $n \times n$ grid, nested dissection methods have a computational complexity of $O(n^3)$ and a memory requirement of $O(n^2 \log_2 n)$, whereas band formulation has a complexity of $O(n^4)$ and a memory requirement of $O(n^3)$ [114, 115]. The reason for the unpopularity of the nested dissection methods is its complicated data structures, which make the development of efficient codes difficult [115].

In serial band formulation, the grid is swept sequentially until all elements are formulated and stored appropriately. Then, $LU$-decomposition, forward elimination, and back substitution are performed to arrive at the final solution. The procedure is inherently sequential, but it can be parallelized through parallelizing the loops, and can thereby achieve good speedups at least theoretically [100].

In frontal algorithms, matrix formulation and $LU$-decomposition are done at the same time. A window holds the active equations and moves through the matrix as shown schematically in Fig. 3-22. Once an equation is fully assembled, it is ready to be factored. Frontal methods have two advantages over band methods for finite and spectral element discretizations. The first advantage is independent of the type of discretization; frontal methods can be easily adapted to use out-of-core memory.
when the amount of in-core memory is small. The decomposed equations are stored out-of-core and are retrieved only when needed for forward elimination and back substitution. The other advantage is that frontal methods reduce the bandwidth relative to traditional band methods because the equations which are fully summed are eliminated and do not contribute to the bandwidth. These fully summed equations correspond to the interior nodes of the elements, so the bandwidth is reduced by a factor of approximately the order of the basis functions in one dimension for a two-dimensional rectangular domain. The variable bandwidth is easily handled for irregular domains.

Serial frontal methods are not inherently parallel. Only the independent loops can operate in parallel, as was the case for the band methods. However, both serial band and frontal methods are amenable to vectorization. The inner loops are vectorized efficiently using basic linear algebra subroutines (BLAS) [78], making these methods excellent for implementation on vector supercomputers. As a result, the serial band and frontal methods have been used very successfully on vector supercomputers.

In contrast to serial band and frontal methods, domain decomposition methods lend themselves naturally to parallel formulation and solution. The essence of domain decomposition is the dividing of the physical domain into subdomains and assigning each subdomain to a processor (or processors). The assigned processor formulates and solves the equations of its subdomain and then cooperates with the other processors to sum and solve the equations for the borders of the subdomains.

Consider a domain decomposed into two subdomains and assigned to two individual processors as in Fig. 3-23. If a multifrontal method [83] is used, each processor goes through its assigned subdomain to formulate and eliminate the equations for the interiors of the subdomains. Equations are fully summed and eliminated until the equation corresponding to the common border of the subdomains is reached. Elimination of these equations is left until the end. Until then, every step is performed in parallel. Adding up the border equations is not fully parallel, nor is eliminating
Figure 3-23: Domain decomposition of a rectangular domain into two subdomains.

them. Since leaving these tasks to one processor reduces the efficiency of parallelism, the processors cooperate to add and eliminate the equations for the common borders. There is complete parallelism when the processors formulate and eliminate the interior subdomains; but there is less parallelism, which is harder to program but necessary to exploit, when the processors have to patch the subdomains together along the common borders. The multifrontal method shown in Fig. 3-23 is naturally more parallel than its serial frontal counterpart. Parallelizing the serial frontal method is similar to treating the whole problem as the common borders are treated in the multifrontal algorithm; therefore, the naturally parallel portion does not exist for the serial frontal algorithm.

3.3 Concepts and Goals

General goals of serial computing, like efficient use of sparsity and minimizing computations, also apply to parallel processing. Other goals appear as well, most importantly minimizing the communication and balancing the load. As in the serial case, computational speed is measured in millions of floating point operations per second (MFLOPS) and the elementary goal is to maximize the speed of a single processor. Also, it is desirable to scale the performance so that increasing the number of processors increases the computation speed linearly. The linear increase implies that
increasing the number of processors should reduce the computational time inversely. Both minimizing communication and balancing the load help approach this goal; the former by preventing time lost in communication, and the latter by preventing idle time for the processors. Scaled performance is measured through efficiency or speedup. Speedup is defined as the time of computation on one processor divided by the time of computation on several processors; the ideal speedup for \( P \) processors is \( P \). Efficiency is the speedup divided by \( P \); it is measured the fraction of the ideal speedup achieved. Direct calculation of speedup may not be possible if the problem is too large to fit on one processor. In that case, the maximum speed of a single processor may be used for speedup calculation. The speedup can alternatively be calculated based on equal problems per processor by dividing the speed on several processors by the speed on one processor, in which case the size of the problem per processor is the same in both instances. Since there are many possible ways of calculating the speedup, it is important to specify clearly what method has been used to calculate it.

Several measures are considered for evaluating the performance of a parallel algorithm. Speedup by itself is not enough. For a good parallel algorithm several conditions must be satisfied:

- Solution of larger problems should be feasible with the increasing number of processors.

- Communication should be kept to a minimum.

- A reasonable fraction of the peak speed should be achieved on a single processor.

- The algorithm speed should scale up linearly. That is, the speedup should not deteriorate with an increase in the number of processors.

- The absolute solution time should be low.

- The algorithm should be independent of the machine to the maximum extent possible.
All of the above measures need to be considered and compared with other existing algorithms before deciding on the usefulness of a parallel algorithm. In addition, the theoretical peak speed of the machine(s) used for testing the algorithm, and the ratio of communication to computation speed should be taken into account to guard against claiming high speedups without, for example, any mention of absolute speeds. The algorithm which achieves high speedups but is slow is not useful for meaningful calculations. The criteria detailed above are all considered in the discussion of the \textit{LU}-decomposition method that is described in the next Chapter.
Chapter 4

A Parallel Method for Direct Solution of Partial Differential Equations Discretized with Finite or Spectral Elements

The development of efficient and robust numerical algorithms for use with parallel computer architectures is an important problem for solution of the wide variety of problems in fluid flow and heat and mass transfer that are of interest in modeling materials processing and manufacturing. The mathematical models for this class of problems typically are composed of a group of differential equations, algebraic equations and integral constraints. For steady-state (time-independent) models, finite element or finite difference discretization of these models leads to large sets of nonlinear differential equations with highly structured coupling between the variables. These dependencies are naturally asymmetric: the linearization of the nonlinear equation sets leads to asymmetric Jacobian matrices. For serial vector computers, Newton's method, coupled with direct $LU$-decomposition, has proven to be an efficient method for solution of nonlinear equation sets, at least for the discretization of problems in two
space dimensions, where the structure of the nonzero entries in the Jacobian matrix makes direct $LU$-decomposition reasonably efficient in terms of operation count and memory usage. Indeed, finite-element/Newton algorithms have been used to solve an enormous variety of transport problems and are the basis of several commercial computer codes used for this purpose. The popularity of this algorithm comes from the robustness of Newton's method for converging to the solution of extremely nonlinear problems and also from the availability of continuation methods, based on the computation of the factors of the Jacobian matrix [274, 162] for mapping out multiple solutions.

The goal of the development of the algorithm described in this Chapter is to extend finite-element/Newton algorithms for the solution of transport problems to MIMD parallel computers. The largest component of this endeavor is the development of an $LU$-factorization algorithm for the solution of large, sparse, asymmetric linear equation sets based on domain decomposition and nested dissection for partitioning the equations to the processors of the MIMD computer. The algorithm used here is based on dividing the geometrical domain recursively into subdomains and separators using nested dissection [107, 111], and assigning subdomains and associated separator data to each processor. This algorithm extends the work of Lucas et al. [180] for symmetric, positive-definite matrices in order to avoid duplicate storage, to allow asymmetric matrices and to allow partial pivoting during factorization. The algorithm stores the upper and lower triangular forms, and performs forward and backward solutions separately.

This Chapter describes the method proposed here for solution of partial differential equations in parallel. An introduction is given in Section 4.1 for the choices involved in direct solution of linear systems which result from finite or spectral element discretizations in conjunction with fixed-point iterative procedures, such as Newton's method, used for solving the nonlinear algebraic equations. A background is presented for different issues involved in sparse matrix factorization in parallel
in Section 4.2. The devised parallel algorithm, called Concurrent Factorization and Storage (CFS), is presented in detail in Section 4.3. The algorithm is summarized in Section 4.4.

4.1 Introduction

Finite element, spectral element and finite difference discretizations of elliptic partial differential equations lead to large sets of nonlinear algebraic equations. Nonlinear equations are solved by some variant of a fixed-point iteration. Most Newton-like iterative algorithms used for this purpose require the solution of large sets of linear algebraic equations during each iteration. In these formulations, the solution of these linear algebraic equation sets is the most computation- and memory-intensive part of the solution procedure.

The choice of algorithm for the solution of either the linear or nonlinear problem entails a compromise between the efficiency of the algorithm for a well-known problem and the robustness of the approach for a larger class of problems. Direct solution methods for solving the linear equation sets have been preferred over iterative matrix methods because of their robustness. Algorithms for LU-decomposition for band matrices have been very popular for single processor computational environments because of the simplicity of the implementation and the potential for vectorization. However, the non-hierarchical nature of these algorithms makes them much less attractive for implementation on parallel computers.

Multifrontal, domain decomposition, and nested dissection algorithms implemented for LU-decomposition are hierarchical and thus are much better suited for parallelization [80, 15]. Some of these algorithms also offer superior performance for serial computation: for example, for a two-dimensional $n \times n$ grid discretized with finite elements, nested dissection methods have complexities of $O(n^3)$ and memory requirements of $O(n^2 \log_2 n)$, whereas LU-decompositions for band matrices have complex-
ities of $O(n^4)$ and memory requirements of $O(n^3)$ [115, 114]. The only drawback of nested dissection methods found in the course of this research is that the complicated data structures which arise in implementation inhibit efficient vectorization and make programming difficult.

Nested dissection for ordering symmetric, positive-definite matrices is described by George and Liu [111]. Lucas et al. [180] used the hierarchical nature of nested dissection in developing a parallel, multi-frontal algorithm based on the SPARSPAK implementation for symmetric, positive-definite systems [53]. Lucas et al. divided the geometrical domain recursively into subdomains and separators based on the nested dissection procedure [107, 111]. Each subdomain was assigned to a processor, and each processor stored data pertinent to its subdomain in addition to setting aside additional storage for relevant separators to reduce communication overhead at the cost of memory usage. The algorithm presented in this Chapter extends the approach of Lucas et al. to avoid duplicate storage and to include application to asymmetric equations by allowing partial pivoting and storage of both the upper and lower triangular matrices.

The extension of existing nested dissection ideas to include partial pivoting is conceptually simple. Incomplete nested dissection allows grouping of nodes into supernodes. Single matrix entries are replaced by blocks representing the supernodes, and pivoting is allowed within each supernode. The concept of blocks has the added advantages of allowing pipelining, vectorization and parallelism within a single block, with only a slight increase in the overall work and storage for the matrix. Duff and Reid [83] suggested the concept of blocks, and George and Rashwan [115] used them in their incomplete nested dissection algorithm. Neither method, however, allowed for pivoting. Duff and Reid [83] presented a multifrontal method for indefinite systems and extended it to asymmetric systems [84]; however, these algorithms did not incorporate the concept of blocks.

For the block approach to be substantially advantageous, the blocks must be large
4.1 Introduction

enough to justify tasks such as pivoting, pipelining and vectorization, as indicated by the work of Dayde and Duff [73] on the implementation of level-3 BLAS routines for dense matrices. Also, the asymmetry and structure of the matrix dictates a minimum number of pivot choices required for accurate \( LU \)-decomposition. This problem has been addressed traditionally by allowing pivoting only within the band or within the active front in serial software [139, 140].

The implementation described here is based on an incomplete nested dissection ordering both for the domain decomposition among processors, and for ordering within the subdomain of each processor. The nested dissection is incomplete so that pivoting is possible within supernodes which consist of groups of equations. Duff and Johnson [81] investigated the effects of various orderings on regular and irregular grids and found that nested dissection performed better than other orderings on regular grids and was only slightly less efficient than algorithms based on tree height minimizations for irregular grids. Duff [80] pointed out that nested dissection gives a balanced elimination tree with good load balancing in parallel implementation and performs almost as well as the other methods in an environment for sequential computations.

The purpose of this Chapter is to present an algorithm for direct \( LU \)-decomposition of large, sparse systems of linear equations that arise in applications of finite element and spectral element methods to linear and nonlinear differential equations. The discussion will focus on the discretization of partial differential equations, but is not specific to any particular type of discretization. The only assumption is that the discretized problem can be divided into subdomains which include elements with basis functions that have local support. For the linearized problem, the resulting set of linear algebraic equations is given by Eq. (3.12) of Chapter 3

\[
Ax = b
\]

where \( A \in \mathbb{R}^{N\times N} \) is large and sparse, and \( x, b \in \mathbb{R}^N \). The structure of \( A \) is dictated by the original problem, by the discretization technique, and by the equation ordering. These choices lead to a natural decomposition of \( A \), \( x \), and \( b \) in the linear problem
where $A_i$, $x_i$, and $b_i$ have nonzero structure only corresponding to the $i$th subdomain or separator, and the portions of the separators bordering that subdomain or separator. A background on the terminology and properties of sparse matrix factorization is given below, followed by a detailed description of solution to Eq. (3.12) using the sparsity structures in Eq. (4.1).

4.2 Background and Choices

4.2.1 Introduction

The overall solution procedure for elliptic PDE's was discussed in Chapter 3 and depicted in Fig. 3-5. Briefly stated, the partial differential equations and boundary conditions are discretized with finite or spectral element methods, as described in the example problems of Chapter 5 for steady-state problems. In general, the discretization results in a set of nonlinear algebraic equations which is solved by Newton's method. Newton's method requires solution of a linear system at each iteration. The basic algorithms for solution of the linear system are $LU$-factorization, forward elimination, and back substitution; these are quite simple, as discussed in Chapter 3 and illustrated in Figs. 3-16, 3-17, and 3-18. The challenges arise from the complicated sparsity patterns, i.e. the zero and nonzero entries in the coefficient matrix $A$, also called the stiffness or Jacobian matrix, and the requirement for efficient use of these patterns for parallel solution. The properties of this linear system are tightly coupled with the equations and boundary conditions, as well as with the discretization techniques.

Here, the issues in solving the linear systems are discussed in terms of the elliptic PDE's and finite element or spectral element discretizations. The nature of the
sparsity patterns in the linear equation set is tightly coupled with the problem and its discretization.

Relevant concepts for solution of general, sparse systems are introduced and are discussed in more detail in the following Section. Simple examples are used for introduction and discussion of the concepts to facilitate the description of the underlying principles. These concepts readily generalize directly to more complicated problems.

4.2.2 Overview

The object here is to solve the linear system of Eq. (3.12)

\[ Ax = b \] (3.12)

where \( A \in \mathbb{R}^{N \times N} \) is large and sparse. To do this, \( A \) is decomposed into lower and upper triangular matrices according to

\[ A = LU \] (4.2)

which yields the solution of the system (3.12) in two steps: first solving the lower triangular system

\[ Ly = b \] (4.3)

by forward elimination, then solving the upper triangular system

\[ Ux = y \] (4.4)

by backward elimination or back substitution.

The sparsity pattern of Eq. (3.12) is dictated by the finite or spectral element discretizations, by the interpolants used, and by the equation ordering. The interpolants used for each finite element have local support which is limited to that particular element; therefore, the equation for a degree-of-freedom of an element depends only on the degrees-of-freedom within that element. The equations couple across the elements
only through the common borders of the elements. The connectivity of the equations does not change with the numbering of the degrees-of-freedom; it is a property which results from problem geometry and discretization. The discretization is applied to the governing equations, and in this sense, the governing equations also determine the connectivity, which is best represented by concepts from graph theory discussed below.

The numbering of the degrees-of-freedom, generally referred to as ordering, does not change the connectivity of the problem, but it determines the order in which the equations are solved by $LU$-factorization, forward elimination, and back substitution. $LU$-factorization introduces fill-in or simply fill; i.e. it changes some zeros in the coefficient matrix to nonzeros. Therefore, the sparsity structure of the coefficient matrix is changed during factorization. Different ordering strategies result in different amounts of fill. Therefore, the sparsity patterns are different for $L$ and $U$ factors of the same matrix but with different orderings.

In $LU$-factorization of sparse linear systems, the zero entries which remain zero through the process are ignored to save computation and storage. Therefore, different orderings give different operation counts, a feature referred to as computational complexity, because of the different evolution of sparsity patterns during factorization. Different ordering methods also require different amounts of computer storage.

In order to be efficient, parallel solution requires equal distribution of tasks and data among the processors and proper management of the data. The connectivity and the ordering limit the possible degree of parallelism. Therefore, the choice of task and data distribution is connected to the choices of discretization and ordering. Once a discretization and an ordering are chosen, efficient distribution of task and data are required to exploit the parallel nature to the greatest possible extent.

The following steps comprise the solution of the sparse linear system of equations [134], given by Eq. (3.12).

1. Ordering. Find a good ordering such that
4.2 Background and Choices

(a) The matrix $A$ suffers the least amount of fill during factorization;

(b) the factorization is amenable to vectorization and parallelism.

2. Symbolic factorization. Determine the data structure for $A$ and the filled matrix $F = L + U$ and set up proper storage and integer addressing arrays for $A$ to locate the nonzero entries of $A$.

3. Formulation. Formulate the entries of $A$ and $b$.


5. Triangular solution. Solve the upper and lower triangular systems as in Eqs. (4.3) and (4.4).

All of the above steps are parallelizable. In practice, the most work-intensive part of the algorithm is numeric factorization. Ordering is also a very difficult combinatorial problem; in fact, this problem is NP-complete [275]. Nevertheless, fast heuristic methods are applied; examples include the heuristic nested dissection algorithm for rectangular domains described in Section 4.3.3, the more general heuristic nested dissection algorithms described by George [107] and George and Liu [110], and the minimum degree algorithm described by Rose [227], Liu [173], and George and Liu [112].

4.2.3 Graphs and Elimination Trees

Both graphs and elimination trees are discussed below as applied to the direct solution of sparse linear equation sets. The notions of trees and graphs have been used in Cholesky factorization of symmetric, positive-definite systems [111, 80, 4]. These subjects have been studied intensely and applied to practical problems. Nevertheless, there has not been much work in the area of general, indefinite sparse systems. The reason is that pivoting, which is required to maintain numerical stability [99, 82],
destroys the sparsity patterns of the indefinite equation sets and makes the application of graphs and trees more complicated. This problem has been circumvented here by grouping equations and allowing pivoting only within each group. Therefore, the sparsity pattern is only destroyed within each group of equations, leaving the overall pattern unchanged. The discussion in this section does not include pivoting. The necessary foundation and the basic principles of graph theory and elimination trees, as used in sparse Cholesky factorization, are described. Generalization to include pivoting is discussed subsequently, once the basic definitions and properties are established.

Graphs

The use of notation from graph theory is helpful for the discussion of sparse matrix factorization. For a comprehensive reference to graph theory, consult Rose [227]. The following discussion is based on the work of George and Liu [111].

The graph $G = (X, E)$ is a finite set of nodes or vertices $X$ with a set of edges $E$, which are ordered pairs of vertices. The vertices of a graph may be ordered, i.e. numbered or labeled, producing an ordered graph, which is denoted by $G^\alpha = (X^\alpha, E)$, where $\alpha$ denotes a specific ordering of the vertices. When the edges have a direction, the graph is called a directed graph.

An unordered graph may be used to represent the connectivity of the degrees-of-freedom for a finite or spectral element discretization. The graph $G = (X, E)$ for a discretized problem is one for which $X$ is the set of degrees-of-freedom, and $\{x_i^g, x_j^g\} \in E$ if and only if $x_i^g$ and $x_j^g$ correspond to two distinct degrees-of-freedom in the same element, because the degrees-of-freedom in an element are connected only in that element.

As an example, consider a square domain which is discretized by four bilinear elements, as shown in Fig. 4-1. For simplicity, consider only one degree-of-freedom per finite element node. The graph for this problem is constructed quite simply
4.2 Background and Choices

Figure 4-1: A square domain discretized into four bilinear elements.

Figure 4-2: The graph for a square domain discretized into four bilinear elements.
by connecting all of the finite element nodes in each element, as shown in Fig. 4-2. The nodes or vertices of the graph are the same as the degrees-of-freedom for the discretized problem, which coincide with the finite element nodes and, in this case, have a one-to-one correspondence with them. The edges of the graph are the lines which connect the nodes together and represent the connectivity of the nodes.

The graph in Fig. 4-2 depicts the structure of the linear equation set for the problem as well. The unordered graph represents the sparsity, i.e. zero and nonzero structure, of the coefficient matrix by determining which unknowns appear in an equation. For example, the center node is connected to all other nodes in the graph of Fig. 4-2; therefore, the equation for the middle node includes all other unknowns. By contrast, the node in the upper left corner is connected to three other nodes to its right, bottom, and bottom-right; therefore, only the four unknowns in the upper left element appear in the corresponding equation for the node in the upper left corner. Note that the structure of the coefficient matrix \( A \) is symmetric. This is generally true for finite or spectral element formulations. The matrix \( A \) is not symmetric (except for its structure), nor is it positive-definite in general.

Because the graph \( G \) represents the coupling of the unknowns, or alternatively the sparsity pattern of \( A \), without implying any ordering, any permutation of \( A, \text{PAP}^T \), where \( P \in \mathbb{R}^{N \times N} \), also is associated with the unlabeled graph \( G \). The representation for the matrix \( A \) requires choosing some ordering scheme. Choosing an ordering scheme for \( A \) is equivalent to numbering the finite element degrees-of-freedom and to labeling the vertices of the graph \( G \). The labeled graph for \( A \in \mathbb{R}^{N \times N} \), which is denoted by \( G^A = (X^A, E) \), is the graph whose vertices are ordered in the same way as \( A \) from 1 to \( N \) such that \( \{x_i, x_{j \neq i}\} \in E^A \) if and only if \( a_{ij} \neq 0 \) (or \( a_{ji} \neq 0 \)), where \( x_i \) is the node in \( X^A \) with label \( i \). A labeled graph for the problem in Fig. 4-1 and its associated matrix are shown in Figure 4-3. Choosing a different ordering for \( A \) is equivalent to transforming \( A \) with some permutation matrix \( P \), which relabels the graph \( G^A \) to \( G^{\text{PAP}^T} \). Two other labeling schemes for the graph \( G \) and the matrix \( A \)
are shown in Figs. 4-4 and 4-5; the new matrices are permutations of $A$ with some permutation matrices $P$ and $Q$.

Two vertices $x$ and $y$ are adjacent in the graph $G$ if the two vertices are connected by an edge, i.e. if $\{x, y\} \in E$. The adjacent set of $Y \subseteq X$ is

$$\text{Adj}(Y) \equiv \{ x \in X - Y \mid \{ x, y \} \in E \text{ for some } y \in Y \}$$  \hspace{1cm} (4.5)

To reiterate, the adjacent set of $Y$ consists of the vertices in $G$ which are not in $Y$, but are adjacent to at least one vertex in $Y$. The adjacent set of a single vertex in a graph consists of all other vertices connected to it. The interpretation of the adjacent set for a matrix is shown by the example in Fig. 4-6.

The degree of the set $Y$ is the number of the members in $\text{Adj}(Y)$: $\text{Deg}(Y) \equiv |\text{Adj}(Y)|$. For one vertex, the $\text{Deg}(y)$ is the number of edges which the vertex $y$ shares with the other vertices.

A subgraph $G' = (X', E')$ of $G$ is a graph for which $X' \subseteq X$ and $E' \subseteq E$. A section
Figure 4-4: A labeled graph based on nested dissection ordering for a square domain discretized into four bilinear elements and the sparsity pattern of the associated matrix.

Figure 4-5: A labeled graph based on minimum degree ordering for a square domain discretized into four bilinear elements and the sparsity pattern of the associated matrix.
Figure 4-6: The adjacent set for $Y = \{x_1, x_2\}$ is $\text{Adj}(Y) = \{x_3, x_7, x_8, x_9\}$ for this graph and matrix.

Graph $G(Y) = (Y, E(Y))$ is the subgraph wherein

$$E(Y) = \{\{x, y\} \in E \mid x \in Y, y \in Y\}$$ (4.6)

A section graph is a graph itself which is obtained by eliminating some vertices and the associated edges from the graph. In terms of the finite element discretization, a section graph is obtained by eliminating some degrees-of-freedom, and in terms of the matrix, a section graph is obtained by deleting the rows and columns associated with any degree-of-freedom.

A section graph is said to be a \textit{clique} when the vertices in it are pairwise adjacent. The section graph associated with each finite element is a clique. A clique corresponds to a full submatrix.

A path from a vertex $x$ to a vertex $y$ of length $\ell \geq 1$ is an ordered set of $\ell$ distinct vertices $\{v_1 = x, v_2, \ldots, v_\ell, v_{\ell+1} = y\}$ such that $v_{i+1} \in \text{Adj}(v_i)$ for $i = \{1, 2, \ldots, \ell\}$. A graph is \textit{connected} if for every distinct pair of vertices, there exists at least one path
which connects the two vertices. A disconnected graph corresponds to a physical domain which has separated subdomains; it consists of two or more connected components. The matrix corresponding to a disconnected graph is block diagonal with each block representing a connected component of the graph.

Consider a directed, ordered graph of a matrix where the direction of the edges \( \{x_i, x_j\} \) is from \( x_j \) to \( x_i \) for \( i > j \). A directed path between any two vertices \( x_i \) and \( x_j \), where \( i > j \), and \( x_i \) and \( x_j \) are not necessarily adjacent anymore, is a path from \( x_j \) to \( x_i \) which follows the direction of the edges connecting the intermediate vertices.

The set \( Y \subset X \) is a separator of the connected graph \( G \) if the section graph \( G(X - Y) \) is disconnected. For example, the set of vertices \( Y = \{x_7, x_8, x_9\} \) in Fig. 4-4 is a separator for that graph since the connected graph \( G(X - Y) \) consists of two components, one with vertices \( \{x_1, x_2, x_5\} \) and the other with vertices \( \{x_3, x_4, x_6\} \). For finite or spectral element discretizations, any set of vertices that consists of element boundaries and traverses the whole domain is a separator for the graph of that problem. This follows from the local support of the finite and spectral element interpolants.

The graph \( G \) for a finite element discretization can be used to determine the position of the fill-ins during \( LU \)-decomposition. At the \( k \)th elimination step, a multiple of the pivot row \( k \) is added to the remaining rows in the pivot submatrix \( A^k \in \mathbb{R}^{N-k \times N-k} \), as shown in Fig. 3-16, where \( a^k_{ij} \equiv a_{i+kj+k} \). The nonzero entries in the pivot row, other than the diagonal entry, introduce fill-ins in \( A^k \) in (row, column) pairs which include all the possible combinations of the column numbers where the nonzero entries appear. In graph theoretical terminology, the section graph for the adjacent set of the pivot vertex, \( G^A(\text{Adj}(x_k)) \), in the section graph \( G^A \equiv G^A(X^k) \) becomes a clique. Here, \( X^k \) represents the vertices which correspond to the submatrix \( A^k \), i.e. \( \{x_{k+1}, ..., x_N\} \). Alternatively, new edges, called filled edges, are introduced in the section graph \( G^A(X^k) \) between the vertices of \( \text{Adj}(x_k) \) if they did not already exist. In the graphical representation, successive section graphs are obtained for the
submatrices $A^k$ during elimination by removing the vertex which corresponds to the pivot, leaving the edges, which are naturally combined to yield the section graph for $A^k$.

This process is shown in Fig. 4-7 for the example of Fig. 4-3, where the section graphs are shown at the first three stages of elimination together with the evolution of the corresponding matrix. The fill-ins in the submatrices $A^k$ are denoted by the symbol $\otimes$ and correspond to the created edges represented by dashes in the section graphs $G^A(X^k)$. At the first stage of elimination ($k = 1$), vertex $x_1$ is removed from the graph. This causes the edges $\{x_1, x_4\}$ and $\{x_1, x_2\}$ to collapse onto the edge $\{x_2, x_4\}$; however, this results in no additional fill because the edge $\{x_2, x_4\}$ already exists. Similarly, edges $\{x_1, x_5\}$ and $\{x_1, x_2\}$ collapse onto $\{x_2, x_5\}$, and edges $\{x_1, x_5\}$ and $\{x_1, x_4\}$ onto $\{x_4, x_5\}$ without introducing additional fill. At the next stage of elimination ($k = 2$), two filled edges are introduced because of the collapse of edges $\{x_2, x_4\}$ and $\{x_2, x_3\}$ onto $\{x_3, x_4\}$, and of edges $\{x_2, x_4\}$ and $\{x_2, x_5\}$ onto $\{x_4, x_6\}$. Collapse of other edges does not introduce any additional fill. The rest of the elimination steps are similar.

The filled graph $G^F$ and the filled matrix $F = L + U$ are shown in Fig. 4-8, after the elimination process is complete. Note that a different labeling scheme would cause a different fill-in pattern because the vertices would be eliminated in a different order.

The following theorem formally establishes a relation between the graph of a matrix and its fill pattern.

**Theorem 4.1** Rose et al. [228]. Let $i > j$. Then $l_{ij} \neq 0$ if and only if there exists a path

$$x_i, x_{p_1}, \ldots, x_{p_l}, x_j$$

in the graph $G^A$ such that all subscripts in $\{p_1, \ldots, p_l\}$ are less than $j$. $\square$

The procedure outlined above for determining the fill pattern of a matrix from its
Figure 4-7: The first three stages for $LU$-decomposition for the matrix $A$ in Fig. 4-3 and the corresponding graph. Nonzero entries of the matrix are shown by $\bigcirc$. The fill-ins are shown by $\bigotimes$ in the matrix and are represented by dashed edges in the graph. $k$ is the elimination step or the pivot number.
graph results from Theorem 4.1. To see this, consider the situation in which \( x_{pi} = x_{pj} \) and \( a_{ij} = 0 \). Theorem 4.1 states that a filled edge is introduced between the vertices \( x_i \) and \( x_j \).

Since the graph of a matrix has such a close relationship with the matrix, the terms graph and matrix are used interchangeably in the rest of this Chapter. Each representation implies the other. To illustrate, elimination of a vertex in the graph is equivalent to forming the corresponding pivot column in the matrix, and introduction of the fill-in in the matrix is equivalent to introduction of the corresponding filled edges in the graph.

The concept of **supernodes** \([8, 10]\) is introduced to extend the ideas for graphs and elimination trees to more general indefinite linear equation systems. This involves grouping the vertices of the graph into supernodes or **supervertices** and treating the section graphs in these supernodes as dense. This approach has several advantages. The contiguous blocks of data corresponding to supernodes are treated as dense, so basic linear algebra subroutines (BLAS) \([78]\) can be used. These improve the speed of
computations, because the data structures are amenable to pipelining and vectorization. Furthermore, integer addressing is reduced; indirect addressing is not required for each single vertex within the supernodes since these data are treated as dense. Most importantly, pivoting may be performed within each supernode without destroying the sparsity structure outside the supernodes. This last property is central to the algorithm developed here, because pivoting within the supernode preserves the overall pattern of the sparsity. The rules which are developed for sparse matrix factorization of symmetric, positive-definite systems directly generalize to the indefinite systems which have symmetric structures, at least when the equations are grouped into supernodes, as is the case with the linear systems which arise in the context of finite or spectral element discretizations.

Elimination Trees

Elimination trees provide important structural information for matrix factorization. Liu [177] presents a comprehensive discussion on the importance of elimination trees in sparse factorization. In addition to their other useful properties which are discussed below, elimination trees help organize tasks and data for parallel implementation. The concept of the elimination tree is relatively recent. The term elimination tree was used by Duff [80], who used a structure somewhat different from what is widely accepted today as an elimination tree; and by Jess and Kees [161], who used a structure that is a special case of the elimination tree used here. The present definition of an elimination tree is attributed to Schreiber [240] and Liu [174]. Liu created the term to describe the structure defined by Schreiber [240].

Variants of the elimination tree have been referred to by different names in the literature. The term dissection tree was used by George [107] and Gilbert and Tarjan [119] for nested dissection orderings in the context of optimal matrix reorderings; Eisenstat et al. [87] introduced the term element merge tree in their element model for Gaussian elimination; and Duff and Reid [83] used the idea of an assembly tree.
for determining the assembly order in their multifrontal method. The row merge tree used by Liu [176] is an extension of the elimination tree for sparse orthogonal factorization. The notion of the elimination tree used here is the one introduced by Schreiber [240] and Liu [177] which is discussed extensively by Liu [177]. For the definitions of the tree terminology, the reader may consult Aho et al. [4]; however, the material presented here is intended to be sufficient.

In the discussion above, the reader was introduced to graphs and their connection to the discretized problem and the structure of system of equations for the problem. Elimination trees are closely related to filled graphs, $G^F$, and can be constructed from them, as explained below. The discussion here will be restricted to problems with connected graphs which are associated with irreducible matrices. Usually, disconnected graphs do not result from discretization of mathematical models for a single physical system. A disconnected graph implies that the system consists of several domains which do not interact. The presentation below easily generalizes to disconnected graphs because disconnected graphs consist of two or more connected components each of which can be treated separately as a connected graph.

A graph has a tree structure if there is only one path connecting any two vertices of the graph. The vertices of a tree may be ordered by picking a vertex to start. First, this vertex, which is called the root, is numbered. Then, the vertices connected to the root are numbered. These vertices are children of the root, and the root is the parent of these vertices. Then, the vertices which are connected to the children of the root are numbered. These are children of the previous vertices and are descendants of the root. The root is an ancestor of the children and, in fact, of all other vertices. The numbering of the vertices continues in this manner until all vertices are exhausted. The final vertices, where the numbering ends, are the leaves of the tree. A subtree is a vertex and all of its descendants. The most useful ordering for matrix factorization is produced by assigning the largest number to the root and the smallest numbers to the leaves. This ordering is referred to as a topological ordering [177] and starts from
the leaves and goes up to the root.

The elimination tree $T^A$ of a matrix is the graph of the filled matrix wherein only the most immediate off-diagonal entries, i.e. those closest to the diagonal, are kept and the rest of the off-diagonal entries are discarded. The elimination tree $T^A$ has the same vertices as $G^A$ or $G^F$, and it is a subgraph of $G^F$. The Parent[·] vector

$$\text{Parent}[j] = \min\{i > j \mid l_{ij} \neq 0\} \quad (4.7)$$

where $l_{ij}$ are the entries of the lower triangular factor $L$, represents the structure of the tree. The elimination tree for the matrix in Fig. 4-8 is shown in Fig. 4-9. Also, the associated filled matrix is shown where the off-diagonal entries closest to the diagonal are emphasized. The elimination trees for the ordered graphs of Figs. 4-4 and 4-5 are shown in Figs. 4-10 and 4-11, respectively.

The subtree of a vertex $x_i$ is denoted by $T^A[x_i]$. This subtree includes the node $x_i$ and all of its descendant's. The subtree for the root, $x_N$, is the whole tree, i.e. $T^A[x_N] = T^A$.

Since the elimination tree is a subgraph of the filled graph, the filled graph can be used to produce the elimination tree. There are various ways of reducing the filled graph to the elimination tree. In transitive reduction [4], one starts with the directed graph of the filled matrix. If there is an edge in addition to the directed path with $\ell > 1$ from one vertex $x_i$ to another $x_j$, the edge is redundant and is removed. This process is repeated for all redundant edges to produce the tree. In depth-first search, the search starts with an initial node $x$, which is marked as visited; then, each node adjacent to $x$ is searched, using a depth first search recursively [4, 252, 253]. The edges that lead to new, unmarked vertices during the depth-first search form a rooted tree, called a depth-first search tree. Liu [177] proved that the elimination tree $T^A$ of a connected graph $G^A$ is a depth-first search tree of the filled graph $G^F$ of $A$; here, the search starts with $x_N$ as the initial node. An easy way to construct the elimination tree from the filled graph is to consider each vertex $x_i$ in the filled graph.
Figure 4-9: The elimination tree and the associated filled matrix from which the tree was obtained for the graph and matrix shown in Fig. 4-8, where the band ordering was used. The shaded off-diagonal entries in the filled matrix represent the edges in the tree.
Figure 4-10: The elimination tree for the nested dissection ordering shown in Fig. 4-4.

Figure 4-11: The elimination tree for the minimum degree ordering shown in Fig. 4-5. Although in this case the tree structure is identical to the one for nested dissection (Fig. 4-10), minimum degree ordering usually gives longer, more slender trees than nested dissection.
once and to keep only one edge \( \{x_i, x_j\} \) such that \( x_j \in \text{Adj}(x_i) \), \( j > i \), and \( j \) is the minimum index which appears in \( \text{Adj}(x_i) \).

Elimination trees contain a great deal of useful information pertinent to sparse factorization. For example, the elimination tree gives information about the fill pattern of the matrix by the following theorem and corollary, which have been proven by Schreiber [240] and by Liu [177].

**Theorem 4.2** Schreiber [240]. If \( l_{ij} \neq 0 \), then the node \( x_i \) is an ancestor of \( x_j \) in the elimination tree. \( \square \)

**Corollary 1** Liu [177]. Let \( T^A[x_i] \) and \( T^A[x_j] \) be two disjoint subtrees of the elimination tree. Then for all \( x_s \in T^A[x_i] \) and \( x_t \in T^A[x_j] \), \( l_{st} = 0 \). \( \square \)

Although the elimination tree is useful, it does not completely replace the graph of the matrix. The graph is needed to complement the elimination tree for obtaining the fill pattern of a matrix. The following theorem by Liu [177] simplifies the role of the graph in determining the fill pattern by using the properties of the elimination tree; it follows from Theorem 4.1.

**Theorem 4.3** Liu [177]. Let \( i > j \). Then \( l_{ij} \neq 0 \) if and only if there exists a path

\[
  x_i, x_{p_1}, \ldots, x_{p_t}, x_j
\]

in the graph \( G^A \) such that \( \{x_{p_1}, \ldots, x_{p_t}\} \subseteq T^A[x_j] \). \( \square \)

The most important application of elimination trees for this thesis is in structuring parallel implementation of the solution of linear equation sets. Elimination trees easily identify the independent tasks and form a model for large-grain distribution of tasks to processors [161, 175]. The elimination tree captures the structure of data dependencies and identifies the tasks that can be performed in parallel. The following were proposed by Liu [177]:
Proposition 4.1 Liu [177]. Let $T^A[x_i]$ and $T^A[x_j]$ be two disjoint subtrees of the elimination tree $T^A$. The columns $L_{*,i}$ and $L_{*,j}$ can be computed in parallel with no overlap data access. □

Corollary 2 Liu [177]. All the leaf nodes in the elimination tree can be eliminated in parallel. □

Corollary 2 applies to various stages or levels of elimination. Once a set of the leaf vertices are eliminated, their parents, which belong to the next level of elimination, are considered; then in turn, these parents become subject to Corollary 2 and are treated as their leaf children were treated in the previous level. Nodes which can be eliminated in parallel are called parallel pivots [177]. Maximizing parallel pivot at each level maximizes parallelism. As a result of all these considerations, parallelism adds another complication to the problem of finding a good ordering. Not only does a good ordering need to minimize fill-in and be amenable to vectorization, but also a tree needs to be produced that has several leaves at each level, i.e. a short, fat tree, as Proposition 4.1 and Corollary 2 suggest.

4.2.4 Ordering

Ordering the equations, degrees-of-freedom, or the graph vertices are equivalent to each other and produce a sequence which is used in factorization of the linear equation set and the triangular solutions. The method used for ordering is important because it determines the fill pattern of the matrix and the graph. Because the zero entries which remain zero throughout factorization are ignored, the ordering is vital in determining the operation count and the required computer storage. The evolution of the fill pattern in the matrix affects the operation count; i.e. different ordering strategies result in different operation counts or computational complexities. Furthermore, different ordering schemes require different amounts of computer storage to hold the original matrix and the fill-in. Therefore, it is important to carefully evaluate differ-
4.2 Background and Choices

tent ordering strategies before choosing a particular one. Since the fill pattern of the
matrix can be determined beforehand when pivoting is absent, or restricted to within
the supernodes when pivoting is present, a good ordering scheme can be determined
before the factorization. There are many possible ordering strategies, including band,
envelope, minimum degree, quotient tree, one way dissection, and nested dissection.
Detailed descriptions (including references) can be found in George and Liu [111].
Band and envelope methods are described below because of their enormous popular-
ity and of their extensive use in direct solution of two-dimensional problems which
are discretized with finite elements. Nested dissection orderings also are described
because of their efficiency in reducing operation count and fill-in and because of their
large potential for parallelization. Furthermore, band and envelope methods have
been used in conjunction with the popular frontal solvers [153, 139, 140] used in clas-
sical implementations of the finite element method. Nested dissection orderings are
the basis for the CFS algorithm developed here.

Band and Envelope Methods

Band ordering and the very closely related envelope or profile methods are some of
the simplest and most widely used ordering schemes. The general object here is to
find an ordering which clusters the nonzero matrix entries close to the main dia-
ogonal. Therefore, the nonzero entries in the filled matrix also are clustered around the
main diagonal. These methods are not optimal in reducing the fill and the opera-
tion count, but they have been widely used [67, 92, 194] because of several reasons.
The data structures are simple; therefore, the overhead for data manipulation during
factorization is small, and the simple data structures make these methods amenable
to vectorization, giving large computation rates on vector supercomputers. Further-
more, the orderings themselves are easily obtained. Finally, row pivoting is incor-
porated without difficulty to extend the methods to asymmetric, indefinite equation
sets without destroying the band structure, albeit while somewhat expanding the
band. Thus, these methods have been very popular for their speed, for their efficient handling of large problems, and for their ease of implementation. Also, frontal methods [153, 139, 140] have been developed based on the envelope schemes, which use out-of-core storage to handle large problems. Therefore, in the more traditional computing environments of vector supercomputers, band and envelope schemes often have been the choice for direct solution of sparse, linear system of equations which arise in the context of solution of partial differential equations which are discretized with finite element methods.

However, recent advancements in parallel computers have made these schemes less attractive. The reason is simple. Band and envelope schemes have long and slender elimination trees. As a result, these methods are inherently more sequential than other ordering strategies such as nested dissection. The large-grained parallelism which the orderings with short, fat trees produce is absent from the band and envelope orderings: compare Figs. 4-9 and 4-10. This is not to say that no parallelism is possible in these cases. In fact, theoretical discussions exist which claim that there is considerable small-grain parallelism which can be exploited [100]; but in practice, the absence of the large-grained parallelism is a drawback. Some definitions and a discussion of band and envelope methods follow.

The \textit{ith bandwidth} of the matrix $A \in \mathbb{R}^{N \times N}$ is defined as [111]

\begin{equation}
\beta_i(A) = i - \min\{j \mid a_{ij} \neq 0\}
\end{equation}

and the \textit{bandwidth} of $A$ as [68]

\begin{equation}
\beta(A) = \max\{\beta_i(A) \mid 1 \leq i \leq N\}
\end{equation}

\begin{equation}
= \max\{|i - j| \mid a_{ij} \neq 0\}
\end{equation}

The \textit{band} of $A$ is the region in $A$ where the entries fall within the bandwidth from
the main diagonal. More precisely,

\[ \text{Band}(A) = \{ (i, j) \mid 0 < i - j \leq \beta(A) \} \]

(4.10)

An easy and common way to store a sparse, banded matrix is the diagonal storage scheme [186]. In this scheme, the band is stored in a rectangular array by rotating the band 45 degrees, as shown in Fig. 4-12, where the storage array \( S \in \mathbb{R}^{N \times 3\beta+1} \) is chosen large enough to allow row pivoting within the whole band. The entries of the sparse matrix which fall in the band are related to the entries of the storage array according to

\[ s_{i,j-i+m} \equiv a_{i,j} \]

(4.11)

where \( m \equiv \beta + 1 \). Equation (4.11) represents the band rotated 45 degrees clockwise. The band also can be rotated counterclockwise if that method results in a more optimal memory access when the entries of \( S \) are retrieved and stored in memory.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Full Matrix</th>
<th>Band Matrix Without Pivoting</th>
<th>Band Matrix With Pivoting</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LU$-Decomposition</td>
<td>$N^3/3$</td>
<td>$N/\beta^2$</td>
<td>$2N/\beta^2$</td>
</tr>
<tr>
<td>Forward or Backward Elimination</td>
<td>$N^2$</td>
<td>$2N/\beta$</td>
<td>$3N/\beta$</td>
</tr>
</tbody>
</table>

Table 4.1: Leading-order terms in operation counts for direct solution of a full matrix $A$ of order $N$ and the corresponding band matrix of order $N$ where the bandwidth $\beta \ll N$ [69].

The amount of storage required to store the band portion of $A$ is

$$S_b = N(3\beta + 1)$$  \hspace{1cm} (4.12)

if row pivoting is used within the band, and it is

$$S_b = N(2\beta + 1)$$  \hspace{1cm} (4.13)

if pivoting is not used within the band. When $\beta \ll N$, the leading-order terms in the operation counts are given as [69]

$$C_b \sim 2N/\beta^2$$  \hspace{1cm} (4.14)

with pivoting, and

$$C_b \sim N/\beta^2$$  \hspace{1cm} (4.15)

without pivoting. The leading-order terms for the operation counts of band orderings are compared with the operation counts of full matrices for $LU$-decomposition and triangular solutions in Table 4.1 [69]. The corresponding operation counts are summarized in Table 4.2, for an $n \times n$ centered, finite difference grid, where the bandwidth is $n$ and the matrix order is $N = n^2$.\)
4.2 Background and Choices

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Without Pivoting</th>
<th>With Pivoting</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU-Decomposition</td>
<td>$N^2$</td>
<td>$2N^2$</td>
</tr>
<tr>
<td>Forward or Backward Elimination</td>
<td>$2N^{3/2}$</td>
<td>$3N^{3/2}$</td>
</tr>
</tbody>
</table>

Table 4.2: Leading-order terms in operation counts for direct solution of a band matrix for an $n \times n$ centered, finite difference mesh where the order is $N = n^2$ and the bandwidth $\beta = n \ll N$.

![Figure 4-13: The envelope for the matrix of Fig. 4-3.](image)

The problem with the band ordering is that if $\beta_i$ varies greatly, the scheme is quite inefficient. Therefore, the envelope or profile methods have been developed which relieve this problem. The envelope methods, which the versatile frontal solvers use, enjoy the simplicity of the band methods without many of their disadvantages.

The envelope methods take advantage of the variation of the $i$th bandwidth ($\beta_i$) with $i$. The envelope of $A$ is defined, which traces the outermost nonzeros from the main diagonal, as

$$\text{Env}(A) \equiv \{i,j\} \mid 0 < i - j \leq \beta_i(A)$$

(4.16)

The envelope for the matrix in Fig. 4-3 is traced in Fig. 4-13. The profile or envelope
size of $A$ is simply the sum of all the $i$th bandwidths of $A$ as

$$|\text{Env}(A)| = \sum_{i=1}^{N} \beta_i(A)$$

(4.17)

In the absence of pivoting, the envelope of $A$ does not change during factorization; i.e. $\text{Env}(A) = \text{Env}(F)$, as proven elsewhere [111]. It follows from the definitions of band and envelope that $\text{Env}(A) \subseteq \text{Band}(A)$. Therefore, the complexity of the envelope methods is at most as large as the complexity of the band methods.

For easier analysis of the envelope methods, the concept of frontwidth is introduced. The $i$th frontwidth of $A$ is

$$\omega_i(A) \equiv |\{k \mid k > i \text{ and } a_{kl} \neq 0 \text{ for some } l \leq i\}|$$

(4.18)

That is, the $i$th frontwidth is the “distance” of the farthest nonzero in the $i$th column from the main diagonal. The $i$th frontwidth is similar to the $i$th bandwidth, except that the former refers to the farthest nonzero in the column $i$ and the latter to the farthest nonzero in the row $i$ in the lower triangular matrix $L$. The frontwidth or the wavefront of $A$ is [153, 194]

$$\omega(A) \equiv \max\{\omega_i(A) \mid 1 \leq i \leq N\}$$

(4.19)

Since by definition the envelope of $A$ traces the farthest nonzeros from the main diagonal of $A$, the envelope size also is determined from the $i$th frontwidths of $A$ as

$$|\text{Env}(A)| = \sum_{i=1}^{N} \omega_i(A)$$

(4.20)

The envelope and frontwidth also can be determined from the graph of the matrix by the following theorem and corollary, which are proven by George and Liu [111].

**Theorem 4.4** George and Liu [111]. For $i < j$, $\{i, j\} \in \text{Env}(A)$ if and only if
4.2 Background and Choices

Figure 4-14: The star-shaped section graph of the arc-length continuation for tracking a nonlinear solution in the parameter space.

\[ x_j \in \text{Adj} (\{x_1, \ldots, x_i\}). \quad \square \]

**Corollary 3** For \( i = 1, \ldots, N \), \( \omega_i (A) = |\text{Adj}(\{x_1, \ldots, x_i\})| \). \quad \square

The utility of envelope methods is most apparent in examples where the \( i \)th bandwidth varies largely. For example, the arc-length continuation method for tracking solutions in a parameter space [162] introduces an equation for the arc-length which is coupled to all of the degrees-of-freedom, resulting in an augmented problem. The section graph for this arc-length equation has a star shape, as shown in Fig. 4-14, where the center vertex represents the arc-length and the surrounding vertices represent the degrees-of-freedom, including the parameters in the equations. The minimum bandwidth ordering is compared to the minimum profile ordering for the augmented problem in Fig. 4-15. The minimum profile ordering, which results from numbering the center vertex last, gives an arrow-shaped matrix, which has the computational complexity, as well as the memory requirement, of \( O(N) \), whereas the band ordering gives a computational complexity of \( O(N^3) \) and the memory requirement of \( O(N^2) \) [111].

There are a number of algorithms which have been developed to reduce the envelope of a matrix [68, 106, 178, 172, 164, 118, 117]. Perhaps the most popular of these is the reverse Cuthill-McKee algorithm described in detailed by George and Liu [111]. The algorithm, which Cuthill and McKee [68] proposed, is based on minimizing the bandwidth. First, a starting vertex in the graph is picked and numbered \((x_1)\). Next, the vertices adjacent to \( x_1 \) are numbered. The unordered vertices adjacent
Figure 4-15: The minimum profile ordering compared to minimum band ordering of the graph shown in Fig. 4-14 [111].

to the ordered vertices continue to be numbered until the vertices are exhausted. In a later work, George [106] proposed reversing the above ordering; this reverse ordering produced a much superior ordering in terms of reducing the profile. Liu and Sherman [178] proved that this reverse ordering is never inferior to the original ordering in terms of reducing the profile. Choosing a good starting vertex $x_1$ is important; George and Liu [111] proposed an efficient heuristic algorithm for doing so.

The extension of the envelope scheme to include indefinite linear equation sets, which arise in the context of finite element discretizations, has been achieved by allowing pivoting within a limited region called the front [139, 140], thereby restricting the destruction of the envelope structure.

**Nested Dissection**

The nested dissection method is based on recursive dissection of the graph using separators. First, a separator, as defined in Section 4.2.3, is identified which divides the graph into several connected components. In turn, separators are found for each connected component, and the process is repeated recursively until no more dissection
is possible. If the process is stopped before complete dissection, an *incomplete nested dissection* ordering results. One advantage of a nested dissection ordering is that the elimination of the several connected components is independent, and hence it can be performed in parallel, as proven in the following theorem.

**Theorem 4.5** Let $S \subset X$ be a separator which divides the graph $G^A = \{X^A, E\}$ into $n$ connected components

$$G_1, G_2, \ldots, G_n$$

where $G_i \equiv G^A(X_i)$ and $X = \bigcup_{i=1}^n X_i \cup S$. Then, the graphs $\{G_i\}$ can be eliminated independently.

**Proof.** Consider the two vertices $x_i \in X_k$ and $x_j \in X_l$, where $i > j$. There is no path

$$x_i, x_{p_1}, \ldots, x_{p_l}, x_j$$

such that all the subscripts $\{p_1, \ldots, p_l\}$ are less than $j$ because any path from $x_j$ to $x_i$ needs to include a vertex in $S$ by definition, and any such vertex has an index larger than $j$. Therefore, $l_{ij} = 0$ by Theorem 4.1. $\square$

Note that the elimination of the separator $S$ generally does depend on the elimination of the connected components. Therefore, aside from the separator, the section graphs $\{G_i\}$ suffer no fill from one another and can be eliminated concurrently. An example is shown in Fig. 4-16 for a separator which divides the graph into two sections, where the associated matrix also is shown.

Finite and spectral element discretizations lend themselves naturally to nested dissection because the element boundaries can be chosen as separators as a result of the local coupling of the degrees-of-freedom within each element. George [107] applied the idea of nested dissection to an $n \times n$ regular grid. The computational complexities and storage requirements, which are given below, are based on such a grid.
Figure 4-16: A separator dividing the graph into two section graphs and the associated matrix $[111]$. 

$$
\begin{bmatrix}
A^{(11)} & 0 & A^{(1s)} \\
0 & A^{(22)} & A^{(2s)} \\
A^{(s1)} & A^{(s2)} & A^{(ss)}
\end{bmatrix}
$$
4.2 Background and Choices

<table>
<thead>
<tr>
<th></th>
<th>Nested Dissection Ordering</th>
<th>Band Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Requirement</td>
<td>$N \log_2 N$</td>
<td>$N^{3/2}$</td>
</tr>
<tr>
<td>Computational Complexity</td>
<td>$N^{3/2}$</td>
<td>$N^2$</td>
</tr>
</tbody>
</table>

Table 4.3: Leading-order terms for storage requirements and computational complexities of $LU$-factorization for band and nested dissection orderings.

George and Liu [111] showed that the storage required for direct solution of a problem associated with an $n \times n$ grid is

$$S_{nd} = O(N \log_2 N) \quad (4.21)$$

and the computational complexity is

$$C_{nd} = O(N^{3/2}) \quad (4.22)$$

The comparison of these two values with the results for the band ordering is shown in Table 4.3. Furthermore, George and Liu [111] in the following theorems proved that the storage requirement and operation count for nested dissection are optimal.

**Theorem 4.6** George and Liu [111]. The factorization of a matrix associated with an $n \times n$ grid requires at least $O(N^{3/2})$ operations. □

**Theorem 4.7** George and Liu [111]. The triangular factor of a matrix associated with an $n \times n$ grid has at least $O(N \log_2 N)$ nonzeros. □

Heuristic algorithms are usually used to generate nested dissection orderings, examples include the scheme presented in Section 4.3.3 for nested dissection of rectangular problems and the method presented by George and Liu [111] for automatic nested dissection of general problems.
4.2.5 Symbolic Factorization

Symbolic factorization includes determining the nonzero structure of the filled matrix $F$. This may be done by carrying out the factorization on the structure of $A$ symbolically. However, this process would be extremely inefficient because it would require the same number of operations as the actual numerical factorization. The structure of the filled matrix also may be determined in $O(\eta)$ operations, where $\eta$ is the number of nonzeros in the filled matrix, as shown by Heath et al. [134]. Once the structure of $F$ is determined, integer arrays are constructed for addressing the location of nonzeros.

As the discussion above on matrix graphs and elimination trees suggests, it is possible to deduce the fill pattern from the graph for the matrix. The methods, which are employed here and discussed below, use the graph for determining the fill and setting up the appropriate integer arrays for addressing.

4.2.6 Numerical Factorization

The numerical factorization of $A$ was depicted in Fig. 3-16 for either a dense matrix, or for a sparse matrix wherein the sparsity structure is ignored. At each pivot step $k$, the algorithm involves scaling the entries below the main diagonal entry ($a_{kk}$) with the value of the main diagonal entry as

$$a_{ik} := -\frac{a_{ik}}{a_{kk}} \text{ for } i > k \quad (4.23)$$

and subsequently updating the submatrix $A^k$ as

$$a_{ij} := a_{ij} + a_{ik} * a_{kj} \text{ for } i, j > k \quad (4.24)$$

In the above algorithm, the pivot column ($\{a_{ik}\}$) is formed first; then, the all of the submatrix $A^k$ is immediately updated before proceeding to form the next pivot
column \( \{a_{i,k+1}\} \). This algorithm is called \textit{submatrix-update} because the submatrix \( A^k \) is updated at each elimination step. This is similar to the \textit{submatrix-Cholesky} terminology used for Cholesky factorization [134].

The above scheme is not the only possible sequence for factorization, although it has been the most commonly used sequence for \( LU \)-factorization with pivoting for dense and band matrices in serial computing environments. If the matrix \( A \) is structurally symmetric, sparse and indefinite, and if the pivoting is restricted to within each group of equations in supernodes, then methods other than the above submatrix-update give better results for parallel implementation.

According to the description by Heath et al. [134], Gaussian elimination on a matrix \( A \) is described as a triple nested loop around the update statement

\[
a_{ij} := a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}
\]

The variations of the elimination algorithm are classified based on the order in which the indices \( i, j, \) and \( k \) are nested:

1. \textit{Row-update}. The index \( i \) is the outer loop, so the algorithm is expressed as

\[
\text{for } i = 1 \rightarrow N \\
\text{for } k = 1 \rightarrow i - 1 \\
a_{ik} := -a_{ik}/a_{kk} \\
\text{for } j = k + 1 \rightarrow N \\
a_{ij} := a_{ij} + a_{ik} \ast a_{kj}
\]

2. \textit{Column-update}. The column index \( j \) is the outer loop, so the algorithm is expressed as

\[
\text{for } j = 1 \rightarrow N \\
\text{for } k = 1 \rightarrow j - 1 
\]
for $i = k + 1 \rightarrow N$

$$a_{ij} := a_{ij} + a_{ik} * a_{kj}$$

for $i = j + 1 \rightarrow N$

$$a_{ij} := -a_{ij} / a_{jj}$$

3. Submatrix-update. Here, the pivot index $k$ is the outer loop, and the algorithm is expressed as

for $k = 1 \rightarrow N$

for $i = k + 1 \rightarrow N$

$$a_{ik} := -c_{ik} / a_{kk}$$

for $j = k + 1 \rightarrow N$

$$a_{ij} := a_{ij} + a_{ik} * a_{kj}$$

These three algorithms are shown graphically in Fig. 4-17.

Row-update resembles the solution of a triangular system at each step. Its counterpart for symmetric, positive-definite factorization, row-Cholesky, is seldom used because of the difficulties in designing compact row-oriented data structures which can be accessed during factorization [174]. Furthermore, it is difficult to vectorize or parallelize the solution of triangular systems as Heath et al. [134] argue and the results in Chapter 5 demonstrate. Since by virtue of these complications row-Cholesky is not often used in practice, there will be no further discussion on the row-update method.

The column-update algorithm is far superior. This algorithm is called a left-looking, or a demand-driven algorithm [134]. It is left-looking because the matrix entries to the left of the current column, which is being updated, are used for the update; in other words, the updates are accumulated by the entries to the “left” of the current column. It is demand-driven because the updates to each column are held until the column becomes current, and further elimination cannot proceed until
Figure 4-17: Variations of the Gaussian elimination algorithm based on the method of update: (a) row; (b) column; and (c) submatrix, based on the presentation by Heath et al. [134] for Cholesky factorization.
the current column is updated. It is customary to refer to this algorithm as the *fan-in* algorithm [9], since the operation at the current step involves combining the effects of the previous columns to update the current column. The column-update algorithm is desirable in a parallel computing environment because it can reduce the communication in the manner discussed below.

In the submatrix-update algorithm, once the elimination operation on the $k$th column is complete, all of the remaining columns, which are to the right of the current column, are updated. For this reason, this algorithm is called a *right-looking* algorithm; it also is known as a *data-driven* algorithm because the current column is used to make all of the modifications it incurs on the subsequent columns as soon as its elimination is complete. It is sometimes called a *fan-out* algorithm, because the single current column produces updates for all of the subsequent columns.

A fan-in algorithm usually is more communication-efficient than a fan-out method in a distributed memory multiprocessor environment, given an appropriate ordering and an appropriate assignment strategy for division of the problem among the processors. As an example, consider the nested dissection ordering represented by Fig. 4-4, where each of the vertices in the graph is a supernode; that is, each vertex represents a number of equations. The elimination tree for this graph is shown in Fig. 4-10. Suppose the equations for this problem are partitioned among four processors of a distributed memory machine such that leaf vertices $\{x_1, x_2, x_3, x_4\}$ are assigned to the corresponding processors, and the ancestors of these leaves are partitioned among the processors which hold their children. For example, vertex $x_5$ is divided among processors $p_1$ and $p_2$ which hold the children of vertex $x_5$, namely the leaf vertices $\{x_1, x_2\}$. The elimination of the leaf vertices introduces updates for the separator vertices $S = \{x_5, ..., x_9\}$. Consider elimination of vertex $x_2$ by processor $p_2$. Elimination of the equations in $x_2$ introduces updates for equations in vertices $\{x_5, x_8, x_9\}$. Vertex $x_5$ is shared by processors $p_1$ and $p_2$, and vertices $\{x_8, x_9\}$ by all the processors.

If the submatrix-update, which is a fan-out algorithm, is used, processor $p_2$ sends
4.2 Background and Choices

updates to all other processors at each step \( k \) of the elimination process for \( x_2 \); this is a broadcast operation at each step \( k \). Therefore, \( O(c_2) \) broadcasts are needed, where \( c_2 \) is the number of columns in the matrix associated with the vertex \( x_2 \). The total number of messages sent is \( O(c_2 P) \) during this step, where \( P = 4 \).

If the column-update, which is a fan-in algorithm, is used, processor \( p_2 \) does not send any updates during the elimination of vertex \( x_2 \). Updates are collected by other processors from \( p_2 \) when the vertices \( \{x_5, x_8, x_9\} \) are to be eliminated. The number of communications for these updates is \( O(c_{x_2} P) \), where \( c_{x_2} \equiv c_5 + c_8 + c_9 \) is the number of the columns of the matrix associated with the separator vertices \( \{x_5, x_8, x_9\} \). In practical situations, \( c_2 \gg c_{x_2} \), and usually \( c_2 \gg c_{x_2} \) because \( c_{x_2}/c_2 \sim \mathcal{P}_2/\mathcal{A}_2 \), where \( \mathcal{P}_2 \) is the perimeter and \( \mathcal{A}_2 \) is the area of the subdomain; therefore, the number of messages is reduced greatly by the fan-in update method.

Heath et al. [134] showed the communication pattern for a model finite element problem on a \( 15 \times 15 \) grid \((N = 225)\), where the nested dissection ordering is used, and the problem is divided among 8 processors of an Intel iPSC/2 hypercube based on the subtree-to-subcube mapping strategy, described below. Figures 4-18 and 4-19, which are reproduced from these authors' work, show the communication patterns for the fan-out and the fan-in algorithms, respectively. In these figures, the horizontal axis represents the execution time and the vertical axis the different processors. A solid horizontal line means that the processor is working; where this line is absent, the processor is idle; and a slanted line represents communication from one processor to another. Clearly, the fan-in algorithm reduces the communication relative to its fan-out counterpart.

The elimination process need not precisely follow the variations described above. Hybrids of the row-update, column-update, and submatrix-update are sometimes more useful. The algorithm developed here uses a hybrid of column-update and submatrix-update methods. Also, frontal [153, 139, 140] and multifrontal [83] methods are variations of submatrix-update, where the updates are not applied to all of
Figure 4-18: Communication pattern of fan-out algorithm for a model problem [134].

Figure 4-19: Communication pattern of fan-in algorithm for a model problem [134].
the remaining columns, but are kept in the front until the assembly of the target column is complete. This minimizes I/O traffic to the out-of-core storage space where the inactive portions of the matrix are stored.

4.2.7 Triangular Solutions

Triangular solutions complete the solution procedure by modifying the right side vector \( b \), using the \( L \) and \( U \) matrices, to produce the solution vector \( x \). The ordering of the right side vector is predetermined by the ordering used for the matrix \( A \), so there are no steps which correspond to ordering or symbolic factorization. Triangular solutions have much smaller computational complexities. As shown in Table 4.2 for band ordering, the leading-order term for triangular solutions is \( O(\sqrt{N}) \) smaller than the leading order term for \( LU \)-factorization. As a result, the triangular solutions usually consume a small fraction of the total solution time. This may change, however, if many triangular solutions are needed for each \( LU \)-factorization, as in modified Newton iterations (Fig. 2-11), where there are many triangular solutions per factorization step. Also, triangular solutions are harder to parallelize; i.e. there is less natural concurrency. This is because the main step in triangular solutions is in two nested loops, whereas the main step in \( LU \)-factorization is in three nested loops. Even parallelization of triangular solutions of dense systems is not easy. Heath and Romine [133] and Eisenstat et al. [86] showed that careful pipelining techniques are required to achieve efficiency rates as low as 50 percent for large dense systems on distributed memory processors. Parallelism is harder to achieve for sparse systems. There are fewer nonzero entries, and therefore less work to distribute to the processors: general sparse matrix triangular solutions require \( O(\eta) \) operations rather than \( O(N^2) \) operations for the dense systems, where \( \eta \) is the number of nonzeros in the sparse matrix and \( N \) is the order of the dense matrix [134]. Complicated sparsity patterns also make it more difficult to exploit parallelism.

Using the elimination tree for distribution of tasks helps improve the parallel
efficiency of the triangular solutions; however, as will be explored in Chapter 5 the efficiencies are smaller than 50 percent; cf. George et al. [108], Lucas et al. [180].

4.2.8 Data Distribution and Task Scheduling

Problem solving on a distributed memory parallel processor requires division of tasks and data among the processors, called scheduling. The primary goal is to reduce the solution time; thus a good division of tasks balances the load among the processors so that idle time is minimized. A good division of data also minimizes communication traffic so that the time spent sending, receiving, and waiting for messages is reduced. Task and data distribution are interdependent because tasks manipulate data. Therefore, the programmer must coordinate the divisions of tasks and data. programmer.

There are two types of scheduling possible: static and dynamic. Static allocation of tasks is a deterministic model [222]. The precedence relations for the tasks and the required execution time for each task are known beforehand. Therefore, the scheduling is performed before the execution starts and does not change throughout the execution.

Predictions of the execution time of the tasks is usually difficult or even impossible using the deterministic model. Moreover, in some cases, new tasks may appear based on the evolution of the problem. In either case, a dynamic, real-time allocation of tasks to the processors is required. Consider parallel LU-decomposition of a general sparse system, wherein pivoting is used within the whole matrix for numerical stability. Pivoting rearranges the structure of the matrix based on the numerical values of the nonzero entries. These values evolve during factorization, so there is no way of knowing beforehand the pattern of pivoting, and the evolution of the sparsity pattern of the matrix follows accordingly. Obviously, the division of tasks for load balancing cannot be known before the factorization starts.

Dynamic allocation of tasks is a much harder problem than static allocation and
usually is avoided if possible. In the case of \textit{LU}-factorization, pivoting within the whole matrix is abandoned in favor of more restricted pivoting only within limited regions (supernodes) which are treated as dense; for scheduling, a static task allocation is used. Fortunately, numerical stability usually does not require very large pivoting regions for most practical applications, and this approach is stable as well as simple and efficient.

Several static scheduling schemes have been used for sparse factorizations. Usually the elimination tree is used for information on data dependencies. One way to distribute the work based on elimination trees is to map the vertices of the tree in a wrap-map fashion to the processors, as shown in Fig. 4-20 [134]. This mapping is efficient in balancing the load, but it results in a large message volume during factorization. The method used by the CFS algorithm is to map the vertices of the tree onto the processors. A description of this method follows.

Based on Corollary 2, the leaf vertices of the elimination tree are assigned to processors, and the elimination of these vertices is completely parallel. If there are more leaf vertices than processors available, subtrees are chosen from a level of the elimination tree which equal the number of processors. In the next level of the tree, which consists of the parents of the leaf vertices, or the subtrees assigned to the processors, there are usually fewer vertices available, so either fewer processors may be employed, or processors cooperate in eliminating those vertices. The latter method increases parallelism, and therefore algorithm efficiency. The processors which are assigned to the leaf vertices cooperate in eliminating the parent vertex. This process goes all the way to the root: where two subtrees merge into a single subtree, their processor sets are merged [134]. Whereas elimination of each leaf vertex by each processor is a coarse level of parallelism, cooperation by processors for eliminating any other vertex is parallelism at a finer degree. The above assignment is called the \textit{subtree-to-subcube} assignment strategy by George et al. [113] and is shown schematically in Fig. 4-21.
Figure 4-20: The elimination tree wrap-mapped onto the processors [134].
The term subcube was used because of the hypercube architecture of the parallel computer used by these authors. Nevertheless, the results can be directly generalized to other machines by using the term subcube more loosely to refer to any connected section graph of the parallel processing network; the graph associated with the parallel architecture has the processors as its vertices and the network as its edges. George et al. [113] have shown that for the subtree-to-subcube assignment and for the fan-out distributed factorization algorithm for model problems on $n \times n$ grids, communication volume can be limited to $O(Pn^2)$, which is asymptotically optimal. Gao and Parlett [103] proved the slightly stronger result that the communication volume is $O(n^2)$ per processor, and thus the communication is balanced among the processors. The subtree-to-subcube mapping is simple to apply to regular problems ordered by standard nested dissection. More irregular problems are harder deal with; however, some progress has been made by Geist and Ng [105] and Sadayappan and Viswanathan [236].

4.3 Concurrent Factorization and Storage (CFS) Algorithm

This section includes a detailed description of the concurrent factorization and storage (CFS) algorithm. This algorithm is developed for MIMD parallel computers with powerful processors. It uses incomplete nested dissection for domain decomposition among the processors as well as for ordering the equations in the subdomain of each processor. Task and data distribution are static and precede $LU$-factorization, forward elimination, and backward elimination. The algorithm is multilevel in all of its steps. All solution steps including nested dissection ordering, symbolic factorization, $LU$-factorization, and triangular solutions are performed hierarchically. The details follow.
Figure 4-21: An example of the subtree-to-subcube mapping for the tree of Fig. 4-20, as given by Heath et al. [134].
Figure 4-22: Domain decomposition by nested dissection of a rectangular two-dimensional domain. A binary reflected Gray code is used for numbering the subdomains to reduce hypercube communication [42].

4.3.1 Overview

For simplicity, consider a mathematical model for transport processes described in a two-dimensional geometrical region represented by the rectangular domain shown in Fig. 4-22. This region is partitioned into \( P \) quadrilateral subdomains, each of which is assigned to a processor of the MIMD parallel computer. The equations that correspond to each subdomain are ordered using incomplete nested dissection [111]. Then, each point on the elimination tree for the nested dissection ordering corresponds to a supernode or a group of equations [192].

For physical problems, the mathematical model is usually composed of differential and algebraic equations, integral constraints and boundary conditions. Finite element methods are used to define local approximations of the field variables and to discretize the partial differential equations and boundary conditions. The details of these discretizations are not important for the discussion here, as long as the in-
interpolating functions for the field variables have compact support within a group of elements. The discretized problem is a large set of nonlinear algebraic equations that is represented as

$$R(x) = 0$$

(4.25)

where $R \in \mathbb{R}^N$ and $x \in \mathbb{R}^N$, where $N$ is the total dimension of the discrete field variables. The variables and equations are associated with specific processors according to

$$x = \{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(P)}\}$$

and

$$R = \{R^{(1)}, R^{(2)}, R^{(3)}, \ldots, R^{(P)}\}$$

where the dimensions of the variables $\{x^{(i)}\}$ and equations $\{R^{(i)}\}$ associated with each processor depends on the number of finite elements allocated to it.

Equations (4.25) are solved by Newton's method using direct $LU$-decomposition of the Jacobian matrix at each Newton iteration. The Jacobian matrix is written as a partitioned matrix among the $P$-processors as

$$A = \begin{bmatrix}
A^{(ii)} & A^{(ie)} \\
A^{(ei)} & A^{(ee)}
\end{bmatrix}$$

(4.26)

where the Jacobian matrix is organized into data that is entirely interior to each subdomain – the \textit{interior matrix} $\{A^{(ii)}, A^{(ei)}, A^{(ie)}\}$ – and data that corresponds to the borders shared by two or more subdomains (the separators in Fig. 4-22) – the \textit{exterior matrix} $A^{(ee)}$. The superscript $i$ denotes the interior and $e$ denotes the exterior. Two superscripts are used for matrix representations because the matrices may represent the coefficients of unknowns in a region for equations in a different region. For example, $A^{(ei)}$ represents the coefficients of interior unknowns as they appear in exterior equations. The exterior matrix is partitioned among adjacent processors to minimize
communication and memory requirements. The data structures used for storage of the interior and exterior matrices are described below. Matrix data corresponding to each supernode is stored in block form to facilitate vectorization and pivoting during the \textit{LU}-factorization.

The algorithm for the \textit{LU}-decomposition of the matrix, shown in Eq. (4.26), is operationally equivalent to direct computation and factorization of the Schur complement [123]. The formation of the Schur complement corresponds to the \textit{LU}-decomposition of the interior matrices of each processor \(\{A^{(ii)}\}\) and to the update of the exterior matrix \(A^{(cc)}\) by those factors. Data is communicated from each processor to form the exterior matrix using the fan-in method of Ashcraft et al. [9]. A hybrid fan-in/fan-out algorithm is used for updating the exterior matrices during factorization and subsequent updating. As demonstrated in Chapter 5, the fan-in and the hybrid fan-in/fan-out methods of update reduce the number of messages and the total message volume for \textit{LU}-decomposition by approximately factors of 2 and 3 over the algorithm of Mu and Rice [201], which was developed using a grid-based, subtree-to-subcube assignment of exterior matrix elements to the processors.

\subsection*{4.3.2 Domain Decomposition by Nested Dissection}

In Section 4.2.4, it was stated that the union of element boundaries which traverses the whole domain is a separator for the graph of a problem discretized with finite or spectral elements. Choosing such a separator decomposes the domain into any number of subdomains. Each of these subdomains corresponds to a supernode in the elimination tree. Further nested dissection of each subdomain arranges each supernode into subtrees. Suppose the separator divides the domain into two subdomains. Each supernode and its associated subtree are assigned to a processor. The elimination of each subtree is independent by Theorem 4.5, and is performed completely in parallel. The elimination of the separator is done cooperatively. This nested dissection of the domain into subdomains, which are assigned to the processors, is identical
to the subtree-to-subcube mapping described in Section 4.2.8.

A schematic of a geometrical domain which has been dissected four times is shown in Fig. 4-22. The circular regions are the subdomains produced by the dissection, and the rectangular regions are the separators which consist of the boundaries of the subdomains [111]. Because the dissection is performed such that the separators fall on the element boundaries, the separators are only one finite element node wide.

For a parallel computer, the subdomains of Fig. 4-22 are assigned to individual processors, where each processor is responsible for its assigned subdomain. The numbering for the subdomains in Fig. 4-22 is based on the well-known binary reflected Gray code [42] to minimize communication in a hypercube architecture by mapping the nearest neighbor processors in the hypercube onto the two-dimensional domain. The responsibility for the separators is shared among the processors. The separators that divide the processor subdomains are called the exterior separators, and their corresponding data is arranged in an exterior matrix. The numbers in Fig. 4-22 correspond to the supernodes on the elimination tree shown in Fig. 4-23. The leaves of the tree are the subdomains assigned to individual processors, and the ancestors of the leaves are the separators. Elimination takes place in the sequence corresponding to the levels shown in Fig. 4-23. The equations in the set corresponding to the supernodes are independent of each other at each horizontal level in this tree and so can be eliminated concurrently.

The subdomains assigned to the processors are large enough to have considerable sparse structure of their own, i.e. there are many finite or spectral elements in each subdomain. The data in the subdomain of each processor need to be arranged in such a way as to account for this sparsity. Nested dissection is chosen for this purpose as well. The data for the assigned subdomain to each processor is arranged based on incomplete nested dissection into interior separators and subdomains. This means that each subdomain in Fig. 4-22 is dissected, and the leaves in the tree of Fig. 4-23 are decomposed further into smaller supernodes so that each leaf in Fig. 4-23 is turned
4.3 CFS Algorithm

Figure 4-23: Elimination tree for the domain shown in Fig. 4-22.

into a subtree. The nested dissection is still not complete; the interior subdomains still contain one to several elements, hence they correspond to many equations. This interior nested dissection does not add to the parallelism of the algorithm; it only serves to lower the computational and memory requirements of a processor for solving its assigned subdomain. The level of incompleteness for nested dissection is limited by a finite element. The dissection cannot continue inside an element. Therefore, the size of the smallest subdomain is the number of degrees-of-freedom inside an element excluding the borders of the element. Because the dissection usually is not carried out down to the level of an element, the smallest subdomains still consist of several elements. Coarser dissection makes larger supernodes; therefore, more choices are available for pivoting, and larger data vectors are available for better vectorization. On the other hand, coarser dissection accounts less for the sparsity; therefore, it results in larger operation counts and less efficient use of the memory. The level of dissection is determined empirically.

Each processor must account for the equations in its assigned subdomain, which
correspond to a leaf in Fig. 4-23, and for the equations in the exterior separators depicted as its ancestors in Fig. 4-23. Because the tree in Fig. 4-23 is binary, the number of supernodes is halved at each level in ascending the tree from the leaves toward the root. Therefore, two processors share a first-level exterior separator, four processors a second-level one, etc. In CFS, the processors that share an exterior separator divide the separator's data in the exterior matrix column-wise in a wrap-map fashion [52]. To minimize communication, the columns in an exterior supernode are divided among the processors in a sequence so that the columns in the children, which are directly above a column in an ancestor, belong to the same processor that holds the column in the ancestor, as shown in the example of Fig. 4-25.

The matrix data for the supernodes are arranged in contiguous blocks. This has the advantages of both facilitating pivoting and being more amenable to pipelining and vectorization within the blocks. Row pivoting is allowed within each supernode. Therefore, the level of dissection must set aside large enough equation sets for the supernodes to ensure numerical stability; the size is problem-dependent and must be determined empirically. The results in Chapter 5 indicate that the smallest size of the supernodes for ensuring numerical instability is not large. In fact, no numerical instabilities were observed for the test problems presented in Chapter 5, even when nested dissection was carried out down to the level of a finite element, where the smallest supernodes consisted of only a few equations. The use of row pivoting within a limited region is common practice in other algorithms for \( LU \)-decomposition using band or frontal storage methods.

Data Organization

The data for each processor are organized into interior and exterior matrices. The exterior matrix \( A^{(ee)} \) corresponds to the exterior separators, \( S_e \), and the interior
4.3 CFS Algorithm

matrix

\[
\begin{bmatrix}
A^{(ii)} & A^{(ie)} \\
A^{(ei)} & A^{(ee)}
\end{bmatrix}
\]

corresponds to the interior subdomains and interior separators, \( X_i \). The data for the borders of subdomains, corresponding to \( G^F(\text{Adj}(X_i)) \), are only stored in the exterior matrix because these borders are parts of the exterior separators; accordingly, the interior matrix has a square section missing from its bottom right corner. The missing section corresponds to this border data.

A more detailed representation for the overall sparsity of the matrix \( A \) is

\[
A = \begin{bmatrix}
A^{(11)} & 0 & \cdots & A^{(1e)} \\
0 & A^{(22)} & \cdots & A^{(2e)} \\
\vdots & \vdots & \ddots & \vdots \\
A^{(e1)} & A^{(e2)} & \cdots & A^{(ee)}
\end{bmatrix}
\]

Each of the submatrices \( A^{(ii)}, A^{(ie)}, A^{(ei)}, \) and \( A^{(ee)} \) has considerable sparsity pattern of its own. The sparsity of the exterior matrix relevant to the subdomain of processor 9 in Fig. 4-22 is shown in the example of Fig. 4-24. The considerable sparsity of the interior matrix also is exploited. Furthermore, the columns of the exterior matrix are divided among the processors which share the separators. The sparsity of the interior matrix, along with the columnwise division of the exterior matrix, is shown in the example of Fig. 4-25.

4.3.3 Preprocessing

Preprocessing involves producing the domain decomposition map by nested dissection, ordering the subdomains by incomplete nested dissection and formulating the integer addressing arrays. These steps are the same as the ordering and symbolic factorization
Figure 4-24: The sparsity of the exterior matrix pertinent to the subdomain of processor 9 in the example of Fig. 4-22.

steps described in Sections 4.2.4 and 4.2.5.

The ordering and symbolic factorization, which together are referred to as indexing [201], are performed hierarchically, thereby taking advantage of the hierarchical structure of nested dissection. That is, the overall domain is divided by nested dissection among the processors, without any division of the subdomains assigned to the processors. Integer addressing arrays are set up for the nonzero blocks in the exterior matrix. The resulting data is distributed to the processors. Each processor orders its subdomain based on incomplete nested dissection and sets up integer addressing arrays for the nonzero structure of its interior data. The incomplete nested dissection of the subdomains by processors is strictly local, allowing the indexing of the interior subdomains to be performed completely in parallel.

A Heuristic Nested Dissection Method for Rectangular Domains

A simple algorithm is devised for producing the nested dissection map and the graph of the matrix. More complicated heuristics have been developed [111, 107, 110]. The main merit of this algorithm is that it takes advantage of the hierarchical nature of the nested dissection: the overall domain is decomposed into subdomains, $X_i$, and
Figure 4-25: The sparsity of the exterior matrix pertinent to the subdomain of processor 9 in the example of Fig. 4-22 shown in more detail. The sparsity of the interior matrix also is shown.
the exterior separators, $S_e$, and this information is sent to the processors; then, each processor uses the same algorithm to order its subdomain based on the incomplete nested dissection ordering.

The algorithm takes the total level of dissection as its input. The order of dissection with respect to $i$- and $j$-axes may be specified; if not, the algorithm alternates between $i$- and $j$-dissections. A map is generated for the matrix, such as the one in Fig. 4-22. The map shows the ordering of the subdomains and separators without any information about the connectivity. The subdomains in the map are numbered based on the binary reflected Gray code [134] to minimize communication in a hypercube computer architecture. The Intel iPSC/860 which is used for numerical experiments described in Chapter 5 has hypercube architecture. The filled graph for the problem is generated by considering subdomains and separators in the map, each of which corresponds to a supernode in the graph and the elimination tree of the matrix. The separators fall on element boundaries, and the elements have local support; therefore, the adjacent set for each subdomain $X_i$ in the filled graph comprises those portions of the separators which border that subdomain, as shown in the example of Fig. 4-26. Similarly, the adjacent set for a separator $S_e$ consists of the portions of the separators which surround $S_e$, as shown in the example of Fig. 4-27.

The map is stored in a rectangular array. The graph is stored by two vectors. The first vector is $g_1 = \{x_i, \text{Adj}(x_i)\}$, where $x_i$ is the $i$th supernode. The second vector is $g_2 = \{f_i\}$ where $f_i$ points to the position of the first $x_i$ in vector $g_1$. This storage scheme is the adjacency structure method for computer representation of graphs, as presented by George and Liu [111], and shown in the example of Fig. 4-28. However, the adjacency structure in CFS is set up for the filled graph $G^F$ rather than for the original graph $G^A$. Other arrays are set up which contain the number of degrees-of-freedom in each supernode based on the number of elements and the degrees-of-freedom per element.

The adjacency structure and the size of the supernodes together are used to set
Figure 4-26: The adjacent set for the subdomain 6 in the filled graph of the example which is shown in Fig. 4-22.

Figure 4-27: The adjacent set for the separator 22 in the filled graph of the example which is shown in Fig. 4-22.
Figure 4-28: The adjacency structure used for computer representation of the graph in Fig. 4-4 which was ordered by nested dissection.

aside storage for the sparse matrix $F$ and to set up integer addressing arrays which identify the position of the nonzero entries in the filled matrix.

The procedure described above is performed twice: first, for decomposing the overall domain on the front-end computer, and second, for ordering each subdomain by the corresponding processor. Upon completion of this procedure, storage is set aside for interior and exterior matrices and integer addressing arrays are set up which are used during the solution procedure to indicate the position of the nonzero blocks in the matrix and the right side vector.

4.3.4 Formulation

Formulation is completely parallel. The stiffness matrix $A$ and the forcing vector $b$ in Eq. (3.12) are assembled for each element and added to the appropriate place in the interior submatrix by each processor. The entries that belong to the exterior matrix are stored in an array for later retrieval. In order to reduce communication overhead, these entries are not added to the appropriate address in the exterior matrix. After the elimination of the interior is complete, these entries are used in formulating the columns of the exterior matrix one at a time, updating the columns by the factors of the interior matrix and adding the column to the appropriate address in the exterior matrix. This last step may require communication.
4.3.5 Multivariable Sets of Differential Equations

The CFS algorithm is directly applicable to single or multivariable differential equations. In the finite or spectral element discretizations, each element may have a number of variables and a number of equations associated with it. Each of these variables may have a different order for the interpolating polynomials that approximate it; this is the case for mixed element methods used for flow problems, which are treated in Chapter 5. The indexing algorithm does not change at all; only the size of the supernodes becomes larger because there are more degrees-of-freedom per supernode. The width of the separators varies in size from one to the number of unknowns per finite element node. This growth increases the communication traffic compared to a similar size problem with only one degree-of-freedom per finite element node; however, the effects are not large in most practical situations, as will be demonstrated in Chapter 5. The algorithms for $LU$-decomposition, forward elimination, and backward elimination are not specific to single or multivariable systems.

4.3.6 $LU$-Factorization

The $LU$-factorization is computed sequentially on levels starting with the leaves of Fig. 4-23 and proceeding toward the root. Each processor first formulates the data for its assigned subdomain and stores the values belonging to the interior matrix in appropriate addresses. The processor then factors its interior matrix. Each of these steps is completely parallel.

The exterior matrix is formed next. Each processor forms the updates to the parts of the exterior separators bordering its subdomain caused by factorization of the interior matrix. Alternatively, updates by the adjacent set of the subdomains are formed. This procedure is done one column at a time: the data, which were formulated first, are fetched for the column, and the column is updated by the factors of the interior matrix. Then, the column is added to the appropriate address in the exterior matrix. This step may require communication if the column of the exterior
matrix where the update column is added belongs to a different processor. Note that this process of forming and updating the exterior matrix is a fan-in procedure, so it yields very efficient communication [9].

The above process is operationally equivalent to the formation of the Schur complement [123]. For a subdomain \( X_i \), the interior matrix

\[
\begin{bmatrix}
  A^{(ii)} & A^{(ie)} \\
  A^{(ei)} &
\end{bmatrix}
\]

is factored to produce the filled, interior matrix

\[
\begin{bmatrix}
  F^{(ii)} & U^{(ie)} \\
  L^{(ei)} &
\end{bmatrix}
\]

Subsequently, the submatrices \( L^{(ei)} \) and \( U^{(ie)} \) are multiplied and added to the formulation of the exterior matrix \( A^{(ee)} \), thereby updating the exterior matrix so that its factorization can start.

At this point, the factorization will have been completed only for the leaves in Fig. 4-23, i.e. for level 0. Next, the exterior matrix is factored one level at a time; in Fig. 4-23, this process proceeds from level 1 to level 4 and modifies the exterior matrix \( A^{(ee)} \) into \( F^{(ee)} = L^{(ee)} + U^{(ee)} \). The processors that share a separator cooperate in factoring that separator. Thus, when two subtrees merge, the corresponding processor sets merge too, as described in Section 4.2.8 for the subtree-to-subcube mapping. For example, processors 0 and 8 factor separator 16 cooperatively, as depicted in Fig. 4-22 and Fig. 4-23. Several modes of parallelism are possible for this step because more than one processor is available for factoring each separator supernode, including supernodes 16-30 shown in Fig. 4-23. The first mode of parallelism is the simplest and most coarse: at each level (except the last) several separators are factored simultane-
ously. The second mode of parallelism is finer: more than one processor is assigned to the factorization of each separator supernode, with the processors cooperating. The coarse mode of parallelism degrades geometrically as the level increases; therefore, it is most advantageous to exploit the finer mechanism. Factorization of a separator supernode requires completion of two tasks: elimination of the supernode itself, and the update of the supernode's ancestors.

The update of ancestors is performed after the factorization of the supernode is complete to save interprocessor communication. Therefore, level 1 supernodes (nodes 16-23 in Fig. 4-23) are eliminated before level 2, 3, and 4 supernodes (24-30) are updated, and so on. This procedure is a hybrid of the fan-in and the fan-out methods described by others, including Heath et al. [134]. A complete fan-in procedure would be more desirable because of communication efficiency; however, it cannot be implemented with this algorithm because the exterior matrix is partitioned column-wise among different processors, and both the $L^{(ee)}$ and the $U^{(ee)}$ matrices are needed for updates.

An explanation of this compromise follows. During the elimination of an exterior separator supernode $S_e$, each processor temporarily stores the $L^{(ee)}$ part of the exterior matrix, which is required for updates to the ancestral supernodes, i.e. the $L^{(ee)}$ corresponding to Adj($S_e$). After the elimination of the supernode is completed, this part of $L^{(ee)}$ is used together with the columns of $U^{(ee)}$ held by the processor in Adj($S_e$), to form the updates to the ancestral supernodes.

The processors which share a supernode cooperate in completing the factorization for it. The processor which holds the pivot column searches the rows of the supernode for the largest entry, pivots if necessary, forms the pivot column, and sends it to its partners. All of these processors store temporarily the $L^{(ee)}$ part of the pivot column, which will be used in forming the updates to the ancestors. Once a processor receives the pivot column, or finishes sending it to its partners, it updates the remaining columns in the supernode.
The ancestors are updated when the factorization of the supernode is complete. Each processor forms the update columns using the $L^{(ee)}$ and the $U^{(ee)}$ part of the supernode that it has just eliminated. Then, the processor adds the update column to the appropriate position in the exterior matrix. This last step may require communication if the resulting position of the exterior matrix belongs to a different processor. The entire above procedure is repeated until the factorization of the entire matrix is complete.

4.3.7 Triangular Solutions

Forward and backward eliminations for solution of lower and upper triangular equation sets complete the solution procedure. These eliminations use the $L$ and the $U$ parts of the factored matrices to compute the solution vector $x$ and store it in the same location as the data vector $b$.

Each processor stores the components of the solution vector for its assigned sub-domain, $X_i$, as well as the components for its borders, $\text{Adj}(X_i)$. The components of the solution vector for the exterior separators are divided among the processors using the same method developed for division of the columns of the exterior matrix.

Forward Elimination

First, the interior matrix $L^{(ii)}$ is used for the elimination. The elimination of the interior is completely parallel. Next, the components of $b$ corresponding to the exterior separators, $b^{(e)}$, are updated by the factor $L^{(ei)}$. This step requires communication of the updates, which are introduced by forward elimination of the interior to $b^{(e)}$. The updated components of $b^{(e)}$ correspond to the borders of the interior.

The forward elimination of the exterior is computed one level at a time. The portion of $L^{(ee)}$ associated with each exterior separator supernode $S_e$ is divided into two parts: $L_1^{(ee)}$ updates the supernode’s solution and $L_2^{(ee)}$ updates the ancestors’ solutions; i.e. $L_1^{(ee)}$ corresponds to $S_e$ and $L_2^{(ee)}$ to $\text{Adj}(S_e)$. $L_1^{(ee)}$ is partitioned into groups
of columns, so that each group has the same number of columns as the processors that share the supernode. Thus, if a supernode is shared among four processors, then the corresponding columns in the exterior matrix are partitioned among four processors in a wrap-map fashion. The $L_1^{(ee)}$ part of the matrix is divided into four-column groups. The processor holding the first column rearranges the vector $b^{(r)}$ according to the pivoting that has occurred. These pivot exchanges may require interprocessor communication if the elements to be exchanged belong to two different processors. Then, the first processor forms the next three updates with the second, third, and fourth entries of the first column and communicates these values to the processor holding the second column.

Next, as the first processor continues to form the rest of the update to $b^{(r)}$ using the part of the first column corresponding to $L_1^{(ee)}$, the second processor simultaneously adds the received updates to its portion of $b^{(r)}$, and rearranges the vector based on pivot indices, if necessary. The second processor forms the next two updates by the third and fourth entries of the second column and passes them to the third processor. It then forms the remaining updates to $b^{(r)}$. The third processor adds the two received updates to its components of $b^{(r)}$ and rearranges the vector based on pivot indices. It forms the next update using the fourth entry of the third column and passes it on to the fourth processor. Subsequently, it forms the remaining updates to $b^{(r)}$. The fourth processor adds the received update to its part of $b^{(r)}$, rearranges the vector based on pivot indices, and forms the remaining updates. When finished, the second, third, and fourth processors pass their updates by the remaining part of $L_1^{(re)}$ to the first processor. The first processor adds these entries to its own updates, and the procedure is repeated for the next group of columns. This process continues until forward solution is completed for this supernode. In the above steps, pivoting necessitates updating the entry $b^{(r)}_{\text{pivot}(k)}$ which is to be swapped with the current entry, $b^{(r)}_k$, where $\text{pivot}(k)$ is the pivot index for the $k$th column. This requires all processors to send update messages for the entry $b^{(r)}_{\text{pivot}(k)}$ to the processor which receives that entry.
Therefore, more than two message exchanges are necessary for swapping $b_{\text{prev}(k)}^{(e)}$ with $b_k^{(e)}$.

Finally, the updates of $b^{(e)}$ that correspond to $L_2^{(ee)}$ need to be computed. Each processor forms its contribution in parallel. These contributions are communicated and added to the appropriate places in each local storage of $b^{(e)}$. This completes forward solution of the supernode. The process is repeated for all supernodes until the forward solution is complete.

Pivoting requires a message exchange during forward elimination only if the entries to be exchanged belong to different processors. Furthermore, the pivot entry itself needs to be updated at this step, which requires further communication. All this communication, which includes many short messages, may degrade the performance of the forward elimination algorithm, as some of the results in Chapter 5 show. The following is one way to reduce the number of messages passed in this algorithm. The portion of the $b^{(e)}$ vector which corresponds to an exterior separator is not divided among the processors which share that separator. Instead, each processor keeps a replica of that portion of the vector $b^{(e)}$. This increases the computer storage requirement for the vector $b^{(e)}$ slightly; however, swapping the entries of the $b^{(e)}$ vector do not require communication. Next, at each step during the forward elimination, all of the updates introduced by the current column of the $L_1^{(ee)}$ portion of $L^{(ee)}$ are exchanged among the processors which share the current supernode. This results in longer update messages, but does not increase their total number. Using this strategy, the update of all entries of $b^{(e)}$, corresponding to the current supernode, is complete at each step of elimination; therefore, no messages are required to update the entries of $b^{(e)}$ to be swapped. On the other hand, some parallelism is lost because updates introduced by $L_1^{(ee)}$ to $b^{(e)}$ are calculated sequentially with this new scheme. Overall, if a great deal of pivoting is required, this scheme may be advantageous; otherwise, the current algorithm is faster.
4.3 CFS Algorithm

Figure 4-29: Representation of a supernode in the matrix.

Backward Elimination

The procedure for backward elimination is very similar to the forward elimination except that no pivoting takes place. Backward elimination starts at the last column and goes down to the first column. It is performed first on the exterior matrix. The intermediate solution from this $s.ep$ is communicated so that each processor holds the solution for the borders of its assigned subdomain. Then, backward elimination of the interior is performed to produce the solution vector. At this point, each processor stores the solutions for its own interior subdomain, and for its borders.

Complexity Estimates

Here, a simple relationship is derived for the computation time for $LU$-factorization by CFS of a two-dimensional problem. The derivation follows a logic similar to that presented by Fischer and Patera [96].

The operation count for elimination of a supernode and update of the rest of the
Figure 4-30: Schematic representation of an $n \times n$ square domain divided using nested dissection into $m \times m$ subdomains among $P$ processors.
matrix is
\[ C(n_a, n_b) = \frac{1}{3} \{(n_a + n_b)^3 - n_b^3 - n_a \} \]  \hspace{1cm} (4.27)

where \( n_a \) is the number of vertices in the supernode \( x_i \); namely \( |x_i| \), and \( n_b \) the number of vertices in \( \text{Adj}(x_i) \), namely \( \text{Deg}(x_i) \), as shown in Fig. 4-29. Equation (4.27) follows from adding the number of operations for calculating the pivot columns

\[ (n_b + n_a - 1) + \ldots + (n_b + 1) + n_b \]

and the number of operations for updating the rest of the matrix

\[ (n_b + n_a - 1)^2 + \ldots + (n_b + 1)^2 + n_b^2 \]

Consider a square domain which is decomposed by the nested dissection method into separators \( \{S_e\} \), and subdomains \( \{X_i\} \) which are identical in size, as shown in Fig. 4-30. Each subdomain \( X_i \), which is a leaf in the elimination tree, is assigned to a processor \( p_i \), where \( 1 < i < P \). If each subdomain is \( m \times m \), the operation count for elimination of a subdomain \( X_i \) and update of the separators is

\[ C_0 \leq \frac{m^6}{3} + 4m^5 + 16m^4 - \frac{m^2}{3} \]  \hspace{1cm} (4.28)

which follows from Eq. (4.27), where \( n_a = m^2 \) and \( n_b \leq 4m \). More precisely, \( n_b = 2m \), \( 3m \), or \( 4m \) depending on whether the subdomain falls at the corner, along the edge, or in the middle of the domain; if \( P \gg 1 \), \( n_b = 4m \) for most of the subdomain.

There are two types of separators for the two-dimensional domain of Fig. 4-30. One type of separator results from the vertical dissections; these separators dissect the \( j \)-axis in Fig. 4-30 and are called \( j \)-separators. The other type of separator results from the horizontal dissections; these separators dissect the \( i \)-axis and are called \( i \)-separators. It is easy to verify from Fig. 4-30 that for the \( i \) - and \( j \)-separators
\[ n_a = 2^{l_i-1}m \quad 2^{l_j}m \]
\[ n_b \leq 6n_a \quad 4n_a \]

where \( l_i \) is the level for the \( i \)-separators and \( l_j \) is the level for the \( j \)-separators in the order of elimination. Therefore, the operation counts for factorization of the \( i \)- and \( j \)-separators are

\[ C_i \leq \frac{127}{3} m^3 2^{3(l_i-1)} - \frac{1}{3} (2^{l_i-1}m) \]  \hspace{1cm} (4.29)
\[ C_j \leq \frac{61}{3} m^3 2^{l_j} - \frac{1}{3} 2^{l_j} m \] \hspace{1cm} (4.30)

and the total operation count per processor is

\[ C_p \leq C_0 + \sum_{l_i=l_j=i=1}^{P} \frac{C_i}{P_i} + \frac{C_j}{P_j} \]
\[ = C_0 + \frac{2}{3} m \left( \frac{1}{\sqrt{P}} - 1 \right) + \frac{371}{6} m^3 \left( \sqrt{P} - 1 \right) \] \hspace{1cm} (4.31)

where \( P_i = 2^{2l_i-1} \) is the number of processors which share an \( i \)-separator and \( P_j = 2^{2l_j} \) is the number of processors which share a \( j \)-separator. But \( m \simeq n/\sqrt{P} \), so

\[ C_p \leq \frac{1}{P} \left\{ \frac{1}{3} \frac{n^6}{P^2} + 4 \frac{n^5}{P^{3/2}} + 16 \frac{n^4}{P} - \frac{1}{3} n^2 \right. \]
\[ + \frac{371}{6} \frac{n^3}{P} \left( 1 - \frac{1}{\sqrt{P}} \right) + \frac{2}{3} n \left( 1 - \sqrt{P} \right) \} \] \hspace{1cm} (4.32)

The total operation count is

\[ C_i = P C_p \]
\[ \leq \frac{1}{3} \frac{n^6}{P^2} + 4 \frac{n^5}{P^{3/2}} + 16 \frac{n^4}{P} - \frac{1}{3} n^2 \]
\[ + \frac{371}{6} \frac{n^3}{P} \left( 1 - \frac{1}{\sqrt{P}} \right) + \frac{2}{3} n \left( 1 - \sqrt{P} \right) \] \hspace{1cm} (4.33)
4.3 CFS Algorithm

The matrices for the subdomain of each processor discussed above were treated as dense. The formulas for operation counts should be modified slightly if a processor dissects its subdomain further. In that case, each subdomain corresponds to a subtree of the elimination tree, as shown in the subtree-to-subcube mapping example of Fig. 4-23. Assume that each processor dissects its subdomain \( d_i \) times into \( Q = 2^{d_i} \) sections. Then, the total operation count is given by Eq. (4.33) where \( P \) is replaced by \( PQ \) because the domain is divided into \( PQ \) sections rather than \( P \) sections. Therefore, the operation count per processor is

\[
C_p = C_i / P \\
\leq \frac{1}{P} \left\{ \frac{1}{3} \frac{n^6}{(PQ)^2} + 4 \frac{n^5}{(PQ)^{3/2}} + 16 \frac{n^4}{(PQ)} - \frac{1}{3} n^2 \\
+ \frac{371}{6} n^4 \left( 1 - \frac{1}{\sqrt{(PQ)}} \right) + \frac{2}{3} n \left( 1 - \sqrt{(PQ)} \right) \right\} \tag{4.34}
\]

The execution time is proportional to the operation count given by Eq. (4.34), provided that the time for an operation is constant, and to the volume of the communication, provided that the startup time for the communication is negligible. The volume of communication is estimated below.

Consider a fan-out method to simplify the analysis. The number of the messages sent during factorization is the same as the number of pivot columns, and the volume of the messages is the total volume of the pivot columns. For each supernode \( X \) with \( n_a \) vertices, where \( \text{Deg}(x_i) = n_b \), the message volume is

\[
M(n_a, n_b) = \frac{1}{2} \left\{ (n_a + n_b)^2 - n_b^2 - n_a \right\} \tag{4.35}
\]

which follows from the volume of the pivot columns, which is

\[
(n_b + n_a - 1) + \ldots + (n_b + 1) + n_b
\]
for factorization of the supernode. The volume of the messages generated from the elimination of each subdomain is

\[ M_0 \leq M(m', 4m') + \sum_{l_i = l_j = 1}^{\frac{1}{2} \log_2 Q} (M_i' + M_j') \] (4.36)

where \( m' \approx n/\sqrt{PQ} \), and \( M_i' \) and \( M_j' \) are the volumes of the messages for the \( i' \)- and \( j' \)-separators, respectively, where the prime distinguishes the dissection of the subdomain from the dissection of the overall domain. The volume of the messages that a processor generates is

\[ M_p = M_0 + \sum_{l_i = l_j = 1}^{\frac{1}{2} \log_2 P} \left[ \frac{M_i}{P_i} + \frac{M_j}{P_j} \right] \] (4.37)

where \( M_i \) and \( M_j \) are the volume of the messages for \( i \)- and \( j \)-separators of the overall domain, respectively, i.e. the exterior separators, and \( P_i \equiv 2^{2l_i - 1} \) and \( P_j \equiv 2^{2l_j} \) are the number of processors which share the \( i \)- and \( j \)-separators, respectively. Substituting the expressions for \( M_0 \), \( M_i \), and \( M_j \) gives the leading-order term for the number of messages per processor as

\[ M_p \approx \frac{31}{8} n^2 \log_2 P \] (4.38)

There are \( P \) processors: furthermore, each message is sent to \( P - 1 \) processors, so the leading-order term for the total message volume is

\[ M_t \approx \frac{31}{8} P \log_2 P n^2 \] (4.39)

In the limit for which the dissection is complete, \( P = n^2 \), and Eq. (4.39) reduces to

\[ M_t \approx \frac{31}{4} n^4 \log_2 n \] (4.40)

This result replicates that of George et al. [109].
Because communication is not sequential, the communication time is not proportional to the value of $M_t$ in Eq. (4.39). An estimate for the communication time requires a slightly different analysis. Consider the following two propositions.

**Proposition 4.2** Communications in different subcubes do not interfere and are simultaneous. □

**Proposition 4.3** A broadcast in a subcube is performed in $O(\log_2 p)$ time, where $p$ is the number of processors in the subcube. □

These propositions are realistic for computer architectures of today and have been used by Fischer and Patera [96] for similar analysis. The communication time $T_M$ is proportional to the number of messages generated by each exterior separator multiplied by the broadcast time, $\log_2 P$, i.e.

$$T_M \sim \left( M_0 + \sum_{l_i=l_j=l=1}^{\frac{1}{2}\log_2 P} [M_i + M_j] \right) \log_2 P \quad (4.41)$$

or to the leading-order term

$$T_M \sim \frac{49}{6} n^2 \log_2 P \quad (4.42)$$

The assumption that the communication time is proportional to $\log_2 P$ for each factorization step may seem unrealistic because a global broadcast is not required at each step. However, this assumption gives the correct communication time, at least to leading-order. To see this, consider only the messages for the pivot columns that are exchanged during factorization of the exterior separators, where these pivot columns are exchanged only among the processors within the subcube that cooperate in factoring the exterior separator. This treatment underestimates the time for message traffic by neglecting the messages for updating the ancestral supernodes.

With this new assumption, the message passing time is proportional to the number of pivot messages for a separator multiplied by the logarithm of the number of
processors in the corresponding subcube

\[ T_M \sim \sum_{l_i = l_j = l = 1}^{\frac{1}{2} \log_2 P} \left[ M_i \log_2 P_i + M_j \log_2 P_j \right] \]  

(4.43)

which reduces exactly to Eq. (4.42) to the leading-order term. It also follows that Eq. (4.42) applies to the hybrid fan-in/fan-out communication strategy used by CFS.

The total factorization time is proportional to the communication time, given by Eq. (4.42), and the operation count per processor, given by Eq. (4.34). To the leading-order terms for communication and computation, this time is

\[ T \sim k_c \frac{n^3}{P} + k_m n^2 \log_2 P \]  

(4.44)

when \( P, Q, \) and \( n \) are large, and \( PQ \sim n^2 \). In Eq. (4.44), \( k_c \) and \( k_m \) are constants proportional to the time for a single computation and communication, respectively.

Equation (4.44) is similar to Eq. (4) in Fischer and Patera [96], but the term proportional to \( n^4/P^2 \) is missing here because nested dissection ordering for each subdomain is more efficient than band ordering. When \( P \) is large \( (P \sim n^2) \) the results are identical because in that case \( n^4/P^2 \sim 1 \); that is, the ordering for each subdomain is irrelevant because of the small sizes of the subdomains. The limit of large \( P \) is not desired in practice because communication becomes large in that limit for two reasons. First, the constant \( k_m \) is usually large compared to the constant \( k_c \); for example, for the Intel iPSC/860 machine which is used for the results presented in Chapter 5, \( k_m \) is two orders-of-magnitude larger than \( k_c \). Second, in the limit of \( P \sim n^2 \), the communication volume \( (O(n^2 \log_2 n)) \) dominates the computation volume \( (O(n)) \). Therefore, CFS gives the best results for medium-grained parallel computers with a moderate number of relatively powerful processors. The 32-processor Intel iPSC/860 with 8 Mbytes of memory per processor is used for evaluation of CFS in Chapter 5.
4.4 Summary

This Chapter lays out a robust method for direct solution of linear equation sets which arise in the context of finite and spectral element discretizations of physicochemically complex problems on MIMD parallel computers. The combination of domain decomposition and nested dissection allows efficient incorporation of the sparsity of the matrix, as well as efficient exploitation of parallelism. Also, the algorithm incorporates partial pivoting for accurate solution of the indefinite and asymmetric equation sets that result from discretization of large classes of differential equations.

Graphs and elimination trees play important roles in solution of sparse, linear systems of equations. These notions, which have traditionally been applied to symmetric, positive-definite (Cholesky) factorization, here have been expanded to general, sparse systems through grouping the equations into supernodes and storing both the lower and upper triangular matrices. Incomplete nested dissection is used for domain decomposition and for ordering of the equations in the subdomain of each processor.

CFS algorithm is hierarchical in all of its steps: ordering, symbolic factorization, formulation, numeric factorization, and triangular solutions. Task scheduling is based on the subtree-to-subcube mapping, which is the most communication-efficient mapping developed to date. A hybrid of fan-in/fan-out updating strategy is used for reducing the communication overhead. CFS is best suited for medium-grained MIMD computer architectures \((P \ll N)\) with powerful processors. That is, each processor should be fast and have a large amount of memory so that \(Q \sim P\). The performance of CFS is examined in Chapter 5.
Chapter 5

Results for Direct Solution of Partial Differential Equations in Parallel

This chapter describes the results from implementation of the Cooperative Factorization and Storage (CFS) method presented Chapter 4 on a 32-node Intel iPSC/860 parallel computer. The architecture, performance, and communication speed of the parallel computer are outlined in Section 5.1. The structure, implementation, and portability of the parallel program on this machine are described in Section 5.2. Solution times and speedups from the test problem of Poisson constitute Section 5.3 and are analyzed in detail in that Section for solution of elliptic boundary-value problems using bilinear, biquadratic, and spectral element discretizations. Results are presented for computation rates $R$, computation times $T$, speedups $S \equiv T_1/T_p$, and efficiencies $\varepsilon \equiv S/P$. Also, the number and volume of the messages, which are measures of a parallel algorithm's efficiency, are compared with the recent work of Mu and Rice in Section 5.3. The test problems of lid-driven flow in a cavity and natural convection in an enclosed cavity are presented in Section 5.4, including computation times, speedups, and performance comparisons with a highly efficient Navier-Stokes solver.
based on conjugate gradient iteration implemented on the Intel hypercube [181, 94], and with a serial frontal solver implemented on a Cray X-MP [139, 140]. The scalability of the problem size with the number of processors and at various levels of dissection is demonstrated in Section 5.5. The summary of these calculations is given in Section 5.6.

5.1 The Intel iPSC/860

The test platform is an Intel iPSC/860 parallel computer with 32 processors and 8 MBytes of memory per processor. Details of machine specifications are described in the subsequent sections.

5.1.1 Architecture and Machine Specifications

The Intel iPSC/860 is a MIMD parallel computer. The interconnection topology is hypercube, as shown in Fig. 3-4. The dimension of the cube is five, meaning that there are \(2^5 = 32\) processors, or *compute nodes*, in the machine; the maximum possible dimension is 7 \((2^7 = 128\) processors) for an Intel iPSC/860. It is possible to add to the system up to 127 specialized *I/O nodes* which manage data traffic in parallel to hard disks and other input/output devices such as Ethernet channels. The hypercube can be partitioned into *subcubes*. The size of a subcube can range from a cube of dimension zero, i.e. one node, to a cube of maximum dimension, e.g. five for the machine used here.

Each compute node has an Intel i860 64-bit RISC microprocessor which operates at 40 MHz and has 8 and 16 KByte caches for instructions and data, a floating point adder, and a floating point multiplier. In addition, each compute node has 8 MBytes of memory expandable to 64 MBytes. The operating system and the executable program instructions use approximately 1.5 MBytes of the memory on each node, leaving about 6.5 MBytes for use by the program. The peak speed is
50-60 double-precision millions of floating point operations per second (*MFLOPS*) per node. Unfortunately, a fraction of this speed, which is about a fifth of the peak value, is usually obtained.

There is a front-end computer called the *system resource manager* (SRM) which performs general managing tasks such as I/O, booting the nodes, loading the nodes with executable programs when requested by the user, and compiling programs. The SRM runs the UNIX System V operating system and the compute nodes run the NX operating system which handles message passing, memory management, and process management for the nodes. Basically, the processing elements run user-executable programs and the SRM does all of the other work. The nodes usually work in a dedicated mode, i.e. one user uses them at a time. Typically, a user logs in to SRM; gets a subcube; runs his/her program; and releases the subcube when finished using it.

Communication is managed by the iPSC's Direct-Connect™ internal network which can establish several simultaneous processor-to-processor communications with uniform performance among all nodes. Each communication channel, which is a bi-directional pathway, may deliver up to 5.6 MBytes/second. The peak speed of communication is 2.8 MBytes/sec or 3.5 KWords/seconds for unidirectional message passing which is more common. Thus, the peak communication speed is more than two orders-of-magnitude smaller than the peak computation speed.

A general system configuration is shown in Fig. 5-1, and system specifications are summarized in Table 5.1 for various numbers of compute nodes.

### 5.1.2 Performance of the i860 Chip

**Basic Performance**

The compilers are *if77* and *icc* for the i860 chip. The former is for Fortran and the latter for C routines. The languages are nearly standard Fortran 77 and C with very minor changes. The most significant difference is the addition of manufacturer
Figure 5-1: A general configuration for Intel iPSC/860 [151].
<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of CPUs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Peak performance (GFLOPS)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>double precision</td>
<td>0.48</td>
<td>0.95</td>
<td>1.9</td>
<td>3.8</td>
<td>7.6</td>
</tr>
<tr>
<td>single precision</td>
<td>0.64</td>
<td>1.3</td>
<td>2.6</td>
<td>5.1</td>
<td>10</td>
</tr>
<tr>
<td><strong>Memory capacity (MBytes)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1,024</td>
</tr>
<tr>
<td>maximum</td>
<td>512</td>
<td>1,024</td>
<td>2,048</td>
<td>4,096</td>
<td>8,192</td>
</tr>
<tr>
<td><strong>Storage (GB/tes)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard</td>
<td>0.65</td>
<td>1.3</td>
<td>1.9</td>
<td>4.5</td>
<td>9.1</td>
</tr>
<tr>
<td>maximum</td>
<td>9.1</td>
<td>20</td>
<td>40</td>
<td>82</td>
<td>165</td>
</tr>
<tr>
<td><strong>Internal network (MBytes/second)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bisection band-width</td>
<td>22.4</td>
<td>44.8</td>
<td>89.6</td>
<td>179.2</td>
<td>358.4</td>
</tr>
<tr>
<td><strong>External I/O</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>available channels</td>
<td>7</td>
<td>15</td>
<td>31</td>
<td>63</td>
<td>127</td>
</tr>
<tr>
<td>maximum I/O (MBytes/second)</td>
<td>19.6</td>
<td>42.0</td>
<td>86.8</td>
<td>176.4</td>
<td>355.6</td>
</tr>
<tr>
<td><strong>Physical size (feet)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>1.8</td>
<td>1.8</td>
<td>3.6</td>
<td>3.6</td>
<td>5.3</td>
</tr>
<tr>
<td>height 5.0, depth 2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Operating temperature</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-30°C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Intel iPSC/860 system specifications [151].
Figure 5-2: Performance of the LINPACK benchmark program with the if77 compiler on the i860 chip; the computation speed as a function of the order of the matrix.

subroutines and functions for communication.

The i860 chip performance, as obtained by the LINPACK benchmark test program [77], is shown in Fig. 5-2. The LINPACK benchmark program does standard LU-decomposition (LU), forward elimination (FE), and backward elimination (BE) or back substitution on a general dense matrix, and calculates the computation rate in MFLOPS for solving the system. This program is available from netlib@ornl.gov by sending an e-mail message with the text: send linpackd from benchmark. The program was compiled with full optimization (-O4 option). The two curves correspond to compilations with IEEE specifications for dividing floating point numbers (default) and with an inline divide algorithm for increased performance (-Knoi

\footnote{An index can be obtained from netlib for all available programs and reports by sending an e-mail with the text send index.}
option) [150]. Intel's inline divide algorithm calculates results that are no more than three units different in the last digit from the IEEE results.

The abscissa in Fig. 5-2 is the order of the matrix and the ordinate is the overall average computation rate $R$ in MFLOPS for LU, FE, and BE algorithms. Matrices of the order 9 to 200 are tested. There is no significant difference in speed in the case of Fig. 5-2 when used with or without IEEE division, but IEEE division reduces the performance considerably for special cases which are discussed below. The large variation in from 1.7 to 6.5 MFLOPS is because of vectorization. The i860 chip has pipelining and vectorization capabilities, and the if77 compiler takes advantage of these features. Vectorization time consists of a fixed startup time and a time based on the vector length for a vector operation; see the description in Section 3.1.2. Operations with shorter vectors take longer because the total startup time is larger than it is for operations with longer vectors. For example, vector lengths range anywhere from 1 to $N - 1$ for LU-factorization of a matrix of order $N$. In summary, matrices of higher order $N$ have longer vectors and yield higher overall speeds for LU-factorization.

The large dependence of speed on vector length causes difficulties when trying to fit computation times from numerical experiments to theoretical expressions. In order to better understand the difficulties, consider the above example of LU-decomposition, forward elimination, and back substitution on a dense matrix. The computational complexity is $2N^3/3 + 2N^2 + O(N)$ for this algorithm [77]. This results in a total computation time of

$$t = c \left( \frac{2N^3}{3} + 2N^2 \right)$$

for solution of the system, which is accurate to $O(N)$. The parameter $c$ depends on the time for a floating point operation on the computer. In fact, $c$ is the inverse of the computation rate, i.e. $c = 1/R$. Comparing Eq. (5.1) with Fig. 5-2, one finds that $c$ is not a constant. Fitting the data to find $c$ would be impossible for the above formula. Therefore, one needs to modify Eq. (5.1) to account for vectorization, as described
Figure 5-3: daxpy subroutine from LINPACK.

below, to fit the data.

Basic linear algebra subroutines (BLAS) handle vector operations in the LINPACK benchmark program. Four BLAS subroutines are used: idamax for finding the maximum value of a vector, dscal for scaling a vector with a scalar \( v := sv \),
daxpy for adding a scalar multiple of a vector to another vector \( v_1 := v_1 + sv_2 \), and
ddot for the dot product of two vectors. Almost all of the calculations take place in
BLAS subroutines. The codes for these routines are optimized for vectorization.\(^2\) The
daxpy routine performs by far the largest number of operations, \( (2N + 5)/(2N + 8) \)

\(^2\)To get the Fortran source codes for these routines from netlib@ornl.gov, send an e-mail, for example, with the text send daxpy from linpack.
of total, followed by \texttt{dsca1}, $3/(2N + 8)$ of total. The routine \texttt{damax} performs no operations; it is used for finding the largest entry in the pivot column for row pivoting, and the routine \texttt{ddot} is only used for FE and BE when solving the transpose system $A^T x = b$. Multiplication time dominates addition time, so scalar-vector multiplications use most of the solution time.

Results for computation rates for solving an upper triangular system are presented in Fig. 5-4. Figure 5-4 is the same as Fig. 5-2, except that only the coefficient matrix $A$ is upper triangular. This causes the components of the vectors to be zero and the scalar non-zero ($v = \{0, 0, ..., 0\}; s \neq 0$) in scalar vector multiplies. This test helps study the change in speed of multiplication when one of the operands is zero. There is a slight gain in speed in this case. The speed ranges from 1.8 to 6.6 MFLOPS in
Figure 5-5: Performance of the LINPACK benchmark program with the if77 compiler on the i860 chip with the coefficient matrix equal to identity; computation rate as a function of the order of the matrix.

Fig. 5-4; it ranged from 1.7 to 6.5 MFLOPS in Fig. 5-2. The compiler causes the slight increase in speed in Fig. 5-4 over Fig. 5-2. The regular source code for daxpy does not differentiate between the zero and nonzero elements in the vector; however, daxpy does differentiate between a zero and a non-zero scalar multiplier; see Fig. 5-3 for the daxpy source code. If the scalar multiplier is zero, the subroutine returns without any operations. Therefore, the speed increases nearly linearly when the coefficient matrix is the identity matrix (A = I) as in Fig. 5-5. The speed increases from 2.5 to 80 MFLOPS, as shown in Fig. 5-5. The gain is quite significant in speed over Figs. 5-2 and 5-4. Again, there is essentially no difference between the two curves corresponding to using IEEE specification for division or not.

The complications that vectorization introduced in fitting the solution time data to
theoretical expressions were described above. The fact that multiplications with zero scalar coefficients take a different amount of time introduces yet another complication. The number of zero scalar coefficients is usually significant in the factorization of sparse matrices. The parameter $c$ in Eq. (5.1) depends not only on what the vector lengths are, but also on whether or not the vectors are multiplied by zero or nonzero scalars. A more general formula than Eq. (5.1) should separate the vectors based on their lengths and should separate the scalar multipliers based on whether they are zero or nonzero. Such treatment results in the time for a scalar-vector multiplication of

\[ t_n = \begin{cases} 
  t_0 & \text{if } s = 0 \\
  (t_0 + t_1) + nt_2 & \text{if } s \neq 0 
\end{cases} \quad (5.2) \]

for a vector of length $n$, where $t_0$ is the time it takes for calling a routine, such as daxpy, and returning from that routine; $t_1$ is the startup time for a vector operation; and $t_2$ is the time for scalar-vector multiplication per vector element. Figure 5-6 shows the calls to dscal and daxpy during the direct solution procedure; see Figs. 3-16, 3-17, and 3-18 for complete $LU$, $FE$, and $BE$ algorithms. Since only the lengths of the vector operations are relevant, only they are shown in Fig. 5-6. The vector lengths are $N - k$ in $LU$ and $FE$ for each $k$, and the vector length is $k - 1$ in $BE$ for each $k$. The addition operation accompanies the multiplication in daxpy, and usually addition and multiplication are counted as two operations in calculations of computation speeds in MFLOPS. Therefore, the length of vectors multiplied by two gives the total number of operations. There is no addition in dscal, but this is ignored because the number of operations of dscal is very small compared to that of daxpy. So, there are $2(N - k + 2)$ vector operations for $LU$ and $FE$, and there are two vector operations for $BE$ for each $k$.

To simplify the situation, consider the dense coefficient matrix of Fig. 5-2 where the scalar is never zero (i.e., $s \neq 0$). Therefore, the time is $t_n = (t_0 + t_1) + nt_2$ for a scalar-vector multiplication of a vector with length $n$ resulting in the computation
\[\text{LU:}\]
\[
\begin{align*}
\text{for } k &= 1 \text{ to } N - 1 \\
\text{\ldots} \\
\text{dscal}(N-k, \ldots) \\
\text{\ldots} \\
\text{for } j &= k + 1 \text{ to } N \\
\text{\ldots} \\
\text{daxpy}(N-k, \ldots)
\end{align*}
\]

\[\text{FE:}\]
\[
\begin{align*}
\text{for } k &= 1 \text{ to } N - 1 \\
\text{\ldots} \\
\text{daxpy}(N-k, \ldots)
\end{align*}
\]

\[\text{BE:}\]
\[
\begin{align*}
\text{for } k &= N \text{ to } 1 \\
\text{\ldots} \\
\text{daxpy}(k-1, \ldots)
\end{align*}
\]

Figure 5-6: LU, FE, and BE portions of the LINPACK benchmark program showing the vector operations with vector lengths.
times of

\[
t_k^{LU+FE} = 2(N - k + 2)[(t_0 + t_1) + (N - k)t_2]
\]
\[
t_k^{BE} = 2[(t_0 + t_1) + (k - 1)t_2]
\]

for each \( k \). Summing up the \( k \)'s gives the total computation time as

\[
t = \sum_{k=1}^{N-1} t_k^{LU+FE} + \sum_{k=1}^{N} t_k^{BE}
\]

Performing the summations and adding up the times give

\[
t = \left(5N + N^2 - 4\right)(t_0 + t_1) + \left(-\frac{8}{3}N + \frac{2}{3}N^3 + 2N^2\right)t_2
\]

Note that ignoring \( t_0 + t_1 \) and \( O(N) \) terms reduces Eq. (5.6) to Eq. (5.1). It also is important to note that \( O(N) \) corrections are significant in Eq. (5.6) and cannot be ignored as they can be in Eq. (5.1) because \( N \) is small for many data points in Fig. 5-2. Including \( O(N) \) terms complicates the theoretical time estimate of Eq. (5.6).

Linear regression\(^3\) of Eq. (5.6) to the data of Fig. 5-2 gives

\[
t_0 + t_1 = 0.468 \times 10^{-6}
\]
\[
t_2 = 0.150 \times 10^{-6}
\]

Figure 5-7 shows the actual timing data and the fit with Eq. (5.6) and the above parameters. The perfect fit\(^4\) demonstrates the validity of Eq. (5.6) and the underlying assumptions which lead to it.

Equation (5.6) is derived for the simple situation of solving a dense system on a single processor. Deriving a similar relation for parallel solution of a general sparse

\(^3\)The routine regression from the library stats of Maple V Release 2 was used for fitting the data.
\(^4\)\( \frac{||r||_2}{N} = 0.0003 \) seconds is smaller than the uncertainty in timing which is \( \pm 0.001 \) seconds.
Figure 5-7: The timing data and fit with Eq. (5.6) for the speed data shown in Fig. 5-2.
linear equation set would entail counting the number of zero and nonzero scalar multipliers, the number of the messages, the length of the messages, and the number and the length of the vectors. The derivation, resulting relations, and regression to data would be lengthy and quite complicated. Detailed treatments are beyond the scope of this text for deriving general relations for fitting solution time data. Instead, simplified equations, such as (5.1), are used in Chapter 4 to compare the complexities of different algorithms.

Improving Performance

In CFS, BLAS routines, mainly daxpy, perform most of the calculations; it follows that their further optimization has great potential for large gains in speed. Usually, these routines are written and optimized at the assembly language level to take advantage of the details of the processor architecture to speed up the calculations; the manufacturer provides these hand-coded routines in a collection or library which is compiled and ready to be linked to the user's object code. Kuck and Associates [167] have developed such a library for the i860 chip. Figures 5-8, 5-9, and 5-10 show the performance of the LINPACK benchmark program using the hand-coded BLAS provided by Kuck and Associates.

First, consider Fig. 5-8 for solving the dense system. Kuck BLAS increase the performance significantly. The speed starts at 1.7 MFLOPS, as in Fig. 5-2, for \( N = 9 \), but increases to 9.8 MFLOPS for \( N = 200 \) which is 1.5 times the result for using standard BLAS codes (Fig. 5-2). Using the IEEE option retards the speed slightly.

Next, consider Fig. 5-9 for solving the system with an upper triangular coefficient matrix \( A \). For this matrix, the elements of the vectors are zero and the scalars nonzero in scalar-vector multiplies. It is immediately observable that marked differences appear in Fig. 5-9 for the results compiled with and without IEEE. The IEEE curve behaves qualitatively differently from the previous figures; furthermore, it has quite a low performance. This curve starts at 1.8 MFLOPS for \( N = 9 \) and increases to 3.6
Figure 5-8: Performance of the LINPACK benchmark program with the Kuck BLAS on the i860 chip; the computation speed as a function of the order of the matrix.
Figure 5-9: Performance of the LINPACK benchmark program with the Kuck BLAS on the i860 chip for an upper triangular coefficient matrix; the computation speed as a function of the order of the matrix.
Figure 5-10: Performance of the LINPACK benchmark program with the Kuck BLAS on the i860 chip for an identity coefficient matrix; the computation speed as a function of the order of the matrix.

MFLOPS, but then the performance suddenly drops and goes down to 1.6 MFLOPS for $N = 36$. The performance picks up slowly from that point, but it never reaches the values of the previous figures. The maximum speed is 4.3 MFLOPS for $N = 200$ for the IEEE curve of Fig. 5-9. The curve for the option without the IEEE arithmetic behaves similarly to the previous figures qualitatively, but the performance is slightly better than the dense matrix case of Fig. 5-8. The speed starts at 2.3 MFLOPS for $N = 9$ and reaches 10.2 MFLOPS for $N = 200$. The strange behavior of the IEEE curve is caused by the details of Kuck hand-coded BLAS and has nothing to do with the i860 chip.

Finally, consider Fig. 5-10 where the LINPACK test is reported for a calculation using the identity matrix. The scalar multiplier is zero for these results, and the
BLAS routines return without any operation. Therefore, the time for a scalar-vector multiply is equal to the time for calling the routine and returning from it. The speed increases linearly with increasing problem size from 3 to 105 MFLOPS for the curve without the IEEE option, and from 2.2 to 100 MFLOPS for the curve with the IEEE option. Compare this to 2.5 to 80 MFLOPS in the previous case (Fig. 5-5) for the curves both with and without IEEE options.

In summary, computation speed of a single processor is a strong function of the vector length. Compiling standard BLAS routines with if77 on a single i860 processor gives a maximum performance of about 6.5 MFLOPS. Using the Kuck hand-coded BLAS increases this performance to about 10 MFLOPS. Zero elements in the vector do not increase the performance significantly, but zero scalar multipliers do. Compiling without the IEEE option does not change the performance when regular BLAS are used; however, it does increase the performance slightly when the Kuck library is used. Using the IEEE arithmetic significantly deteriorates the performance when the Kuck library is used, the elements of the vector are zero, and the scalar multiplier is nonzero. The strong dependence of speed on vector length and the operands' value (zero or nonzero) requires complicated timing estimates to fit the measured execution times.

5.2 Implementation on an Intel iPSC/860 Hypercube

5.2.1 Program Structure

Programs are compiled on the SRM, and are loaded onto the nodes of the subcube which the user has allocated for execution. If the algorithm does not require preprocessing and general management, node programs alone suffice. These are the programs which run on the processing elements of the cube and are loaded onto the nodes by
the operating system environment of the SRM. These node programs start executing once they are loaded onto the cube and run until they finish or otherwise exit, e.g. by user interruption.

It is possible to have a manager program which runs on the SRM. This host program performs the preprocessing, loads the node programs, sends initial data to the nodes, manages node programs during the run by sending and receiving appropriate messages, and receives the final solution from the nodes.

Usually, all the nodes run a single program on multiple or different sets of data. Some refer to this mode as the single program multiple data (SPMD) mode. In other words, replicas of the same program run on the nodes and operate independently on different data. The node programs follow different paths based on each node’s data, and the processors communicate and synchronize at various stages of the program to exchange data. A flow diagram of the algorithm discussed in Chapter 4 as implemented on the Intel hypercube is shown in Fig. 5-11.

The host program reads the parameters of the run from a file, dissects the domain into subdomains, and sends various parameters and dissection information to the nodes.

The node program does initialization and Newton iterations. Each iteration consists of formulation, LU-factorization, triangular solutions, and tests for convergence; the node program ends by sending the solution back to the host. Each node initializes its variables based on the data it receives from the host. That is, it sets up its assigned subdomain and further dissects it based on the information it receives from the host about its assigned subdomain. The nodes then formulate the problem on their assigned subdomains, do LU-factorization on the interior equations, update the exterior equations, and do other tasks as shown in Fig. 5-11 until the solution of the linear equation set is complete. The nodes need to communicate to update data during various stages of the calculation; the communications are shown with horizontal arrows. After the linear system is solved, the solution is tested for convergence.
Figure 5-11: Flow diagram of the domain decomposition with nested dissection algorithm as implemented on the Intel hypercube.
Different paths are followed based on the results of the convergence test. If Newton’s method has converged, the solutions are sent back to host; otherwise, another Newton’s iteration takes place which starts at the top of the diagram with formulation. The process repeats as described above until the Newton iteration converges or until a predefined maximum number of Newton iterations is reached.

5.2.2 Portability

Portability of the parallel source code is very important, because parallel computers continuously evolve and change. Manufacturers introduce new machines and old machines become obsolete. If a program is strongly dependent on a computer, its life will depend on the life of that computer. Generality of an efficient parallel program is especially important because a large amount of time and effort is invested in developing the program.

Minimizing machine dependence has been a significant goal in developing the parallel program in this thesis. Both the host and the node programs are written mainly in Fortran. Message passing and node synchronization are handled through calls to routines that are machine dependent and are provided by the manufacturer for the particular purpose. All manufacturers provide message passing and synchronization routines for their MIMD parallel computers. However, currently there are neither standards for the kinds of routines provided, nor for their format, so porting a parallel program to a different platform requires some changes in the message passing sections of the program.

Intel parallel MIMD computers provide three modes for message passing:

1. **Synchronous** message passing blocks the process until sending or receiving is complete.

2. **Asynchronous** sending takes place in the background, and asynchronous receiving posts a request for receiving a message in the background. The programmer
needs to check a flag to make sure the message passing is complete.

3. **Interrupt** sending and receiving is similar to asynchronous, except that an interrupt handler routine, which is written in C by the user, executes upon completion of the sending or receiving of the message. Interrupt receives are very advantageous because the program need not block and wait for the incoming messages, nor does the program need to post receive requests continuously or check whether the messages have arrived. Usually, the routine for handling interrupts posts another request for a receive after completing whatever action is necessary for the message that has just arrived.

The send and receive routines used here are synchronous except for those forming the exterior matrix after the interior matrix is eliminated and for those updating the ancestral supernodes after a supernode is eliminated; interrupt receive routines are used in those instances so that a processor avoids wasting time waiting for incoming messages. As an update message comes in, it interrupts the program and appropriate action is taken. Critical parts of the program that should not be interrupted are masked [148], i.e. they are prevented from being interrupted. No attempts have been made here to measure the amount of time saved by incorporating these interrupt receive instructions.

### 5.3 Tests with Poisson’s Problem

Results are reported in this section for solution of Poisson’s equation on a square domain discretized with bilinear, biquadratic, and spectral elements. The discrete matrix problems that result from Poisson’s problem are symmetric and positive-definite; however, the program does not exploit these properties and treats the system as a general asymmetric, indefinite equation set. Row pivoting is allowed within the subdomain or the separator which is being factored. Pivoting takes place for the basis functions of higher order than bilinear because the largest entry is not necessarily the
diagonal even though the matrix is symmetric and positive-definite, and pivoting is theoretically not required.

5.3.1 Message Traffic

The time it takes for a message exchange consists of a startup time plus the sending time, which depends on the length of the message. This relationship between time and message length is similar to that for vector operations. However, the message passing time depends on other factors besides startup time and message length, such as the path through which the message is routed; for example, the time for passing a message depends on whether a physical connection exists between the processor which originates the message and the processor which receives it. If a direct connection exists, the message takes less time to travel. It is desirable to minimize the time for exchanging messages because message passing time reduces the speedup. This is because message passing is slow, and has no effect on data operation. The simplest model for communication time is the linear one

\[ t_m = t_s + t_c m \] (5.9)

where \( m \) is the message length, \( t_s \) is the startup time, and \( t_c \) is the time for communicating a message of length one. The total time for sending and receiving messages \( t_M \) is the sum over all of the messages \( m \)

\[ t_M = \sum_m (t_s + t_c m) \] (5.10)

Therefore, the total message time \( t_M \) is reduced by sending fewer messages and reducing the total message volume. Thus, the total number of messages and the total volume of messages can be used as measures of a parallel algorithm’s efficiency.

The number and volume of the messages generated by the \( LU \)-factorization algorithm developed here are compared in Figs. 5-12 and 5-13 with measures for the
recent algorithm of Mu and Rice [201]; in these calculations the domain is divided among 16 processors. The discretization scheme does not change the data for these figures because the exterior separator matrices are full no matter what type of discretization is used for the interior matrices. The only important fact is that these data are for the domain divided into 16 subdomains and assigned to 16 processors. Mu and Rice propose a grid-based subtree-to-subcube assignment strategy for solving boundary value problems on hypercubes [201]. Briefly, they suggest dividing matrix columns for the exterior separators among the processors based on the proximity of the processors. For example, the upper part of separator 24 in Fig. 4-22 is closest to processors 0 and 4, and the lower part is closest to processors 8 and 12. Therefore, the upper portion of separator 24 is divided between processors 0 and 4 and the lower
Figure 5-13: The volume of messages for the domain divided among 16 processors as a function of the number of unknowns in one direction compared to the recent work of Mu and Rice [201].
portion between processors 8 and 12. The standard method would be to wrap-map all of separator 24 among processors 0, 4, 8, and 12.

Because elimination of the subdomains of processors 0 and 4 generates updates only for the upper portion of separator 24, the communication is reduced if that portion of separator 24 is divided between processors 0 and 4 only. The improvement from the standard assignment strategy by the new subtree-to-subcube assignment strategy is evident in Figs. 5-12 and 5-13 as published by Mu and Rice [201]. The algorithm in this thesis improves significantly both the number of messages and the total volume over the values reported by Mu and Rice for both their standard and new assignment strategies. This improvement in efficiency is the result, as described in Chapter 4, of both the fan-in update of the exterior matrix after the interior is factored, and of the partial fan-in update of the ancestral exterior supernodes. Although the grid-based subtree-to-subcube assignment strategy of Mu and Rice [201] has not yet been implemented in CFS, the grid-based strategy should reduce the number and total volume of messages even further.

5.3.2 Bilinear Finite Elements

This section contains preliminary results for evaluating the performance of the parallel program. Poisson’s problem is discretized with bilinear quadrilateral finite elements and Galerkin’s method. This discretization is similar to the centered finite differences which uses the star finite difference stencil; however, it is equivalent to it only for right triangular elements because bilinear right triangular elements have both the same stencil shape and the same interpolating order as the centered finite difference approximation.

First, numerical formulation is discussed. Then, some background is presented on details of computations and the method of speedup calculation. Lastly, results are shown for the speedup associated with the \(LU\)-decomposition and for the times for \(LU\), FE, and BE.
Formulation

Poisson's problem defined on a square two-dimensional domain is

$$\nabla^2 u = f(x, y)$$  \hspace{1cm} (5.11)

where $0 < x < 1$, $0 < y < 1$, and $f$ is some known function of position. The boundary condition for this problem is chosen as

$$u = 0 \text{ on } \partial D$$  \hspace{1cm} (5.12)

where $\partial D$ is the boundary of the domain. The domain $D$ is divided into $N_e$ finite elements such that $D = \bigcup_{e=1}^{N_e} D_e$. The unknown $u$ is approximated by bilinear Lagrangian interpolants $\{\Psi^j\}$, which are given by Eqs. (5.16)-(5.19), as

$$u \simeq \hat{u} = \sum_{j=1}^{N} u_j \Psi^j(x, y)$$  \hspace{1cm} (5.13)

where $\hat{u}$ is the approximate solution, $\{u_j\}$ are the unknown coefficients, and $N$ is the total number of nodes in the finite element mesh. The global position vector $(x, y)$ is mapped isoparametrically onto a local coordinate system $(\xi, \eta)$ using the same bilinear interpolating functions as

$$x = \sum_{j'=1}^{4} x_{j'}^c \Psi^{j'}(\xi, \eta)$$  \hspace{1cm} (5.14)

$$y = \sum_{j'=1}^{4} y_{j'}^c \Psi^{j'}(\xi, \eta)$$  \hspace{1cm} (5.15)

where $\{x_{j'}^c\}$ and $\{y_{j'}^c\}$ are the coordinates of the nodes in the finite element mesh and $\{\Psi^{j'}\}$ are

$$\Psi^1(\xi, \eta) = \frac{(1-\xi)(1-\eta)}{4}$$  \hspace{1cm} (5.16)
5.3 Tests with Poisson’s Problem

Figure 5-14: Mapping an element from the global coordinates to local coordinates shown graphically.

\[
\begin{align*}
\Psi^2(\xi, \eta) &= \frac{(1 + \xi)(1 - \eta)}{4} \\
\Psi^3(\xi, \eta) &= \frac{(1 + \xi)(1 + \eta)}{4} \\
\Psi^4(\xi, \eta) &= \frac{(1 - \xi)(1 + \eta)}{4}
\end{align*}
\]

in the element and zero outside the element, i.e. they have local support. This mapping is shown graphically in Fig. 5-14.
In general, Lagrangian interpolants of any order \( q \) are given as
\[
\chi^i(\xi) = \frac{\prod_{j=0,j\neq i}^{q+1}(\xi - \xi_j)}{\prod_{j=0,j\neq i}^{q+1}(\xi_i - \xi_j)}
\] (5.20)
in one dimension, where the function \( \chi^i \) has the value of one at the node \( \xi = \xi_i \) and has the value of zero at any other node \( \xi = \xi_{j\neq i} \); see Prenter [221]. Higher dimensional Lagrangian interpolants are constructed as the tensor products of \( \chi^i \); for example, the two-dimensional interpolants \( \chi^{mn} \) are
\[
\chi^{mn}(\xi, \eta) = \chi^m(\xi)\chi^n(\eta)
\] (5.21)
Usually, only one index is used to identify the higher dimensional interpolants as in Eqs. (5.16)-(5.19) where the indices \( m \) and \( n \in \{1, 2\} \) are combined into one index \( j' \in \{1, 2, ..., 4\} \).

The requirement that the inner product of the Poisson's equation be orthogonal to some weighting functions results in the weak form as
\[
\int_D \Psi^i(\nabla^2 u - f) \, dA = 0
\] (5.22)
where the weighting functions \( \{\Psi^i\} \) are the same as the basis functions for the Galerkin formulation; these functions are zero on the boundary of the domain because of the imposed essential boundary conditions. Integrating the above equation by parts gives
\[
\int_D (\nabla \Psi^i \cdot \nabla u + \Psi^i f) \, dA = 0
\] (5.23)
Substituting the approximate form of \( u \) in the weak form results in a linear set of equations
\[
\sum_{j=1}^{N} \int_D \nabla \Psi^i \cdot \nabla \Psi^j \, dA \, u_j = -\int_D \Psi^i f \, dA
\] (5.24)
which needs to be solved for the unknown coefficients \( \{u_j\} \). Comparing Eq. (5.24)
5.3 Tests with Poisson's Problem

with Eq. (3.12), one identifies the elements of the coefficient matrix $a_{ij}$ and the right side vector $b_i$ as

$$a_{ij} = \int_D \nabla \Psi^i \cdot \nabla \Psi^j \, dA$$

$$b_i = -\int_D \Psi^i f \, dA$$

which need to be formed before the solution of the linear system. The above integrals over the whole domain $D$ are decomposed into the sum of the integrals over all elemental subdomains $D_e$ as

$$\int_D g \, dA = \sum_{e=1}^{N_e} \int_{D_e} g \, dA$$

where $g$ stands for an integrand. This elemental decomposition makes the integral evaluations convenient because of the local support of the basis functions. Only the basis functions for the element $e$ need to be considered when evaluating the integral $\int_{D_e} g \, dA$ because the interpolating functions of all other elements are zero outside those elements.

The elemental integrals are transformed into local coordinates $(\xi, \eta)$ from the global coordinates $(x, y)$ to simplify integration. The relationship between the local and global coordinates is

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} \frac{dx}{d\xi} & \frac{dx}{d\eta} \\ \frac{dy}{d\xi} & \frac{dy}{d\eta} \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix}$$

where the the elemental Jacobian $J_e$ transforms the local coordinates into the global coordinates. Using Eq. (5.28) to transform the vector form of the area element, $d\tilde{A} = d\tilde{x} \times d\tilde{y}$, from the global coordinates into the local coordinates gives $| d\tilde{A} | = \det(J_e) d\xi d\eta$. The elemental integrals are approximated by Legendre Gaussian quadra-
ture [124, 122, 247] as
\[
\int_{D_e} g \, dA \simeq \sum_{k=1}^{N_g} j_k w_k \tag{5.29}
\]
where \( g_k \) is the integrand evaluated at the quadrature point \( k \), \( w_k \) is the quadrature weight, and \( N_g = 4 \) is the total number of quadrature points.

**Background**

Because the problems which are tested on 4 and 16 processors are too large to fit onto one processor, the speedups cannot be calculated by dividing the time for one processor by the time for 4 or 16 processors. A scheme has been developed instead to calculate the computation speed on one processor which is used in speedup calculations.

The Kuck library with IEEE division is used for the results in this section. Recall that the speed behavior is not smooth for multiplying elements involving zero from either the vector or the scalar, as shown in Figs. 5-9 and 5-10. Consequently, the operations are counted into two groups: those involving nonzero operands and those involving zero operands. This division is a crude way of accounting for the complicated behavior for operations involving zero operands, as shown in Figs. 5-9 and 5-10. The relations for calculating a single processor’s speed in MFLOPS which is used for calculating speedup data are derived below.

Assume that there are two computation rates possible: \( M_0 \) and \( M_1 \); the former for operations involving a zero operand, and the latter for operations involving nonzero operands. Then the following relations exist:

\[
M_1 = \frac{f_{1i}}{t_{1i}} \tag{5.30}
\]
\[
M_0 = \frac{f_{0i}}{t_{0i}} \tag{5.31}
\]
\[
t_i = t_{1i} + t_{0i} \tag{5.32}
\]
where \( f \) is the number of floating point operations, \( t \) is time, the indices 0 or 1 refer to the operations involving zero or nonzero operands, respectively, and the index \( i = \{1, \ldots, n\} \) is the number of data points, i.e. runs. It is possible to count the number of zero and nonzero operations separately by adding a subroutine for that purpose, but it unfortunately is only possible to find the total time since adding statements or subroutines which separate the time measurements would add to the overall computation time, and hence defeat the purpose. Therefore, the knowns in the above relations are \( f_{0i} \), \( f_{1i} \), and \( t_i \). The total time is written in terms of the operations and the computation rates for a calculation as

\[
\frac{f_{0i}}{M_0} + \frac{f_{1i}}{M_1} = t_i \tag{5.33}
\]

Defining \( N \equiv 1/M \) gives

\[
N_0 f_{0i} + N_1 f_{1i} = t_i \tag{5.34}
\]

Fitting Eq. (5.34) to data yields the zero and nonzero rates, \( M_0 \) and \( M_1 \), or their inverses \( N_0 \) and \( N_1 \). For example, consider the least squares method. The residual to be minimized is

\[
r = \sum_i \varepsilon_i^2 \tag{5.35}
\]

\[
= \sum_{i=1}^{n} (N_0 f_{0i} + N_1 f_{1i} - t_i)^2
\]

Minimizing \( r \) requires setting the derivative with respect to the unknowns \( N_0 \) and \( N_1 \) to zero as

\[
\frac{\partial r}{\partial N_0} = \sum_i 2(N_0 f_{0i} + N_1 f_{1i} - t_i) f_{0i} = 0 \tag{5.36}
\]

\[
\frac{\partial r}{\partial N_1} = \sum_i 2(N_0 f_{0i} + N_1 f_{1i} - t_i) f_{1i} = 0 \tag{5.37}
\]
Performing the summations on data results in two equations and two unknowns as

\[ N_0 \sum_i f_{0i}^2 + N_1 \sum_i f_{1i}f_{0i} = \sum_i t_if_{0i} \quad (5.38) \]
\[ N_0 \sum_i f_{0i}f_{1i} + N_1 \sum_i f_{1i}^2 = \sum_i t_if_{1i} \quad (5.39) \]

which are solved for \( N_0 \) and \( N_1 \) to yield the zero and nonzero computation rates.

The IEEE curves shown in Figs. 5-9 and 5-10 show that the above is a very crude way of fitting the data, but precise formulas which depend on vector length are quite complicated. In order to minimize vector-length effects, Eqs. (5.38) and (5.39) are used to find \( M_0 \) and \( M_1 \) on one processor for runs which have the same problem per processor as the runs on multiple processors. The least square fit to data gives:

\[ M_0 = 23 \text{ MFLOPS} \quad (5.40) \]
\[ M_1 = 4.9 \text{ MFLOPS} \quad (5.41) \]

These values are used for calculating the speedups in Fig. 5-15 which is discussed below.

**Computation Times and Speedups**

The speedups and absolute computation times for 4 and 16 processors are shown in Figs. 5-15 and 5-16 as a function of the number of unknowns. The domain is dissected a total of 16 times in all cases. For \( P = 4 \), the domain is dissected twice first, and the four subdomains are assigned to the processors. Each processor dissects its assigned subdomain 8 times into interior subdomains and separators. For \( P = 16 \), the domain is dissected 4 times first, and the 16 subdomains assigned to the processors, subsequently. Each processor dissects its assigned subdomain 4 times. More processors can fit larger problems; therefore, the curves for \( P = 16 \) extend to much larger numbers of unknowns. Since the level of dissection does not change,
Figure 5-15: Speedup for $LU$-decomposition of Poisson's problem discretized with bilinear finite elements as a function of the number of unknowns. The domain is divided 16 times in all cases.
Figure 5-16: Computation times for solution of Poisson's problem discretized with bilinear finite elements as a function of the number of unknowns. The domain is divided 16 times in all cases.
the size of supernodes increases with the increasing number of unknowns. The sizes of interior subdomains range from \(3 \times 3\) for the smallest problem with about 4,000 unknowns to \(11 \times 11\) for the largest problem with approximately 36,500 unknowns. For \(P = 4\), the speedup almost reaches the ideal value of 4; and for \(P = 16\), the speedup is about 85% of the ideal value of 16 for the largest problem.

Computation times are shown in seconds as an indication of the absolute speed of the program because only a high value of speedup and a high absolute speed are indicative of a useful algorithm. The time for \(LU\)-factorization is less than 7 seconds for a problem with about 36,500 unknowns. Furthermore, times for triangular solutions are more than an order-of-magnitude less than the factorization times. Although forward elimination and back substitution times are small, they are not as small as operation counts would indicate relative to \(LU\)-factorization because these algorithms are inherently less parallel than \(LU\)-factorization. Detailed discussions of speedups are presented in Sections 5.3.3 and 5.3.4 for higher order discretizations.

### 5.3.3 Biquadratic Finite Elements

This Section contains the results for the timings, absolute speeds in MFLOPS, and speedups for the solution of Poisson’s problem discretized with biquadratic finite elements. The presentation of the results is preceded by a description of the problem formulation and a background on the calculation of the results.

**Formulation**

The problem formulation with Galerkin’s finite element method is identical to the one described in Section 5.3.2 except for the choice of the basis and weighting functions. Biquadratic Lagrangian interpolants \(\Phi^i\) are used instead, which are

\[
\Phi^1(\xi, \eta) = \frac{\xi(\xi - 1)\eta(\eta - 1)}{4}
\]  

(5.42)
\[ \Phi^2(\xi, \eta) = \frac{(1 - \xi^2)\eta(\eta - 1)}{2} \] (5.43)
\[ \Phi^3(\xi, \eta) = \frac{\xi(\xi + 1)\eta(\eta - 1)}{4} \] (5.44)
\[ \Phi^4(\xi, \eta) = \frac{\xi(\xi - 1)(1 - \eta^2)}{2} \] (5.45)
\[ \Phi^5(\xi, \eta) = (1 - \xi^2)(1 - \eta^2) \] (5.46)
\[ \Phi^6(\xi, \eta) = \frac{\xi(\xi + 1)(1 - \eta^2)}{2} \] (5.47)
\[ \Phi^7(\xi, \eta) = \frac{\xi(\xi - 1)\eta(\eta + 1)}{4} \] (5.48)
\[ \Phi^8(\xi, \eta) = \frac{(1 - \xi^2)\eta(\eta + 1)}{2} \] (5.49)
\[ \Phi^9(\xi, \eta) = \frac{\xi(\xi + 1)\eta(\eta + 1)}{4} \] (5.50)

in the element where \(-1 \leq \xi \leq 1\) and \(-1 \leq \eta \leq 1\) and zero outside the element.

**Background**

The program is compiled without the IEEE option; therefore, the speed for zero vector multiplication behaves similarly to the speed for nonzero vector multiplication; see the curve in Fig. 5-9 without the IEEE option. The only difference in behavior happens when zero multipliers are used. Multiplications involving zero multipliers are fast; see Fig. 5-10. The speeds in MFLOPS are calculated using only scalar-vector operations involving nonzero scalar multipliers. The times are assumed to be negligible for zero scalars multiplying vectors, and these operations are omitted in the calculations of the speed. This treatment somewhat lowers the absolute rate. However, the purpose is not only to show high absolute rates.

Although high absolute rates are desirable, they are not enough indication for a useful algorithm. Consider an example. One is to travel from point A to point B and has the choices of using a bicycle on a shorter path or a car on a winding road. The bicycle travels at 10 mph and the car at 40 mph on the average. However, the path for the bicycle is 5 times shorter. The car takes 25% longer to reach point B
even though the bicycle goes much more slowly. This trivial example is analogous to
choosing nested dissection ordering over band ordering. Nested dissection ordering
produces shorter vector lengths for operations, but requires many fewer operations.
Now, suppose the bicycle can take a few short-cuts which take a negligible amount
of time but reduce the length of the path considerably. If the length of these short-
cuts is small and is ignored, the average speed of the bicycle is still nearly 10 mph,
maybe a little less because the time which it took to go through the short-cuts was
not subtracted, but still the total time for the travel is considerably less. These
short-cuts correspond to avoiding operations involving zero scalars in scalar-vector
multiplications. Now, suppose that one could use a motorcycle and travel the same
path as the bicycle but at 20 mph. The increase in speed directly lowers the time;
this is analogous to using Kuck hand-coded BLAS to increase the computation rate.
Note that one need not only go faster but also go via a shorter path and use whatever
short-cuts are available for arriving at the destination (solution) in a shorter time.

In this Section, the speedups are calculated differently from in the previous and the
following Sections. This is because different methods of speedup calculations can give
different answers, and a better judgment of the efficiency of the algorithm is made by
comparing speedups calculated with different methods. Here, the computation rate
$R$ versus the number of unknowns $N$ are fit to an exponential of the form

$$R = k_1 e^{-\frac{k_2}{N}}$$  \hspace{1cm} (5.51)

on one processor to find $k_1$ and $k_2$ constants.\(^5\) Subsequently, the computation rates
on several processors are divided by the value calculated by the above fit to arrive at
the speedup. The above procedure is applied to calculate speedups for $LU$, FE, and
BE.

The interior subdomains are dissected 6 times for all the results in this section.

\(^5\)The function \texttt{fmins} was used in Matlab version 4.1. This function uses a Simplex search method
for minimization of a nonlinear function of several variables.
This means that the interior subdomains are subdivided into $8 \times 8 = 64$ smaller subdomains in all cases. For example, the host process divides the domain 2 times for $P = 4$; then each processor divides its assigned subdomain 6 times into 64 interior subdomains. The constant interior dissection is in contrast to the results in Section 5.3.2 for discretization with bilinear elements where the total number of subdomains was constant. The smallest number of elements a processor has is $8 \times 8$ for an $8 \times 8$ dissection, i.e. one element per subdomain. The results, shown in this Section's figures, range from $10 \times 10$ elements per processor, to whatever the memory allows, e.g. $33 \times 32$ for $P = 16$. The results for large numbers of processors lack data for small numbers of unknowns because of the $8 \times 8$ lower bound of the number of elements per processor. Fine interior dissection is used because of its better use of sparsity, and the fact that it permits solution of larger problems. The goal here is using the memory efficiently to fit large problems on each processor, and then increasing the number of processors as required for solution of ever larger problems. The reader interested in finding the cross-over points for smaller numbers of unknowns, where using a larger number of processors becomes less efficient, is referred to Section 5.3.4 where the results are presented from spectral element discretizations of Poisson's problem.

**Absolute Speeds**

First, consider the computation rates attainable on one processor. Fig. 5-17 contains the results for the computation rate for $LU$-decomposition with the exponential fit of the form of Eq. (5.51). The parameters of Eq. (5.51) are

\[
\begin{align*}
  k_1 &= 6.32 \\
  k_2 &= 743
\end{align*}
\]

for fitting the data in Fig. 5-17. The maximum computation rate of about 6 MFLOPS is low even though Kuck library is used. Recall that the maximum performance is
10 MFLOPS for the LINPACK benchmark program with Kuck library, as shown in Fig. 5-8. The reasons for the low speed are the overhead paid for integer addressing calculations and shorter vector lengths by domain decomposition algorithm, as opposed to the LINPACK benchmark program. Figures 5-18 and 5-19 contain the computation rates for FE and BE on one processor with the exponential curve fits of the form of Eq. (5.51). The parameters of Eq. (5.51) are

$$k_1 = 5.52 \quad (5.54)$$
$$k_2 = 812 \quad (5.55)$$

for the FE, as fit from data in Fig. 5-18, and are

$$k_1 = 5.33 \quad (5.56)$$
$$k_2 = 767 \quad (5.57)$$

for the BE, as fit from data in Fig. 5-19.

The computation rate for LU in MFLOPS is shown in Fig. 5-20 for several processors. More processors can fit larger numbers of unknowns, so the curves for larger processors extend to higher values of $N$. The minimum problem size per processor limits the starting points of the curves in Fig. 5-20 as discussed above. The trend is increasing speed for all curves. The reason is the increasing vector lengths with the increasing number of unknowns. The computation rate increases more slowly with the increasing number of unknowns for each curve in Fig. 5-20, although the curves do not show saturation. The maximum rate is not particularly high (120 MFLOPS) even for the largest problem ($N = 130,000$) on 32 processors. However, the maximum factorization time is low (15.5 seconds) for 130,000 unknowns on 32 processors, as shown in Fig. 5-21. Maximum computation rates are shown in Table 5.2 corresponding to the last points on each curve of Fig. 5-20. The maximum rate nearly doubles as the number of processors is doubled. The maximum efficiencies drop slightly as the
Figure 5-17: The computational speed for $LU$ in MFLOPS versus the number of equations for Poisson's problem discretized with biquadratic finite elements on one processor.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$R$</th>
<th>$\frac{R_P}{R_{P-1}}$</th>
<th>$\frac{R_P}{R_1}$</th>
<th>$\varepsilon \equiv \frac{R_P}{PR_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.7</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11.1</td>
<td>1.9</td>
<td>1.9</td>
<td>0.95</td>
</tr>
<tr>
<td>4</td>
<td>21.1</td>
<td>1.9</td>
<td>3.7</td>
<td>0.92</td>
</tr>
<tr>
<td>8</td>
<td>37.6</td>
<td>1.8</td>
<td>6.6</td>
<td>0.82</td>
</tr>
<tr>
<td>16</td>
<td>64.5</td>
<td>1.7</td>
<td>11.3</td>
<td>0.71</td>
</tr>
<tr>
<td>32</td>
<td>122.1</td>
<td>1.9</td>
<td>21.4</td>
<td>0.67</td>
</tr>
</tbody>
</table>

Table 5.2: Maximum computation rate $R$ for different numbers of processors from Fig. 5-20. The corresponding efficiencies $\varepsilon$ also are shown.
Figure 5-18: Computational speed for FE in MFLOPS versus the number of equations for Poisson’s problem discretized with biquadratic finite elements on one processor.
Figure 5-19: Computational speed for BE in MFLOPS versus the number of equations for Poisson's problem discretized with biquadratic finite elements on one processor.
Figure 5-20: Computational speed for $LU$-decomposition in MFLOPS versus the number of equations for Poisson’s problem discretized with biquadratic finite elements on more than one processor.
Figure 5-21: Computation time for LU-decomposition as a function of the number of equations for Poisson's problem discretized with biquadratic finite elements.
number of processors increases. The increase in computation rate starts at a factor of 1.9, goes down to 1.7, and back up to 1.9 again each time the number of processors is doubled; computation rates increase by a factor of $1.84 \pm 0.09$ on the average. Of course, the ideal increase is a factor of 2; this means that the efficiency drops on the average by a factor of $1.84/2 = 0.92$ when doubling the number of processors. Thus, the maximum efficiency is estimated by the empirical relation

$$\varepsilon = 0.92^d$$  \hspace{1cm} (5.58)

where $d$ is the dimension of the subcube or $d = \log_2 P$. Clearly, indefinitely increasing the number of processors is not efficient. The decrease in efficiency is inevitable because as the number of processors increases, the exterior separations comprise a larger portion of the problem, and thus, the communication associated with manipulating the exterior matrix increases. Accordingly, the size of the communication network grows as the number of processors increases, and communication becomes less efficient overall. Setting a lower acceptable efficiency bound of 0.5 gives a $d$ of between 8 and 9. In other words, the maximum number of processors is $2^8 = 256 < P < 2^9 = 512$. These numbers of processors correspond to efficiencies of 0.51 and 0.47, to speedups of 131 and 242, and to computation rates of 747 and 1,380 MFLOPS, respectively. The scope of applicability for these numbers is limited to the current test problem on the iPSC/860. A computer with larger memory per processor would be able to do larger problems with a fixed number of processors and improve the computation efficiency. Likewise, a computer with a smaller ratio of communication to computation time improves the efficiency. The important point is that increasing the number of processors decreases the efficiency, and there is a limit to the maximum number of processors that should be used.

Next, consider the computation times and speeds for FE, as shown in Figs. 5-22 and 5-23. The computation rate increases nearly linearly with the number of unknowns. The linear increase signifies shorter vector lengths in the FE vector op-
Figure 5-22: Computation time for FE as a function of the number of equations for Poisson's problem discretized with biquadratic finite elements.
Figure 5-23: Computational rate for FE as a function of the number of equations for Poisson's problem discretized with biquadratic finite elements.
erations. Only the computation rates for up to 8 processors show slightly downward curvatures which signify slight saturation of vector speeds. The maximum computation rate is 26 MFLOPS, which is quite low compared to that for \( LU \)-decomposition. One reason for this relatively low performance is that FE is inherently less parallel than \( LU \). Another reason is that pivot exchanges take place during FE which require message passing and further reduce the performance for FE. Therefore, the FE results for \( P = 32 \) show a slight degradation compared to the results for \( P = 16 \). It is possible to improve the FE algorithm at a slight cost in memory usage, as described in Chapter 4; however, the computation times for FE remain quite low compared to \( LU \)-factorization. The largest computation time for FE is 0.75 seconds, which is only five percent of the factorization time. Only if a modified Newton’s method is used, such as in the transient nonlinear calculations of Chapter 2, where several triangular solutions are required per \( LU \)-factorization, might the time for FE become a significant fraction of the total solution time.

Back substitution does not involve pivoting, so the results show higher computation rates and faster speeds because of the absence of the extra communication time associated with pivoting. Figures 5-24 and 5-25 show the times and computation rates for BE. The back substitution for \( P = 32 \) performs much better than the corresponding forward elimination, and the computation rate for \( P = 32 \) overtakes the rate for \( P = 16 \). Furthermore, the computation rates are much larger. The maximum computation rate is 39 MFLOPS for BE as opposed to 26 for FE, and the BE time for the largest problem is 0.46 seconds, which is nearly half of the corresponding FE time (0.75 seconds).

**Speedups**

The speedups are shown in Figs. 5-26, 5-27, and 5-28 for \( LU \)-factorization, FE, and BE, respectively. The speedups are calculated by fitting the exponential relation of Eq. (5.51) to the computation rate for \( P = 1 \), and then dividing the rate for \( P \)
Figure 5-24: Computation time for BE as a function of the number of equations for Poisson's problem discretized with biquadratic finite elements.
Figure 5-25: Computation rates for BE as a function of the number of equations for Poisson's problem discretized with biquadratic finite elements.
Figure 5-26: The speedup for LU-factorization as a function of the number of unknowns for Poisson's problem discretized with biquadratic finite elements.
Figure 5-27: The speedup for FE as a function of the number of unknowns for Poisson's problem discretized with biquadratic finite elements.
Figure 5-28: The speedup for BE as a function of the number of unknowns for Poisson's problem discretized with biquadratic finite elements.
<table>
<thead>
<tr>
<th>$P$</th>
<th>$S$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.88</td>
<td>0.94</td>
</tr>
<tr>
<td>4</td>
<td>3.47</td>
<td>0.87</td>
</tr>
<tr>
<td>8</td>
<td>6.07</td>
<td>0.76</td>
</tr>
<tr>
<td>16</td>
<td>10.30</td>
<td>0.64</td>
</tr>
<tr>
<td>32</td>
<td>19.40</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Table 5.3: The highest values of speedups $S$ for $LU$ and the corresponding efficiencies $\varepsilon$ for Poisson's problem discretized with biquadratic finite elements.

<table>
<thead>
<tr>
<th>$P$</th>
<th>FE</th>
<th>BE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.78</td>
<td>1.84</td>
</tr>
<tr>
<td>4</td>
<td>2.88</td>
<td>3.10</td>
</tr>
<tr>
<td>8</td>
<td>4.46</td>
<td>5.01</td>
</tr>
<tr>
<td>16</td>
<td>4.72</td>
<td>6.28</td>
</tr>
<tr>
<td>32</td>
<td>4.51</td>
<td>7.39</td>
</tr>
</tbody>
</table>

Table 5.4: The highest values of speedups for FE and BE for Poisson's problem discretized with biquadratic finite elements.

processors by the rate predicted for one.

The speedups for $LU$ increase with the increasing problem size for each $P$ but do not saturate, even for the largest problems. The largest values for $LU$ speedups and the corresponding efficiencies for different values of $P$ are summarized in Table 5.3. The maximum speedups for $LU$ are a reasonable fraction of the ideal value, and the efficiencies are reasonable fractions of one.

The highest speedups are shown in Table 5.4 for FE and BE. The speedups are higher than half of the ideal value for $P \leq 8$ and lower than half of the ideal value for $P > 8$. The speedups for BE are larger, especially for $P > 8$, than the speedups for FE; the largest speedup for BS is more than 1.5 times that for FE for $P = 32$. 
The low performance of FE is caused by the message passing during pivoting, which slows down the FE process.

5.3.4 Spectral Elements

The results in this Section are for Poisson's problem discretized with spectral elements as developed by Patera and colleagues [216, 226, 181, 136]. Results are presented for computation times, computation rates, and speedups for $LU$-factorization, FE, and BE subsequent to a description of the spectral element discretization and a background on the computations and calculations of the results.

Formulation

The formulation is very similar to the one for bilinear finite elements which is described in detail in Section 5.3.2. The differences are highlighted below.

The order of interpolating polynomials is no longer constant. Higher order interpolants, rather than mesh refinement, are used for higher accuracies. These higher order interpolants are calculated by the general formula for calculating Lagrangian interpolants of arbitrary order, as described by Eq. (5.21). To simplify formulation, the nodal points for the elements are chosen such that they coincide with the quadrature points. There is no need to compute or store the values of the basis functions at the quadrature points since they can only be either zero or one. The trivial values of the basis functions at the quadrature points reduce the computational complexity of the formulation of the Jacobian and the right side vector. Regular Gauss points do not include the end points of the domain and cannot be used; therefore, Gauss-Lobatto quadrature points are used instead, which include the end points of the domain [124, 122, 247]. The derivatives of the basis functions are calculated and stored at the quadrature points. These derivatives are calculated for one-dimensional basis functions, and tensor products are used for forming the two-dimensional counterparts. The global $(x, y)$ to local $(\xi, \eta)$ coordinate transformation is performed
subparametrically with bilinear interpolants as in Section 5.3.2. Since the elements are rectangular, the subparametric mapping does not cause any loss of accuracy.

Increasing the polynomial order for higher accuracies is often called $p$-refinement, and refining the mesh is called $h$-refinement. Usually, the term “spectral elements” refers to combining the two methods ($hp$-refinement) for best results. The power of $hp$-refinement combined with adjusting the position of elements is discussed in a recent article by Oden [211].

The method of $p$-refinement exhibits exponential convergence for problems whose solutions are expandable in Taylor Series, wherein adding higher order terms in polynomial interpolants gives better approximations; otherwise, $h$-refinement gives better approximations. The resulting exponential drop in error is demonstrated for Poisson’s problem discretized with spectral elements, as compared to the errors by bilinear and biquadratic finite element discretizations in Fig. 5-29. The spectral elements reach machine precision with two hundred unknowns, whereas the bilinear and biquadratic finite elements are several orders-of-magnitude behind in accuracy even with one hundred thousand unknowns. In general, when solutions with higher accuracies are required, spectral elements have a smaller computational complexity for computing a solution of the same order-of-accuracy. The low-order bilinear or biquadratic finite elements are advantageous for computing solutions with low accuracy, which are not shown in Fig. 5-29. Another advantage of spectral elements is their ability to resolve boundary layers [125, 200, 35].

The advantages of spectral element discretizations are demonstrated below as combined with CFS. Spectral elements maximize connectivity within subdomains without increasing the separator widths; therefore, solutions of higher accuracy can be obtained with $p$-refinement without increasing the communication associated with patching up the solutions at the subdomain boundaries. As a result, when high accuracies are desired, spectral elements yield better computation rates and speedups, and lower times for solutions of similar accuracies than do finite elements.
Figure 5-29: Convergence of solution of Poisson's problem discretized with bilinear finite elements, biquadratic finite elements, and $8 \times 8$ spectral elements as a function of the number of unknowns $N$ where $e \equiv \| \hat{u} - u \|_\infty$. 
Background

The program was compiled with IEEE option, and the speedups were calculated by dividing the time for solving the problem on \( P \) processors by the time for solving the problem on one processor if the problem was small enough to fit on one processor; otherwise, a quadratic curve was fit to the results for the largest three numbers of unknowns and this fit was used to extend the speedup calculations to higher values of \( P \). For example, the last two points on the \( P = 2 \) speedup curve were calculated by extrapolation and used for the \( P > 2 \) speedup calculation as

\[
S_P = \frac{T_2}{T_P} S_2, \quad P > 2
\]  

(5.59)

where \( S \) stands for the speedup and \( T \) for computation time. The last three points for \( P = 4 \) speedup curve were calculated by extrapolation and used for \( P > 4 \) speedup calculation in a similar way

\[
S_P = \frac{T_4}{T_P} S_4, \quad P > 4
\]  

(5.60)

The domain is divided into \( 8 \times 8 = 64 \) spectral elements for all the results in this section. The basis function interpolation order \( q \) ranges from \( q = 2 \) to the largest allowable based on the amount of memory per processor. Larger numbers of processors accommodate higher order basis functions because each processor holds fewer spectral elements. The maximum basis function orders are \( q = 8 \) for \( P = 1 \) in which the processor holds all 64 spectral elements, and \( q = 22 \) for \( P = 32 \) in which each processor holds only two spectral elements.

Absolute Speeds

Figures 5-30 and 5-31 show the computation rates for \( LU \) as a function of the number of unknowns for various numbers of processors. The difference between these two figures is that Fig. 5-30 includes all operations whether or not they involve zero operands, and Fig. 5-31 includes only the operations which involve nonzero vector
Figure 5-30: Computation rate for $LU$ as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. All operations are included, i.e. those involving zero and nonzero operands.
Figure 5-31: Computation rate for $LU$-factorization as a function of the number of unknowns for Poisson’s problem discretized with $8 \times 8$ spectral elements. Only operations involving nonzero operands are included.
elements and scalar multipliers. Recall that for IEEE arithmetic the operations involving zero vector elements do not behave smoothly and are slow (Fig. 5-9), and those involving zero scalar multipliers are fast (Fig. 5-10). The actual computation rate is in between the values shown by Figs. 5-30 and 5-31. The maximum number of unknowns is much lower in this Section than in Section 5.3.3: 31,000 versus 130,000 for the biquadratic discretization. This is because spectral elements produce much denser matrices which require more memory. Despite this drawback, spectral element approximations are much more accurate for the same number of unknowns, as shown in Fig. 5-29.

The computation rates for $LU$ are invariably much higher for the spectral element approximations than for biquadratic approximations. The maximum rates are nearly double compared to the biquadratic case; compare Figs. 5-31 and 5-20. The reasons for high computation rates are:

1. The large ratio of interior subdomain computation to exterior separator computation; hence a smaller ratio of communication to computation.

2. Longer vectors in vector operations because of fewer interior subdomains for a larger number of processors.

3. Smaller overhead by integer addressing because of fewer subdomains for a larger numbers of processors.

Maximum computation rates for Fig. 5-31 are tabulated in Table 5.5. The results in Table 5.5 cannot be extrapolated indefinitely as they were in Table 5.2 in Section 5.3.3, because the total number of elements is fixed in this case at 64, and the maximum number of processors which can be used for solving this problem is 64. The rates increase by a factor of $2.26 \pm 0.18$ on average when the number of processors is doubled; an unusual event since the ideal increase is a factor of 2. The reason is that the larger vector lengths and the decrease in integer addressing per processor saves more time than is lost in additional message passing. The increasing vector lengths
cause the vector speed to approach saturation as the number of processors increases. Furthermore, as \( P \) increases, each processor holds fewer subdomains and separators because the total dissection is constant at \( 8 \times 8 \); fewer subdomains and separators require fewer integer addresses, and the associated overhead for integer addressing is smaller. Therefore, for the largest problem that fits in memory, the average computation speed increases as the number of processors increases; this appears as superlinear improvements in processor utilization in Table 5.5.

The computation speed per processor is shown in Fig. 5-32 as a function of the number of unknowns for different numbers of processors. Figure 5-32 is identical to Fig. 5-31, except that the computation speed is divided by \( P \) to show saturation of speed for the different numbers of processors. Because the curve for \( P = 1 \) nearly coincides with the curve for \( P = 2 \) in this figure, it is replotted in the inset. Fig. 5-32 clearly indicates that the computation speed shows stronger signs of saturation as the number of processors increases. The speed for \( P = 1 \) does not show saturation; therefore, using the largest speed for \( P = 1 \) as a basis for speedup calculation results in high speedups. This is an important consideration when speedup curves in which the unsaturated maximum speed on one processor has been used as the basis for speedup calculations are evaluated. This comparison causes a larger value of the

<table>
<thead>
<tr>
<th>( P )</th>
<th>( R )</th>
<th>( \frac{R_P}{R_{P-1}} )</th>
<th>( \frac{R_P}{R_1} )</th>
<th>( \frac{R_P}{P \cdot R_1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.41</td>
<td>-</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>11.0</td>
<td>2.5</td>
<td>2.5</td>
<td>1.2</td>
</tr>
<tr>
<td>4</td>
<td>26.2</td>
<td>2.4</td>
<td>5.9</td>
<td>1.5</td>
</tr>
<tr>
<td>8</td>
<td>54.8</td>
<td>2.1</td>
<td>12.4</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>121</td>
<td>2.2</td>
<td>27.4</td>
<td>1.7</td>
</tr>
<tr>
<td>32</td>
<td>254</td>
<td>2.1</td>
<td>57.6</td>
<td>1.8</td>
</tr>
</tbody>
</table>

Table 5.5: Maximum possible computation rate \( R \) in MFLOPS for different numbers of processors from Fig. 5-31.
Figure 5-32: Computation rate per processor for $LU$-factorization as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements. Nonzero operations are included, i.e. those involving nonzero operands. The curve for $P = 1$ nearly coincides with the curve for $P = 2$ and is replotted in the inset for clarity.
Figure 5-33: Computation times for LU-factorization for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns for various numbers of processors.

speedup than the value reached by extrapolating the $P = 1$ speed to larger numbers of unknowns.

The times for LU-factorization are very low, as shown in Fig. 5-33. The maximum computation time is 19.3 seconds for LU-factorization for 31,000 unknowns and 32 processors.

The computation rates for FE are shown in Figs. 5-34 and 5-35; the former figure includes all of the operations, whereas the latter includes only the operations involving nonzero operands. The trend is similar in these two figures; only the values of the rates are different. The actual rate lies in between that which is indicated in these two figures. The computation rates do not show any indications of reaching saturation for FE except for $P < 8$ where the curves show some downward bend. Pivoting causes
Figure 5-34: Computation rate for FE for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes all operations involving zero or nonzero operands.
Figure 5-35: Computation rate for FE for Poisson’s problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes only operations involving nonzero operands.
the speed to be lower for $P = 32$ than for $P = 16$, as was the case in Section 5.3.3 for biquadratic discretization. The computation times for FE are shown in Fig. 5-36. The curves are not smooth in Fig. 5-36 because of the effect of pivoting. The pivot search is for the largest entry in the separator or subdomain submatrix, and the pattern of the largest entries changes with the number of unknowns. Therefore, the pattern of pivot exchanges results in irregular patterns in FE times. The absolute times are low for FE. The maximum time is for the largest problem on 32 processors. This time is less than 0.49 seconds, as compared to 19.2 seconds for $LU$-factorization. In other words, the computation time for FE is only 2.5 percent of the computation time for $LU$.

The computation speeds for BE are shown in Figs. 5-37 and 5-38. Figure 5-37 includes all operations, whereas Fig. 5-38 includes only the operations involving
Figure 5-37: Computation speeds for BE for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes all operations involving zero or nonzero operands.
5.3 Tests with Poisson’s Problem

Figure 5-38: Computation speeds for BE for Poisson’s problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns. Operation count includes only operations involving nonzero operands.
Figure 5-39: Computation time for BE as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements.

The difference is only in the values; the trends are similar in the two cases. The curve for $P = 32$ overtakes the curve for $P = 16$ for BE because BE does not involve pivot exchanges, hence the communication overhead caused by pivot exchanges is not present. The computation rate increases nearly linearly for $P > 8$ curves, but for $P \leq 8$ the curves show a slight downward bend indicating saturation for $P = 2$ and $P = 4$. The absolute computation times are lower for BE than for FE, as shown in Fig. 5-39. The maximum computation time for BE is 0.31 seconds compared to 0.49 for FE for the largest problem on 32 processors. The curves are closer in Fig. 5-39 for various values of $P$ than they are in Fig. 5-36 for computation times of FE; this behavior is an indication that the algorithm better utilizes the processors for back substitution than it does for forward elimination. Again, message passing needed for pivot exchanges in the FE algorithm causes this inefficiency.
Figure 5-40: Speedups for \( LU \)-factorization for Poisson's problem discretized with 8 \( \times \) 8 spectral elements as a function of the number of unknowns.

Speedups

The speedups presented in this section are calculated from computation times when possible as \( S = T_1 / T_P \), or by extrapolating speedups for smaller numbers of processors as described above. The speedups for \( LU \) are shown in Fig. 5-40. The maximum speedups are nearly ideal in Fig. 5-40; i.e. the speedup for \( P \) processors nearly reaches the value of \( P \). The high speedups confirm the high computation rates shown in Figs. 5-30 and 5-31. Table 5.6 summarizes the maximum \( LU \) speedups and efficiencies for a range of processor numbers. It also shows that the performance does not degrade at all with the increasing \( P \). The maximum efficiencies start from 0.98 for \( P = 2 \), fall to 0.89 for \( P = 8 \), and rise again to 0.92 for \( P = 32 \). Long vector lengths, fewer integer addressings, and less interior sparsity work in favor of the algorithm
for larger numbers of processors and make up for the time lost in communication. The net effect is nearly eliminating the performance degradation by communication. Figure 5-41 is an enlargement of Fig. 5-40 for small values of $N$ and clearly shows the cross-over points at which using a larger number of processors may actually degrade the performance. The fact that cross-overs occur at very low $N$ ($N < 2500$) attests to the efficiency of the parallel $LU$ algorithm. The speedups are always higher than two, so there is always a gain by choosing more than two processors for this problem.

The efficiency of $LU$-decomposition is demonstrated in Fig. 5-42. The efficiencies approach the optimum value with the increasing number of unknowns. There is a small rise in efficiency for calculations with all processors for small problems, i.e. at small $N$. In fact, the curve for $P = 2$ shows an efficiency of 1.25 at $N \approx 300$! This rise is attributed to the effect of the cache, which is the limited amount of high speed memory on each processor. Small problems fit into the cache and use this high speed memory exclusively. There is little or no need for accessing the slower global memory.

The speedups for FE are shown in Fig. 5-43. The speedups are excellent up to $P = 16$ where the processor utilization is $10/16 \approx 0.62$; however, the performance for $P = 32$ is degraded because of pivot exchanges. The speedups for BE are better in general, as shown in Fig. 5-44, and reach 15 for $P = 32$. The speedups for FE saturate for $P = 2$, show signs of saturation for $P = 4$, and do not saturate at all.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$S$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.96</td>
<td>0.98</td>
</tr>
<tr>
<td>4</td>
<td>3.72</td>
<td>0.93</td>
</tr>
<tr>
<td>8</td>
<td>7.13</td>
<td>0.89</td>
</tr>
<tr>
<td>16</td>
<td>14.5</td>
<td>0.91</td>
</tr>
<tr>
<td>32</td>
<td>29.3</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Table 5.6: The highest values of speedups for $LU$ and the corresponding efficiencies for Poisson's problem discretized with $8 \times 8$ spectral elements.
Figure 5-41: Speedups for LU-factorization for Poisson's problem discretized with $8 \times 8$ spectral elements as a function of the number of unknowns for small values of $N$ to show the cross-over points, where using more processors is clearly less efficient.
Figure 5-42: Efficiency for \( LU \)-decomposition as a function of the number of unknowns for Poisson's problem discretized with \( 8 \times 8 \) spectral elements.
Figure 5-43: Speedups for FE as a function of the number of unknowns for Poisson's problem discretized with $8 \times 8$ spectral elements.
Figure 5-44: Speedups for BE as a function of the number of unknowns for Poisson’s problem discretized with $8 \times 8$ spectral elements.
5.4 Multivariable PDE’s

<table>
<thead>
<tr>
<th>$P$</th>
<th>FE</th>
<th>BE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.87</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>3.55</td>
<td>3.38</td>
</tr>
<tr>
<td>8</td>
<td>6.41</td>
<td>6.68</td>
</tr>
<tr>
<td>16</td>
<td>10.0</td>
<td>12.3</td>
</tr>
<tr>
<td>32</td>
<td>9.57</td>
<td>15.2</td>
</tr>
</tbody>
</table>

Table 5.7: The highest values of the speedups for FE and BE for Poisson’s problem discretized with $8 \times 8$ spectral elements.

for $P \geq 8$. The speedups for BE saturate for $P < 8$, show signs of saturation for $P = 8$, and do not saturate for $P > 8$. The cross-over points, below which using a larger number of processors actually degrades the performance, occur at much larger problem sizes for FE and BE compared to those for $LU$; this is because the FE and BE algorithms are less parallel. Table 5.7 summarizes the maximum speedups possible for various numbers of processors. The speedups for BE are slightly lower for $P < 8$, but overtake the speedups for FE for $P \geq 8$, wherein message passing associated with pivoting becomes more important.

5.4 Multivariable Sets of Differential Equations

The results described above demonstrate the excellent performance of the parallel nested dissection algorithm, especially when it is combined with spectral element discretizations, for the solution of a single elliptic differential equation; in this case, Poisson's equation. The results in this Section demonstrate that this algorithm generalizes easily to multiple differential equations with multiple field variables, and to the application of Newton’s method for the solution of nonlinear problems; furthermore, the results show the superior performance of the algorithm based on finite element discretizations, Newton’s method for solving nonlinear equations and direct solution
of linear systems. The types of problems tested describe physical situations. The first test concerns natural convection in a cavity. Fig. 5-45 is the schematic representation for this problem. The second test concerns flow in a cavity with a moving wall. This problem is better known as the *lid-driven cavity* and is shown schematically in Fig. 5-75. The flow causes asymmetry which increases with the intensity of convection in these problems.

The problem of natural convection in a cavity has been studied extensively for low Prandtl number fluids. Interest in fluid motion and transport processes has been increasing in the past two decades; in particular, special attention has been devoted to oscillatory behavior of fluids with low Prandtl numbers, because of the crucial impact of such flow oscillations on the quality of the crystals grown, such as semi-conductors or metallic alloys, for advanced technology applications [230]. Such oscillations have been shown to occur in enclosures as well as in open cavities; also, in real crystal growth systems as well as in model experiments without any growth. After Hurle et al. [146] demonstrated experimentally the onset of oscillatory instabilities for liquid gallium in a parallelepiped container, several other investigators studied theoretically the onset of instabilities; and others computed the system behavior, as described by Roux et al. [232]. However, significant discrepancies existed in the results reported by different authors even for two-dimensional models. As a result, it was decided in 1987 to propose a set of benchmark problems concerning a rectangular cavity with an aspect ratio \( A = \text{length/height} \) of 4. Two types of containers were to be tested, one with a rigid upper surface and the other with a stress-free upper surface, corresponding to closed and open containers, respectively. Two values of the Prandtl number were considered: \( \text{Pr} = 0 \) and \( \text{Pr} = 0.015 \). The advantages of \( \text{Pr} = 0 \) is that the temperature field decouples from the flow and the governing equations reduce to Navier-Stokes equations with a source term; linear stability for this system was studied by Hart [132] and Roux et al. [232]. The value of \( \text{Pr} = 0.015 \) was chosen based on the previous linear stability results. The benchmark proposed several values of the Grashof number
(Gr- defined below), based on the bifurcation results of Winters [270]. Results were presented for these benchmark problems for Grashof numbers slightly above and slightly below the critical value where the steady convection loses its instability to an oscillatory mode. This threshold occurs at velocity and temperature differences small enough that the flow is laminar and quasi-incompressible; therefore, the Oberbeck-Boussinesq approximation is a reasonable model for the system. These results were presented in the GAMM-Workshop in Marseille, France in 1988, and were published in the proceedings [230]. Various formulations and numerical methods were used. The numerical results again exhibited some discrepancies which were attributed either to mesh size effects or to a limited time integration interval.

The benchmark problem of natural convection in an enclosed cavity is used here to test the performance of CFS. The results, which are presented in Section 5.4.1, agree with the values reported by other investigators. Furthermore, computation speeds, speedups, and timings are reported to evaluate the performance of the parallel algorithm. Finally, a comparison of the computation times for obtaining the solution is made with a highly-efficient parallel iterative Navier-Stokes solver, called Nekton [97].

The problem of lid-driven flow in a cavity has a rich history which goes back several decades; it is a simple example for fluid flow with closed streamlines which has a nontrivial solution. A moving wall drives a recirculating flow in a cavity which is filled with fluid. There are singularities in the domain where the moving and stationary walls meet. The structure of the flow and the nature of the singularities are discussed below.

As an example, see Pan and Acrivos [215], who studied the system both numerically for the Stokes flow and experimentally for a range of Reynolds numbers (20-4000). The work prior to Pan and Acrivos' is summarized in a review paper by Burggraf [34]. The reasons for studying this flow have changed over the years from studying the structure of the flow to testing new numerical techniques [242, 208, 182, 211].
Figure 5-45: Diagram of the model problem of two-dimensional natural convection in an enclosed cavity.

In the current context, the lid-driven cavity flow problem is used to test the speed of CFS, and to compare it with a highly efficient algorithm using the frontal technique on a serial vector supercomputer. The results for this problem are presented in Section 5.4.2.

5.4.1 Natural Convection in a Cavity

The problem is shown schematically in Fig. 5-45. The system is two-dimensional and rectangular. The left wall is heated and the right wall is cooled. In the schematic representation, gravity points downward causing the lateral temperature gradient to drive a flow for all imposed temperature differences. The importance of buoyant forces compared to viscous forces is measured by the Grashof number, which is used as a measure of the overall imposed temperature gradient for a system of fixed geometry and thermophysical properties.

Natural convection is steady and laminar for sufficiently low Grashof numbers; however, as Gr increases, the steady solution loses its stability to an oscillatory mode at some critical Gr which depends on the geometry, boundary conditions, and the Prandtl number. The critical Gr numbers for natural convection in a cavity with an aspect ratio of 4 are summarized in Table 5.8, as predicted by Winters [270].
Table 5.8: Critical Grashof number $\text{Gr}_c$ for the onset of oscillatory convection for natural convection in a cavity with $A = 4$ [270].

<table>
<thead>
<tr>
<th></th>
<th>stress-free upper surface</th>
<th>rigid upper surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Pr}$</td>
<td>$5.5 \times 10^4$</td>
<td>$1.0 \times 10^5$</td>
</tr>
<tr>
<td>$0.015$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.1 \times 10^5$</td>
</tr>
</tbody>
</table>

Figure 5-46: Streamlines and isotherms for the steady solution and real and imaginary parts of the critical eigenvector at the first Hopf bifurcation point, for $\text{Pr} = 0.015$, $A = 4$, and a rigid upper surface [270].
Figure 5-47: Oscillatory convection shown as instantaneous streamlines eight times during one cycle, for a value of the Grashof number close to threshold, an aspect ratio of 4, a Prandtl number of 0.015, and a rigid upper surface [270].
Table 5.8 shows that the rigid upper surface has a stabilizing effect on the steady-state solutions; i.e. oscillatory instabilities occur at higher Grashof numbers in an enclosed cavity than in an open cavity. The streamlines and the temperature field for the steady-state solution are shown in Fig. 5-46, as are the eigenvectors at the first Hopf bifurcation point for natural convection in an enclosed cavity with an aspect ratio of 4 [270]. The steady-state flow field consists of a single cell which is symmetric about the center of the cavity; the flow is strong because of the high Grashof number of about $10^5$. The temperature field deviates slightly from the linear profile because of the low Prandtl number of 0.015; the largest deviation occurs at the center of the cavity where the contours clearly show the effects of the counterclockwise flow. The eigenvectors show a multicellular structure for both flow and temperature fields.

Figure 5-47 shows eight snap-shots of the flow field during an oscillation period for a bifurcated solution slightly above the onset of the oscillatory instability for flow in an enclosed cavity [270]. The flow field changes from one to three, and back to one cell during an oscillation period. The three-cell structure which appears during an oscillation period is consistent with the multicellular structure of the eigenvectors predicted by the linear stability results.

An open cavity, which has stress-free boundary condition at the upper surface, shows a slightly different behavior, as shown in Figs. 5-48 and 5-49. The solutions are not symmetric about the cavity center because the boundary conditions are no longer symmetric about the center. The steady-state flow field consists of two cells, one of which is weaker. The eigenvectors for the onset of oscillatory instability show multicellular structure as in the closed cavity case [270].

Eight solutions are shown in Fig. 5-49 at regular intervals during an oscillation for a solution on the bifurcating branch from the first Hopf point for flow in an open cavity [270]. The solution changes character during an oscillation from a single cell to multicell and back to a single cell; this is similar to the behavior exhibited by the closed cavity, as shown in Fig. 5-47; however, the solution is no longer symmetric
Figure 5-48: Streamlines and isotherms for the steady solution and real and imaginary parts of the critical eigenvector at the first Hopf bifurcation point, for Pr = 0.015, A = 4, and a stress-free upper surface [270].

about the center of the cavity for the stress-free upper surface.

Equations and Boundary Conditions

The steady-state flow is described by the incompressible Navier-Stokes equations with Boussinesq approximation for density:

\[ \nabla \cdot \mathbf{v} = 0 \]  

\[ \rho \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla P + \mu \nabla^2 \mathbf{v} + \rho g (T - T_0) \]

where \( \mathbf{v} = (u, v) \) is the velocity vector, \( P \) is the pressure field, \( T \) is the temperature, \( \rho \) is the density, \( \mu \) is the viscosity, and \( g \) is the acceleration of the gravity. The energy equation is

\[ \mathbf{v} \cdot \nabla T = \alpha \nabla^2 T \]
5.4 Multivariable PDE's

Figure 5-49: Oscillatory convection shown as instantaneous streamlines eight times during one cycle, for a value of the Grashof number close to threshold, an aspect ratio of 4, a Prandtl number of 0.015, and a shear-free upper surface [270].


<table>
<thead>
<tr>
<th>variable</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>( (x, y) )</td>
</tr>
<tr>
<td>velocity</td>
<td>( (u, v) )</td>
</tr>
<tr>
<td>pressure</td>
<td>( P )</td>
</tr>
<tr>
<td>temperature</td>
<td>( T )</td>
</tr>
</tbody>
</table>

\[
\text{scale} = \frac{H}{\nu \sqrt{Gr}}
\]

Table 5.9: The variables and the corresponding scales used for putting the variables into dimensionless form.

\[
\begin{align*}
(x', y') & \equiv \left( \frac{x}{H}, \frac{y}{H} \right) \\
(u', v') & \equiv \left( \frac{H}{\nu \sqrt{Gr}} u, \frac{H}{\nu \sqrt{Gr}} v \right) \\
P' & \equiv \frac{H^2}{\rho \nu^2 \sqrt{Gr}} P \\
T' & \equiv A \frac{T_h - T_c}{T_h - T_c}
\end{align*}
\]

Table 5.10: The definition of dimensionless variables as produced by using the scales which are summarized in Table 5.9.

where \( \alpha \) is the thermal diffusivity, which is \( \alpha = \rho / k C_p \) where \( k \) is the thermal conductivity, and \( C_p \) is the heat capacity. The field equations (5.61)-(5.63) are put in dimensionless form with the characteristic scales which are summarized in Table 5.9. The scales in Table 5.9 for the velocity and pressure are appropriate for flows with intense convection and for large Grashof numbers where the dominant balance in the bulk of the liquid involves inertia and buoyancy [230].

Several dimensionless groups arise in the final problem statement. The aspect ratio is \( A \equiv L/H \) and scales the length to the height of the cavity. The Grashof number is \( \text{Gr} \equiv \beta g \Delta T H^4 / L \nu^2 \), where \( \beta \) is the coefficient of thermal expansion, \( g \) is the magnitude of the acceleration of gravity, \( \nu \) is the kinematic viscosity, and \( \Delta T \equiv T_h - T_c \). The dimensionless variables are defined in Table 5.10. The following dimensionless equations result by using the definitions in Table 5.10 and dropping
the prime symbols from the dimensionless variable representations for simplicity

\[ \nabla \cdot \mathbf{v} = 0 \] (5.64)

\[ \sqrt{\text{Gr}} \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla P + \nabla^2 \mathbf{v} + \sqrt{\text{Gr}} \mathbf{e}_y T \] (5.65)

\[ \sqrt{\text{Gr}} \mathbf{v} \cdot \nabla T = \text{Pr}^{-1} \nabla^2 T \] (5.66)

where \( \mathbf{e}_y \) is the unit vector in the \( y \)-direction and the Prandtl number is \( \text{Pr} \equiv \nu/\alpha \).

The boundary conditions are shown in Fig. 5-45 and are

\[ u = v = 0 \quad \text{on} \quad \partial D \] (5.67)

\[ T = 0 \quad \text{on} \quad \partial D_4 \] (5.68)

\[ T = A \quad \text{on} \quad \partial D_2 \] (5.69)

\[ T = x \quad \text{on} \quad \partial D_3 \cup \partial D_1 \] (5.70)

**Finite Element Formulation**

A mixed element formulation is necessary for a convergent numerical scheme for Navier-Stokes equations. Biquadratic basis functions \( \Phi^i \) are chosen for the velocity, and lower order bilinear ones \( \Psi^{ij} \) are chosen for the pressure because of the compatibility condition known as *inf-sup* or *LBB* condition [36, 168, 12, 25]. Furthermore, a biquadratic approximation for temperature is chosen which is standard for natural convection problems. Therefore, the field variables are approximated as

\[
\begin{bmatrix}
u(x, y) \\
v(x, y) \\
T(x, y)
\end{bmatrix} = \sum_{j=1}^{N} \Phi^j(x, y) \begin{bmatrix} u \\ v \\ T \end{bmatrix}_j \] (5.71)

\[ P(x, y) = \sum_{j'}^{M} \Psi^{j'}(x, y)P_{j'} \] (5.72)
where the index \( j = \{1, 2, ..., N\} \) refers to the nodes of the biquadratic elements, and the index \( j' = \{1, 2, ..., M\} \) refers to the nodes of the bilinear elements. Choosing bilinear weighting functions for the continuity equation and biquadratic ones for momentum and energy equations gives the weak forms of Eqs. (5.61)-(5.63) as

\[
\int_A \Psi^{i'} \nabla \cdot \mathbf{v} = 0 \tag{5.73}
\]

\[
\sqrt{\text{Gr}} \int_A \Phi^i \mathbf{v} \cdot \nabla \mathbf{v} \, dA = \int_A \nabla \Phi^i P \, dA - \int_A \nabla \Phi^i \cdot \nabla \mathbf{v} \, dA + \sqrt{\text{Gr}} \, e_y \int_A \Phi^i T \, dA \tag{5.74}
\]

\[
\sqrt{\text{Gr}} \, \text{Pr} \int_A \Phi^i \mathbf{v} \cdot \nabla T \, dA = -\int_A \nabla \Phi^i \cdot \nabla T \, dA \tag{5.75}
\]

where \( \Phi^i \) are the biquadratic weighting functions and \( \Psi^{i'} \) are the bilinear weighting functions. The weighting functions are zero on all the boundaries because of the essential boundary conditions which are imposed.

The position of the mesh nodal points are approximated with biquadratic interpolants as

\[
x = \sum_{j'=1}^{9} x_{j'}^c \Phi^{i'}(\xi, \eta) \tag{5.76}
\]

\[
y = \sum_{j'=1}^{9} y_{j'}^c \Phi^{i'}(\xi, \eta) \tag{5.77}
\]

where the biquadratic interpolants are defined by Eqs. (5.42)-(5.50).

Substituting the approximate forms of the field variables into Eqs. (5.73)-(5.75) results in a set of nonlinear algebraic equations in the unknowns, \( \{u_j, v_j, T_j, P_{j'}\} \), which is solved by Newton's method, as described in Fig. 3-5. The elements of the Jacobian and the right side vector are formed at each Newton step using Gaussian quadrature as described in Section 5.3.2.
5.4 Multivariable PDE's

Solutions for Different Parameters and Validation of Results

Calculations are first carried out for low values of Grashof number where the flow is dominated by viscosity and buoyancy. The calculated field variables for $Gr = 5,000$, $Pr = 0.015$, and $A = 1.0$ are shown in Figs. 5-50 to 5-53 for several finite element meshes: $4 \times 4$ (Fig. 5-50), $8 \times 8$ (Fig. 5-51), $16 \times 16$ (Fig. 5-52), and $32 \times 32$ (Fig. 5-53). The results are calculated on four processors for the first three meshes and on 16 processors for the largest mesh. The solution is very simple for these parameter values; the flow contains only a single vortex in which the fluid moves up along the hot wall and down along the cold wall. Because of the small value of the Prandtl number and the weak convective flow, the temperature is little disturbed from the one-dimensional linear profile when the Prandtl number is zero and the temperature field decouples from the rest of the governing equations. Calculations with the smallest mesh capture the qualitative structure of the solution.

The calculated field variables are shown in Figs. 5-54 to 5-61 for the benchmark problem of natural convection in a closed cavity with parameters $Gr = 20,000$, $A = 4$, and $Pr = 0$ and 0.015. Four meshes ($8 \times 2$, $16 \times 4$, $32 \times 8$, and $64 \times 16$) are used for calculations at each value of the Prandtl number. The three smaller meshes are solved on 4 processors, and the largest one on 16 processors. All of the meshes capture the qualitative structure of the solution even for this large value of $Gr$; however, the solutions for the two smaller meshes are crude. Figures 5-54 to 5-57 are computations for $Pr = 0$ and the four finite element meshes. The energy equation is decoupled from the other equations for $Pr = 0$, so the temperature field is always linear in $x$ and independent of $y$. Figures 5-58 to 5-61 are for $Pr = 0.015$. Here the flow field affects the temperature, and the temperature contours show the influence of the flow which circulates counterclockwise. The flow consists of a single roll cell and the qualitative structures of the flow and the temperature fields for $Pr = 0.015$ are the same as the steady-state solution at a higher Grashof number of $10^5$ calculated by Winters. This solution is shown in Fig. 5-46 and is discussed above.
Figure 5-50: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 5,000$, $A = 1.0$. A $4 \times 4$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-51: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $\text{Pr} = 0.015$, $\text{Gr} = 5,000$, $A = 1.0$. An $8 \times 8$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-52: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 5,000$, $A = 1.0$. A $16 \times 16$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-53: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 5,000$, $A = 1.0$. A $32 \times 32$ mesh of rectangular elements is used. The results were computed on 16 processors.
Figure 5-54: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.0, Gr = 20,000, A = 4.0. An 8 × 2 mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-55: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.0$, $Gr = 20,000$, $A = 4.0$. A $16 \times 4$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-56: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.0$, $Gr = 20,000$, $A = 4.0$. A $32 \times 8$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-57: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are Pr = 0.0, Gr = 20,000, A = 4.0. A 64 × 16 mesh of rectangular elements is used. The results were computed on 16 processors.
Figure 5-58: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. An $8 \times 2$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-59: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. A $16 \times 4$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-60: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. A $32 \times 8$ mesh of rectangular elements is used. The results were computed on 4 processors.
Figure 5-61: The velocity, temperature, and pressure fields for natural convection in an enclosed cavity. The parameters are $Pr = 0.015$, $Gr = 20,000$, $A = 4.0$. A $64 \times 16$ mesh of rectangular elements is used. The results were computed on 16 processors.
Convergence of Newton iterations is shown in Fig. 5-62 for $Gr = 20,006$, $Pr = 0.015$, and $A = 4$; where the converged solution for $Gr = 5,000$, $Pr = 0.015$, and $A = 4$ was used as the initial guess for starting the Newton iterations. The error in Fig. 5-62 is computed as the maximum value of the correction to the solution at the $k^{th}$ iteration, i.e.

$$e_k \equiv |\delta u_j^k|_{max}, \quad j = \{1, 2, ..., N\}$$

(5.78)

where $N$ is the total number of unknowns. The error at successive Newton steps shrinks as

$$e_{k+1} \sim e_k^2$$

(5.79)

assuming the error at the $k^{th}$ step is smaller than one. That is, Newton's method converges quadratically from one iteration to the next; for example if $e_k \sim 0.1$, then $e_{k+1} \sim 0.01$. The error in Fig. 5-62 obeys Eq. (5.79) for $e_k < 1$. This attests to the consistency of Jacobian matrix entries with the residual equations.

The minimum and maximum values of the velocity components are plotted in Fig. 5-63 as a function of the number of degrees-of-freedom. For the third mesh, the maximum value for $u$ is accurate to three significant figures and the value of $v$ is accurate to two significant figures.

Two values of the velocity components have been used by others [230] to test the accuracy of the solution: the maximum value of $v(x, 0.5)$, and the maximum value of $u(A/4, y)$, which are referred to as $v_m$ and $u_m$, respectively. These values, which are calculated by the CFS algorithm for $Pr = 0.015$, are: $v_m = 0.479$ at $x = 2.438$, and $u_m = 0.685$ at $y = 0.156$. The values computed by CFS are compared with the results of Daube and Rida [71] who computed the same flow using a very different numerical method based on finite difference spatial discretization and implicit time integration coupled with a stream function-vorticity formulation. They report values of: $v_m = 0.474$ at $x = 2.43$ and $u_m = 0.684$ at $y = 0.16$ which are in excellent agreement with the calculations presented here. The calculations for $Pr = 0$ also are compared
Figure 5-62: Convergence of natural convection in an enclosed cavity with Newton iteration. The parameters are $Gr = 20,000$, $Pr = 0.015$, and $A = 4$. The initial guess is the converged solution for $Gr = 5,000$, $Pr = 0.015$, and $A = 4$. 
Figure 5-63: Maximum values of velocity components, \( z \equiv \frac{1}{N} \frac{\partial \rho}{\partial y} \) or \( \frac{\partial \rho}{\partial x} \), as a function of the number of unknowns for the four finite element meshes which are used for natural convection in an enclosed cavity with \( \text{Gr} = 20,000 \), \( \text{Pr} = 0.015 \), and \( A = 4 \).
5.4 Multivariable PDE's

<table>
<thead>
<tr>
<th></th>
<th>( \Pr = 0 )</th>
<th></th>
<th>( \Pr = 0.015 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>( v_m(x_m, 0.5) )</td>
<td>( v_m(A/4, y_m) )</td>
<td>( v_m(x_m, 0.5) )</td>
</tr>
<tr>
<td>present algorithm</td>
<td>0.5229 ((x_m = 2.4375))</td>
<td>0.6754 ((y_m = 0.1562))</td>
<td>0.479 ((x_m = 2.438))</td>
</tr>
<tr>
<td>Daube and Rida (1990)</td>
<td>0.5155 ((x_m = 2.44))</td>
<td>0.6753 ((y_m = 0.15))</td>
<td>0.474 ((x_m = 2.43))</td>
</tr>
<tr>
<td>Behnia and Vahl Davis (1990)</td>
<td>0.5226</td>
<td>0.6750</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.11: Maximum values for \( v(x, 0.5) \) and \( u(A/4, y) \) computed here compared to the values reported in the literature for natural convection in an enclosed cavity. The parameters are \( \Pr = \{0, 0.015\}, \text{Gr} = 20,000, A = 4.0 \).

with the values from the literature in Table 5.11. The values predicted by CFS are in good agreement with the results of Daube and Rida [71] for \( \Pr = 0 \). Two other investigators [14] reported values for \( v_m \) and \( u_m \) for \( \Pr = 0 \). These investigators used the stream function-vorticity formulation, and finite difference spatial discretization, but used a different numerical method based on the method of the false transient proposed by Mallinson and de Vahl Davis [183] to speed up the convergence of time integration to the steady-state solution. These authors reported results which were extrapolated to vanishing mesh size using Richardson extrapolation. The predictions for the maxima \( v_m = 0.5226 \) and \( u_m = 0.6750 \) agree to three significant figures to the values obtained here.

**Computational Efficiency**

The calculations reported here were performed with the program compiled without the IEEE option; see the curves in Figs. 5-8, 5-9, and 5-10 for the relevant performance of the Intel i860 processor for the LINPACK benchmark program. The operation counts are for nonzero scalar multipliers in vector operations; zero scalar multiplier operations are omitted because of their high speed; see Fig. 5-10.
Results are presented for the interior domain dissected 6 times into \( 8 \times 8 = 64 \) smaller subdomains in a manner similar to the results presented in Section 5.3.3 for Poisson's problem discretized with biquadratic finite elements. The smallest number of elements that is compatible with this dissection is \( 8 \times 8 \); therefore, the curves in the following figures have low cut-off points. For example, the minimum number of elements is \( 8 \times 8 \times 32 = 2048 \) for \( P = 32 \), which corresponds to 27,300 unknowns. Similarly, the curves extend to larger numbers of unknowns for larger \( P \) because more processors make possible the solution of larger problems. On the other hand, the maximum number of elements per processor gets smaller with increasing \( P \) because the size of the exterior matrix grows, and because the exterior matrix contributes to the storage overhead. The maximum number of elements in the \((x\text{-direction} \times y\text{-direction})\) per processor is \( 17 \times 16 \) for \( P = 1 \) and \( 12 \times 12 \) for \( P = 32 \).

The speedups are calculated based on the problem size per processor; i.e. the speed for \( P \) processors is divided by that for one processor, which has the same number of elements as does each one of the \( P \) processors.

The computation speeds, speedups, and times for \( LU \)-factorization, FE, and BE are reported per Newton iteration step.

**Absolute Speeds**

The computation speed for \( LU \)-factorization is shown in Fig. 5-64. The maximum possible speed is close to that for Poisson's problem discretized with biquadratic finite elements, as shown in Fig. 5-20. The curves for the computation speed tend to asymptote for large \( N \) for each number of processors as the vector performance of the machine saturates. The computation speed for the problem scales well with the increasing number of processors; Table 5.12 shows the maximum possible speed from Fig. 5-64 for various values of \( P \). The performance degrades only slightly from that for the Poisson's problem discretized with biquadratic finite elements. The speed increases by a factor of \( 1.82 \pm 0.11 \) on the average when the number of processors
Figure 5-64: Computation speed for $LU$-factorization as a function of the number of unknowns for natural convection in an enclosed cavity. Operation count includes only nonzero scalar multipliers in vector operations.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$R$</th>
<th>$\frac{R_P}{R_{P-1}}$</th>
<th>$\frac{R_P}{R_1}$</th>
<th>$\varepsilon \equiv \frac{R_P}{PR_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.4</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>12.4</td>
<td>1.9</td>
<td>1.9</td>
<td>0.97</td>
</tr>
<tr>
<td>4</td>
<td>23.4</td>
<td>1.9</td>
<td>3.6</td>
<td>0.91</td>
</tr>
<tr>
<td>8</td>
<td>40.9</td>
<td>1.7</td>
<td>6.4</td>
<td>0.80</td>
</tr>
<tr>
<td>16</td>
<td>69.9</td>
<td>1.7</td>
<td>10.9</td>
<td>0.68</td>
</tr>
<tr>
<td>32</td>
<td>132.3</td>
<td>1.9</td>
<td>20.7</td>
<td>0.64</td>
</tr>
</tbody>
</table>

Table 5.12: Maximum possible computation rate $R$ for different number of processors from Fig. 5-64.
is doubled. The ideal increase is a factor of two; therefore, the efficiency decreases by only a factor of $1.82/2 = 0.91$, and the maximum efficiency is estimated from the empirical relation

$$\varepsilon = 0.91^d$$

(5.80)

where $d$ is the dimension of the subcube. The above relation is very similar to the one for Poisson's problem discretized with biquadratic finite elements; see Eq. (5.58), but the constant of 0.92 in Eq. (5.58) is slightly larger. Setting an acceptable lower bound of 0.5 for the efficiency gives $7 \leq d \leq 8$, or $128 < P(=2^d) < 256$. The corresponding computation speeds are 423 and 770 MFLOPS, respectively.

Computation times for $LU$-factorization are shown in Fig. 5-65 for one Newton iteration. The maximum times range from about 8.5 seconds for $P = 1$ up to nearly 30 seconds for $P = 32$. The rise is about a factor of 3.5 for increasing from $P = 1$ to $P = 32$.

The computation speed for FE is shown in Fig. 5-66. The speed increases quickly for $P < 8$. However, the curves show degradation in performance for $P = 13$ and $P = 32$, caused by communication associated with pivoting for larger values of $P$. Computation times for FE are shown in Fig. 5-67. The times are small; about 1 second compared to 30 seconds for $LU$-decomposition.

The computation speed for BE is shown in Fig. 5-68. The computation speeds for BE are higher than the ones for FE in general, especially for larger values of $P$. Moreover, the curves for larger $P$ overtake the ones for smaller $P$ in all cases. Computation times for BE are also smaller than the times for FE, as indicated by comparing Fig. 5-69 with Fig. 5-67. The maximum computation time for BE of 0.56 seconds is nearly half of the maximum time of 1 second for FE.

The cost of formulation is significant, but it scales well with the number of processors, as shown in Fig. 5-70. The cost of formulation is second only to the cost of $LU$-factorization. There is no communication for formulation, and the only degradation comes from the arrangement of border data. A processor needs to sort and store
Figure 5-65: Computation times for LU-factorization as a function of the number of unknowns for the calculation of natural convection in an enclosed cavity.
Figure 5-66: Computation speed for FE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. The operation count involves only the vector operations with nonzero scalar multipliers.
Figure 5-67: Computation times for FE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity.
Figure 5-68: Computation speed for BE as a function of the number of unknowns for natural convection in an enclosed cavity. The operation count involves only the vector operations with nonzero scalar multipliers.
Figure 5-69: Computation times for BE as a function of the number of unknowns for computation of natural convection in an enclosed cavity.
Figure 5-70: Computation times for formulation as a function of the number of unknowns for the computation of natural convection in an enclosed cavity.
Table 5.13: Computation times per Newton iteration of the largest mesh possible for computation of natural convection in an enclosed cavity on one to 32 processors. The percentage of the total time consumed are shown in parentheses for each part of the solution.

The data for bordering separators until elimination of the interior is finished and the exterior separator matrix is updated. This additional work slows down the process of formulation, which goes down with increasing $P$ because the maximum number of elements per processor goes down.

The values of computation times for different parts of the solution per Newton iteration are summarized in Table 5.13. Forward elimination and back substitution comprise a very small fraction of the solution time. Most of the time is spent in formulating the Jacobian and its $LU$-factorization. The computation time for formulation is dominant for $P = 1$ at 59 percent of the total time, but it is reduced to 22 percent of the total time for $P = 32$. Therefore, further optimization of formulation may reduce the solution time significantly especially for smaller numbers of processors.
This optimization would not require any parallel programming because formulation requires neither message exchange nor synchronization of processors. In this case, it may be worthwhile to consider spectral element discretization, described in Section 5.3.4, for improving the time spent in formulation. Because of the coincidence of the mesh nodal point with the Gauss-Lobatto quadrature points for integration, formulation is considerably faster for spectral element discretizations. The values of the basis functions at the quadrature points are either one or zero, trivially reducing the number and complexity of computations that need to be performed. Further optimization of forward elimination and back substitution would not result in significant improvements in the total computation time because of the small fraction of the solution spent in these portions of the solution procedure. No optimization of formulation or any other portion of the code has been attempted for the results reported here.

**Speedups**

The speedups $S$ are calculated based on dividing speeds for the problems which have the same problem per processor. Speedups for $LU$-factorization are shown in Fig. 5-71 for calculation of natural convection in an enclosed cavity. The speedups do not show an increase with the number of unknowns. This flat speedup behavior is attributed to the method of speedup calculation. To see why, consider the lower limits of the curves in Fig. 5-71, i.e. the values of $S$ for small $N$. The absolute size of the problem is larger for $P$ processors in a speedup calculation even though the problem per processor is the same for both one and $P$ processors. Larger problems involve longer vectors in the operations, yielding more efficient vector calculations. Therefore, the $P$ processor problem comes closer to saturation speed for vector operations than does the one processor problem. In other words, the fact that the one processor problem operates at a lower speed than does the $P$ processor problem causes the speedup to be higher for smaller unknowns.
Figure 5-71: Speedups for $LU$-factorization as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedup calculation is based only on vector operations involving nonzero scalar multipliers.
Table 5.14: Speedups and efficiencies for the largest number of unknowns $N$ on $P$ processors for calculation of natural convection in an enclosed cavity.

For large unknowns, consider the end of the curves in Fig. 5-71. The size of the problem per processor is larger; therefore, the vector speed for the problem on one processor is larger and nearer saturation. The vector speed for the problem on $P$ processors is also larger, but still closer to the vector speed on one processor than it is for the low-$N$ limits of the curves. As $N$ increases, the vector speed for $P = 1$ approaches the speed for $P > 1$. In summary, the speed on one processor increases more quickly with increasing $N$ than does the speed on $P$ processors, causing the left ends of the curves in Fig. 5-71 to be pushed up.

Table 5.14 shows the speedups and efficiencies for the largest problems on $P$ processors. The speedups are high, and the drops in efficiencies are not large.

The speedups for FE are low in general, as shown in Fig. 5-72. The speedup values for FE increase up to $P = 8$, but the performance degrades for $P > 8$. The performance degradation is worse than it is for Poisson’s problem in Sections 5.3.3 and 5.3.4 because more pivoting is involved in solution of the natural convection problem. The speedups for BE are better, as shown in Fig. 5-73. The speedups for BE increase with increasing $P$, even though the magnitudes of the speedups are small. The speedups for FE and BE do not exhibit the flat behavior which is observed for those of $LU$-factorization because vector speeds do not reach saturation for any of the problems. Table 5.15 shows the maximum values of speedups for FE and BE.
Figure 5-72: Speedups for FE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. The speedup calculation is based only on vector operations involving nonzero scalar multipliers.
Figure 5-73: Speedups for BE as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. The speedup calculation is based only on vector operations involving nonzero scalar multipliers.
Table 5.15: The highest values of speedups for FE and BE for calculation of natural convection in an enclosed cavity.

The speeds for BE are more than half of the ideal value for $P \leq 8$, but smaller for $P > 8$. The speeds for FE are more than 50% of the ideal value for $P \leq 4$, but smaller for $P > 4$. The speeds in Table 5.15 for BE are invariably larger than the speeds for FE.

Fig. 5-74 shows the speeds for formulation. These are calculated by dividing the formulation times based on equal problems per processor for $P$ and one processors. Speedups are invariably high for formulation because formulation is highly parallel and no time is lost in communication. The degradation in formulation speedup is caused by the extra time spent managing the border data. Therefore, the formulation speedups are not the same for all processors. The processors in the interior domain share more borders with the other processors and therefore take more time to manage the border data. The speedup values for the worst cases are shown in Fig. 5-74; these speedups are for the processors that are located in the middle of the domain.

**Performance Comparison with a Parallel Iterative Solver**

Iterative methods are preferred by many because they require less memory, and therefore can solve larger problems than the direct methods can. Furthermore, efficient software for these methods is easier to develop. On the other hand, one advantage of the direct methods is their robustness; direct methods, unlike iterative methods,
Figure 5-74: Speedups for formulation as a function of the number of unknowns for calculations of natural convection in an enclosed cavity.
do not rely on using good preconditioners in order to converge at reasonable rates. This Section compares the time for calculation of natural convection in an enclosed cavity by the CFS algorithm with times for calculation by a highly efficient parallel conjugate-gradient-based Navier-Stokes solver [181, 94], called Nekton.

CFS requires more storage than Nekton; therefore, the largest problem solved by CFS is smaller. However, this maximum problem size is not small in the absolute sense, as shown in Table 5.13, and increases with the increasing number of processors. For example, 32 processors of the Intel iPSC/860 fit a problem with 61,000 unknowns.

Nekton uses a spectral spatial discretization and a semi-implicit temporal operator splitting, which produce a set of symmetric, positive-definite equations at each pseudo time step. A preconditioned conjugate gradient method is used for solving the linear system of equations. The preconditioned conjugate gradient iteration is based on a two-level iteration developed by Einar Rönquist, coupled with a local finite element method based preconditioner developed by Paul Fischer [95]. The two-level iteration computes the coarse modes directly, and computes the fluctuations about the (local) element average via conjugate gradients. The use of the coarse grid effectively filters low-modes out of the system and results in improved conditioning. Nekton uses single precision arithmetic; this speeds up Nekton’s computation rate compared to CFS, which needs to use double precision arithmetic for maintaining accuracy during LU-decomposition.

The natural convection in an enclosed cavity was computed with the parameters of $Gr = 1$, $Pr = 0.015$, and $A = 4$ with both algorithms. Even though the Grashof number is not high, the very low value of the Prandtl number causes the time scales for the flow and temperature to vary by two orders-of-magnitude, resulting in a stiff problem and requiring small steps for time integration. The pseudo time stepping scheme of Nekton makes possible the taking of larger steps for calculation of steady-state solutions. The low value of the Grashof number does not impose significant strains on the iterative solution scheme.
The parameters used for the calculations and the corresponding times for Nekton and CFS are summarized in Table 5.16 and described below.

For calculation by Nekton, the order of spectral basis functions used was 11 for velocity and temperature, and 9 for pressure. The mesh used was $16 \times 8$, resulting in 58,000 unknowns which were divided among 8 processors. Time integration was stopped after 100 pseudo time steps and the solution declared steady at that point. The total time for the calculation was 3,200 seconds. The calculation time for 32 processors is estimated at 800 seconds with the assumption of a perfect speedup of 4 from 8 to 32 processors. This assumption is reasonable considering the large size of the problem per processor.

For calculation by CFS, a finite element mesh of $96 \times 48$ was used with 61,000 unknowns, divided among 32 processors. The results converged after 3 Newton iterations with an absolute error $|\delta u_j^k|_{\text{max}} < 10^{-12}$ to the steady-state solution. The total time for the calculation was 120 seconds, 6.7 times faster than Nekton. The gap in timing is expected to grow for larger Grashof numbers when the steady-state solution stays stable because the problem gets more stiff. The gap may close for computations of time-dependent solutions.

### 5.4.2 Lid-Driven Flow in a Cavity

Fig. 5-75 contains a schematic of the problem. The bottom wall moves to the right, creating a circulating flow in the cavity. The flow has a one-dimensional solution for long cavities, where $A \gg 1$. In that case, the stream function reduces to

$$
\psi = y(1 - y)^2
$$

[5.81]

The velocity goes to zero at the center of the vortex, i.e. $u = \partial \psi / \partial y = 0$. The center occurs at $y = 1/3$.

The flow at the corners, where the moving and the stationary walls meet, was
Table 5.16: Performance comparison of CFS with Nekton for calculation of natural convection in an enclosed cavity with $Gr = 1$, $Pr = 0.015$, and $A = 4$. The calculation by CFS is 6.7 times faster.

![Diagram](image-url)

Figure 5-75: Diagram of the lid-driven flow in a cavity. The bottom wall moves to the right with a constant dimensionless speed of one.
analyzed by Taylor [254] and Moffat [199]. These corners cause singularities in the pressure field, as the discussion below demonstrates [199].

The Stokes equation for the stream function \( \psi(r, \theta) \), \( \nabla^4 \psi = 0 \), admits separated solutions in plane polar coordinate system of the form

\[
\psi = r^\lambda f_\lambda(\theta)
\]

(5.82)

where \( \lambda \) is a real or a complex number [199]. A polar coordinates is employed in the analysis such that the corner lies at \( r = 0 \). The plane at \( \theta = \pi/2 \) is at rest and the plane at \( \theta = 0 \) is scraped along parallel to itself with a dimensionless velocity equal to one. Near the corner, where the Stokes approximation is valid, the stream function is independent of the Reynolds number, and dimensional analysis shows that \( \psi \) must be of the form \( \psi = rf_1(\theta) \) where

\[
f_1(\theta) = a \cos \theta + b \sin \theta + c \theta \cos \theta + d \theta \sin \theta
\]

The corresponding velocity components are

\[
u = \frac{1}{r} \frac{\partial \psi}{\partial \theta} = f_1'(\theta)
\]

(5.83)

\[
u = \frac{\partial \psi}{\partial r} = -f_1(\theta)
\]

(5.84)

Applying the boundary conditions of

\[
f_1(0) = 0, \quad f_1'(0) = 1, \quad f_1(\frac{\pi}{2}) = f_1'(\frac{\pi}{2}) = 0
\]

(5.85)

gives the function \( f_1 \) as

\[
f_1(\theta) = \frac{\frac{\pi}{2} \sin \theta \left( \theta - \frac{\pi}{2} \right) + \theta \cos \theta}{1 - \frac{\pi^2}{4}}
\]

(5.86)

The pressure field is determined from the Stokes equation

\[
\nabla P = \nabla^2 \mathbf{v}
\]

(5.87)
which gives

\[ P = \frac{1}{r} \left( \frac{2 \cos \theta + \pi \sin \theta}{1 - \frac{n^2}{4}} \right) + P_0 \quad (5.88) \]

The pressure behaves as \(1/r\) close to the corner and grows without a bound as the corner is approached. The velocity field does not show any problems because it is independent of \(r\) close to the corner, as shown by Eqs. (5.83) and (5.84). The singularity in the pressure field causes problems for the convergence of the numerical solutions with the refining mesh. The pressure field and its gradient grow close to the corner as the mesh is refined. The problem of the pressure field can be circumvented by a stream function-vorticity formulation, or by changing the boundary conditions at the singular corners. Neither remedy is attempted here because the purpose of the calculations is only to test the speed of the algorithm.

**Equations and Boundary Conditions**

The flow is governed by the Navier-Stokes equations for an incompressible fluid, which are

\[ \nabla \cdot \mathbf{v} = 0 \quad (5.89) \]

\[ \rho \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla P + \mu \nabla^2 \mathbf{v} \quad (5.90) \]

The variables are made dimensionless with the characteristic scales which are summarized in Table 5.17. The dimensionless variables are summarized in Table 5.18. Using the definitions for the dimensionless variables and, for the sake of brevity, dropping the prime symbols give the dimensionless governing equations as

\[ \nabla \cdot \mathbf{v} = 0 \quad (5.91) \]

\[ \text{Re} \, \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla P + \nabla^2 \mathbf{v} \quad (5.92) \]
<table>
<thead>
<tr>
<th>variable</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>$(x, y)$</td>
</tr>
<tr>
<td></td>
<td>$H$</td>
</tr>
<tr>
<td>velocity</td>
<td>$(u, v)$</td>
</tr>
<tr>
<td></td>
<td>$U$</td>
</tr>
<tr>
<td>pressure</td>
<td>$P$</td>
</tr>
<tr>
<td></td>
<td>$\frac{\rho \nu^2}{H^2}$</td>
</tr>
</tbody>
</table>

Table 5.17: The scales used to put the variables into dimensionless form for lid-driven flow in a cavity. $U$ is the dimensional value of the speed of the lid.

\[
\begin{align*}
(x', y') & \equiv \left( \frac{x}{H}, \frac{y}{H} \right) \\
(u', v') & \equiv \left( \frac{u}{U}, \frac{v}{U} \right) \\
P' & \equiv \frac{H^2}{\rho \nu^2} P
\end{align*}
\]

Table 5.18: The definition of dimensionless variables as produced by using the scales which are summarized in Table 5.17 for governing equations of lid-driven flow in a cavity.

where $\text{Re} \equiv \frac{UH}{\nu}$ is the Reynolds number. The aspect ratio appears as a dimensionless number in the boundary conditions which are no-slip for the velocity as

\[
\begin{align*}
u = 0, u = 1 & \quad \text{on} \quad \partial D_1 \\
\end{align*}
\]

The calculations presented here are for $\text{Re} = 1$ and $A = 4$.

Finite Element Formulation

Finite element discretization for velocity and pressure is similar to what was used in Section 5.4.1. Biquadratic basis functions $\{\Phi^i\}$, as given by Eqs. (5.42)-(5.50), are used to approximate velocity, and bilinear functions $\{\Psi^i\}$, as given by Eqs. (5.16)-
(5.19), are used to approximate pressure as

\[
\begin{bmatrix}
  u(x,y) \\
v(x,y)
\end{bmatrix} = \sum_{j=1}^{N} \Phi^j(x,y) \begin{bmatrix}
u \\
v_j
\end{bmatrix}
\]

\[P(x,y) = \sum_{j'=1}^{M} \Psi^{j''}(x,y) P_{j'}\]  

(5.95)  

(5.96)

The weak forms of the continuity and the momentum equations are

\[
\int_A \Psi^i \nabla \cdot \mathbf{v} = 0
\] 

\[
\int_A \Phi^i v \cdot \nabla \mathbf{v} dA = \int_A \nabla \Phi^i P dA - \int_A \nabla \Phi^i \cdot \nabla \mathbf{v} dA
\] 

(5.97)  

(5.98)

where \{\Psi^i\} and \{\Phi^i\} are the same bilinear and biquadratic interpolants as the basis functions. These weighting functions are zero on the boundaries of the domain so that the weak forms satisfy the imposed essential boundary conditions. The approximate forms of the field variables are substituted into Eqs. (5.97) and (5.98) to give a nonlinear set of algebraic equations. These are solved by Newton iteration, as outlined in Section 5.4.1. At each Newton iteration, the elements of the Jacobian matrix and the right side vector are formed using Gaussian quadrature, outlined in Section 5.3.2.

Results

The results presented here are for the program compiled without the IEEE option; see Figs. 5-8, 5-9, and 5-10 for the performance of the LINPACK benchmark program. The subdomain of each processor is dissected 6 times; i.e. each processor dissects its subdomain into \(8 \times 8 = 64\) smaller subdomains. This sets a lower bound on the number of elements distributed to each processor. Because of this, the curves in the following figures do not extend to very low numbers of unknowns for large \(P\). On the other hand, larger numbers of processors can accommodate larger problems, so the curves for larger \(P\) extend to larger values of \(N\). Only timing results are presented
Numerical Experiments

here; the other results (for absolute speeds, speedups, and efficiencies) are similar to the ones presented in Section 5.4.1.

The computed velocity and pressure fields are shown in Figs. 5-76 to 5-81 for calculations with 2, 4, 8, 16, and 32 processors. The solutions shown are for the largest problems which fit on each number of processors. The mesh ranges from $21 \times 21$ for $P = 1$ (Fig. 5-76) to $136 \times 64$ for $P = 32$ (Fig. 5-81); these calculations have 4,182 and 79,339 equations, respectively. The velocity field does not change qualitatively with mesh refinement. However, the gradient of the pressure field develops larger values closer to the singular corners with the refining mesh. Even though $A$ is not very large, the flow in the middle of the cavity is parallel to the horizontal walls and exhibits features similar to the flow for an infinitely long cavity. The center of the vortex occurs at $y = 0.336$ for the $136 \times 64$ mesh, comparing favorably with the analytic result of $1/3$ as predicted by Eq. (5.81) for a long cavity.

Timing Results

The computation times for $LU$-factorization are presented in Fig. 5-82. The slopes of the curves, which are indications of processor utilization, decrease with increasing $P$. Computation times are below 28 seconds in all cases.

The computation times for FE and BE are shown in Figs. 5-83 and 5-84, and are low, as they were in Sections 5.3.2, 5.3.3, 5.3.4, and 5.4.1. The maximum computation time for FE is 1 second and the maximum computation time for BE is 0.54 seconds. Again, FE performance is degraded by pivot exchanges despite the low absolute computation times for FE.

Again, the computation time for formulation is significant, as shown in Fig. 5-85. The maximum time for formulation is 11 seconds (56% of the total) for one processor, and 8 seconds (23% of the total) for 32 processors.

The computation times per Newton iteration are summarized in Table 5.19, where the results are for the largest problems on one to 32 processors. Again, the formulation
Figure 5-76: The velocity and pressure fields for lid-driven flow in a cavity with $A = 4.0$ and a $21 \times 21$ mesh run on 1 processor.
Figure 5-77: The velocity and pressure fields for lid-driven flow in a cavity with $A = 4.0$ and a $42 \times 20$ mesh run on 2 processors.
Figure 5-78: The velocity and pressure fields for lid-driven flow in a cavity with $A = 4.0$ and a $40 \times 38$ mesh run on 4 processors.
Figure 5-79: The velocity and pressure fields for lid-driven flow in a cavity with \( A = 4.0 \) and a \( 76 \times 36 \) mesh run on 8 processors.
Figure 5-80: The velocity and pressure fields for lid-driven flow in a cavity with $A = 4.0$ and a $68 \times 68$ mesh run on 16 processors.
Figure 5-81: The velocity and pressure fields for lid-driven flow in a cavity with $A = 4.0$ and a $136 \times 64$ mesh run on 32 processors.
Figure 5-82: The computation times for LU-factorization as a function of the number of unknowns for lid-driven flow in a cavity with $A = 4$. 
Figure 5-83: The computation time for FE as a function of the number of unknowns for lid-driven flow in a cavity with $A = 4$. 
Figure 5-84: The computation times for BE as a function of the number of unknowns for lid-driven flow in a cavity with $A = 4$. 
<table>
<thead>
<tr>
<th>$P$</th>
<th>mesh</th>
<th>unknowns</th>
<th>formulation</th>
<th>$LU$</th>
<th>FE</th>
<th>BE</th>
<th>total</th>
</tr>
</thead>
</table>
| 1   | 21 $\times$ 21 | 4,182    | 11.2        | 8.6  | 0.1 | 0.1  | 20.103 (%)
|     |          |          | (55.9%)     | (42.8%) | (0.66%) | (0.69%) |
| 2   | 42 $\times$ 20 | 7,873    | 11.0        | 10.4 | 0.2 | 0.1  | 21.67 (%)
|     |          |          | (50.8%)     | (47.8%) | (0.70%) | (0.68%) |
| 4   | 40 $\times$ 38 | 14,075   | 10.6        | 13.3 | 0.2 | 0.2  | 24.257 (%)
|     |          |          | (43.6%)     | (54.9%) | (0.84%) | (0.70%) |
| 8   | 76 $\times$ 36 | 25,187   | 9.9         | 16.2 | 0.3 | 0.2  | 26.605 (%)
|     |          |          | (37.2%)     | (61.0%) | (1.02%) | (0.76%) |
| 16  | 68 $\times$ 68 | 42,299   | 8.9         | 21.9 | 0.5 | 0.3  | 31.58 (%)
|     |          |          | (28.1%)     | (69.3%) | (1.65%) | (1.00%) |
| 32  | 136 $\times$ 64 | 79,339   | 8.5         | 27.7 | 1.0 | 0.5  | 37.805 (%)
|     |          |          | (22.6%)     | (73.2%) | (2.76%) | (1.43%) |

Table 5.19: Computation times for one Newton iteration of the largest mesh possible for computation of lid-driven flow in a cavity on one to 32 processors. The percentages of the total time consumed are shown in parentheses for each part of the solution.

and $LU$-factorization consume most of the total computation time in all cases.

**Performance Comparison with a Serial Frontal Solver**

Initially, the frontal solver was introduced for the solution of symmetric, positive-definite systems [153], and was later extended to include asymmetric, indefinite systems [139, 140]. Frontal solvers have been used successfully by many investigators for solving the linear equation sets that arise in numerical solutions of general two-dimensional boundary value problems; for example, the results presented in Chapter 2 were calculated using a frontal solver. Frontal methods lessen the large storage requirement of $LU$-factorization by using auxiliary storage; they are amenable to efficient vectorization and have been used on vector supercomputers very successfully;
Figure 5-85: Computation times for formulation as a function of the number of unknowns for calculation of lid-driven flow in a cavity with \( A = 4 \).
in addition, they are robust, applying to general finite element discretizations. The calculations reported in this section will be used to compare the performance of CFS with a serial frontal algorithm.

An efficient Navier-Stokes solver is used for the comparison [20]. This algorithm uses the same finite element discretization described in this Section for CFS, and it uses Newton's method for solving the nonlinear set of algebraic equations. The difference between this algorithm and CFS is that the former uses a frontal solver for formulation and solution of the linear systems of equations at each Newton iteration step, whereas CFS uses the method described in Chapter 4. The frontal solver assembles equations element by element in a working space called the front. The equations which are fully-summed, i.e. contributions from all the elements are added and factored; and the factors are stored in an auxiliary storage medium to create space for the remaining equations to be assembled and factored. Therefore, the front moves through the Jacobian matrix, assembling and factoring the equations in succession.

CFS and the frontal algorithm were first tested on a Cray X-MP. A $68 \times 68$ finite element mesh was used ($N \approx 42,000$) on one dedicated Cray X-MP processor and 16 processors of the Intel hypercube. The CPU time for one Newton iteration was 162 seconds for the serial frontal solver, 144 seconds for CFS on the Cray, and 31.6 seconds for CFS on 16 processors of the hypercube. CFS runs only 12% faster than the serial frontal solver on the Cray but 5 times faster on 16 processors of the hypercube. CFS performs more slowly on the Cray is for several reasons. First of all, integer addressing overhead is paid by one processor rather than by 16. Next, CFS does not vectorize as well as the frontal solver. Lastly, CFS is not optimized as well as the frontal solver. The total elapsed time for the computation is 15 minutes for the serial frontal code, 13 minutes for CFS on the Cray, and 2 minutes for CFS on the hypercube.

Afterward, the time of computation was compared for the largest problem on 32 processors by CFS (a $136 \times 64$ mesh; $N \approx 80,000$) to the time of computation of the same problem by the serial frontal algorithm. The timing comparisons are
Table 5.20: Performance comparison with a serial frontal solver on a Cray X-MP.

summarized in Table 5.20. CFS is faster by an order-of-magnitude; it reduces both the time per Newton iteration and the total solution time by the large factors of 7 and 10, respectively. Because the cost of the Cray is one order-of-magnitude larger than the cost of the hypercube, a two order-of-magnitude savings in costs is possible by using CFS on the hypercube rather than the traditional frontal code on the Cray. It should be mentioned that some of the superior performance of CFS is because of parallelism and some because of the lower computational complexity of nested dissection over the band ordering used by the frontal solver.

5.5 Memory Limitations and Sizes of the Problems

The importance of problem size on scalability was emphasized in Chapter 3. The maximum numbers of unknowns which fit on the processors are shown in Figs. 5-86, 5-87, and 5-88 as a function of the number of processors for the three test problems: Poisson's problem discretized with biquadratic finite elements (Fig. 5-86), lid-driven flow in a cavity (Fig. 5-87), and natural convection in an enclosed cavity (Fig. 5-88). These problems have one, three, and four degrees-of-freedom per element, respectively. Curves are shown for levels of interior dissection ranging from 0 to 6, i.e. from
Figure 5-86: Maximum number of unknowns as a function of number of processors for various levels of interior dissection for Poisson's problem discretized with biquadratic finite elements.
Figure 5-87: Maximum number of unknowns as a function of number of processors for various levels of interior dissection for lid-driven flow in a cavity.
Figure 5-88: Maximum number of unknowns as a function of number of processors for various levels of interior dissection for natural convection in an enclosed cavity.
one to $8 \times 8 = 64$ smaller subdomains for each processor. The maximum number of unknowns increases with the level of dissection; this is expected because more dissection results in more efficient use of sparsity and less memory usage. The maximum number of unknowns decreases as the degrees-of-freedom per element increases, because problems with the same number of unknowns but more degrees-of-freedom per element have less sparsity. The fact that the maximum number of unknowns increases nearly linearly in all cases attests to the excellent scalability due to efficient management of sparsity.

5.6 Conclusions

The results presented in this Chapter document the excellent performance of CFS as described in Chapter 4 for three representative test problems defined on two-dimensional grids. The computation time for $LU$-factorization dominates the solution time. Formulation consumes the largest fraction of the total computation time after $LU$-factorization. The computation times for FE and BE trail far behind. Matrix formulation and $LU$-decomposition show excellent performance and scalability as the number of processors is increased. The performance of FE degrades somewhat with increasing $P$ because of the required pivot exchanges, but the computation times for FE remain very low. Furthermore, the performance of FE can be improved (if desired) by methods described in Chapter 4. BE’s performance is superior to FE’s, and while the speedups for BE increase with increasing $P$, they are lower than the speedups for either matrix formulation or $LU$-factorization. Overall, the scalability of BE is satisfactory given its very low computation times and the inherently non-parallel nature of the algorithm. The triangular solutions consume a small fraction of the total solution time per Newton iteration (about 4% at most). If a modified Newton’s method is used, such as was the case for the transient calculations in Chapter 2, the triangular solutions will consume a larger portion of the computation time in each
iteration, and further optimization of these algorithms will be useful. The spectral element discretizations show superior speeds and speedups; even the BE speedup reaches a sizable 15.2 for $P = 32$.

The applicability of the algorithm to physical problems is demonstrated by solving general flow problems; its performance degrades minimally. The performance comparisons of finite element/Newton/CFS with two state-of-the-art solvers, Nekton on the Intel hypercube, and the finite element/Newton/serial frontal algorithm on the Cray, demonstrate the nearly order-of-magnitude savings in time by CFS on the Intel hypercube. Moreover, with CFS, the maximum number of unknowns $N$ scales well with increasing processors $P$, indicating efficient memory usage.

It should be emphasized that CFS is very general. Because it includes pivoting, it will work for a wide range of mathematical models for complex physicochemical systems; the finite/spectral element discretizations of these problems generate asymmetric, indefinite matrices that are not easily solved with iterative solution methods.
Chapter 6

Discussion

The purpose of the research presented in this thesis is to contribute to understanding transport processes in directional solidification. This objective was fulfilled by following two paths. The first path, studying the stability and nonlinear behavior of binary alloys, was a direct and immediate contribution. The second path, developing an efficient numerical solver of complex mathematical models, was a less direct, but more general contribution. Nonetheless, the second contribution laid an important foundation for reducing the costs of the future theoretical investigations not only for directional solidification but also for any physicochemically complex system. Below, a summary of the achievements is presented. Some directions for future research are discussed following the summary.

6.1 Linear Stability and Nonlinear Evolutions in Binary Alloy Solidification

Current state-of-the-art numerical methods were used to study the behavior of dilute binary alloys during directional solidification. Fully nonlinear simulations were combined with linear stability theory as a powerful method for studying the stability and nonlinear behavior of these systems. Linear stability was used to compute the onset of
instability at a critical concentration \( c_{0c} \) for which the planar melt/crystal interface becomes unstable to a disturbance of small amplitude. The critical concentration was computed as a function of the wavenumber (or wavelength) of the disturbances. Nonlinear simulations were carried out to study the states with a curved melt/crystal interface and with convective flow. These nonlinear states appeared when the planar melt/crystal interface became unstable.

Tin-lead and succinonitrile-acetone binary alloys were considered; these alloys have been used widely by other investigators \([59, 60, 16, 121, 72]\) as prototypical systems for both theoretical and numerical investigations of binary alloy solidification because of their low melting points, well-characterized physical properties, and in the case of succinonitrile, transparency. Linear stability and nonlinear evolutions were studied in rectangular and cylindrical geometries with rigid and shear-free sidewalls. The rectangular geometry with shear-free sidewalls corresponds to the infinite system for which linear stability analysis is usually performed \([59, 60, 203, 213]\). This system is only a good approximation when the wavelengths of instabilities are small compared to the lateral dimensions of an actual crystal growth configuration. Conversely, the cylindrical geometry with rigid sidewalls is a good approximation of the actual vertical Bridgman crystal growth configuration when the wavelengths of the instabilities are comparable to the lateral dimensions of the actual system.

Convective supercooling instabilities, which had previously been found to be the result of the coupling between the melting temperature and the concentration field \([60]\), were investigated. Here, the temperature field was stabilizing, and the coupling of the fluid flow with the concentration of the solute was ignored. The wavelengths of the instabilities were on the order of several centimeters in general, which is comparable to the dimensions of an ampoule in a typical vertical Bridgman crystal growth system. The effects of the boundary conditions and geometry were small when the value of \( c_0 - c_{0c} \) was small; that is, the lateral variations in the field variables were small. However, the solutions showed qualitatively different behaviors as \( c_0 - c_{0c} \) in-
creased. The interaction between different finite amplitude solutions was weak. Even though strong secondary flows were computed as a function of \( c_0 - c_{0c} \), the temperature field departed minimally from the linear profile for unidirectional conduction, and the interface shape did not depart significantly from the planar shape. The nearly planar interface shape is counter-intuitive because the origin of these instabilities is morphological. The ratio of thermal conductivities of solid to melt is an important driving force for these convective supercooling instabilities, as was pointed out by Coriell and McFadden [60] and was demonstrated in Chapter 2. Slight imperfections caused by heat loss through the lateral boundaries left a continuous transition for the onset of flow and curved melt/crystal interfaces. In fact, heat leakage through the sidewalls, which cannot be avoided in real crystal growth systems, caused very strong secondary flows compared to the convective motion due solely to the convective supercooling instabilities.

Thermosolutal instabilities were studied for a succinonitrile-acetone system. The wavelength for these instabilities is on the order of a millimeter, and hence much smaller than the ampoule dimensions in a crystal growth system. Therefore, calculations aimed at studies of these flow structures were based on the rectangular geometry with shear-free sidewalls. First, the thermal conductivity of the solid was set equal to the value for the melt, and the heat of solidification was set to zero to rule out the convective supercooling instabilities which were mentioned above. With these approximations, the temperature field was fixed except for the contributions of convective heat transfer driven by the solidification velocity and by thermosolutal motion. Calculations of the nonlinear steady-states that evolved from the neutral stability curves developed large interface deflections similar to the deep cells observed for a purely morphological instability. Calculations could not continue beyond five percent above the critical concentration because of the very large interface deflections. Refining the finite element mesh did not improve the solution quality or result in convergence for these computations; this suggested that the interface shape prob-
ably became reentrant because of constitutional supercooling, as described by Tiller et al. [255].

These calculations demonstrated that convection induced by the thermosolutal instability cannot be thought of as being decoupled from the interface shape. In fact, the computed interface deformation was as large as what is traditionally associated with finite amplitude solidification cells. Therefore, the usual labeling of the linear stability curves as either convective or morphological might be quite misleading when considering the finite amplitude behavior of the system. Calculations also were carried out for a succinonitrile-acetone alloy with a nonzero heat of solidification to compute a time-periodic state for a small \( c_0 - c_{0c} \). This time-periodic state also showed pronounced morphological and convective character.

The interactions of the nonlinear states also were studied close to the point where oscillatory-convective and morphological branches meet on the linear stability diagram, a codimension two point formed between a Hopf and a stationary bifurcation point. This study was performed for a contrived alloy with a very large surface energy in order to sharpen the flat bottom of the morphological curve so that only a few morphological modes became unstable at concentrations slightly above the critical concentration for the onset of the oscillatory mode. The results showed that the first bifurcating solution family at the morphological onset was the stable solution; oscillatory solutions could not be computed because these states were unstable to perturbations that led back to the stable steady-state solution family.

The studies of binary alloy solidification demonstrated the high degree of coupling of flow, temperature, and concentration fields. The temperature field departed very slightly from the linear profile in all cases even for the succinonitrile-acetone system, which has a large Prandtl number. Alternatively, the nonlinear states that resulted from thermosolutal and convective supercooling instabilities did not drive a flow strong enough to cause large lateral variations in the temperature profile, and hence it was concluded that these instabilities do not contribute to large thermal
stresses in the crystal. The thermal stresses in the crystal are mainly due to lateral
temperature gradients caused by the heat leakages through the sidewalls of the crys-
tal growth system. However, the flow does redistribute the solute, causing significant
variations in crystal composition. The scale of variations in crystal composition ranges
from the the centimeter scale, corresponding to the convective supercooling instabili-
ties studied in the former part of Chapter 2 to the millimeter scale, corresponding to
the thermosolutal instabilities studied in the latter part of Chapter 2.

6.2 A Robust Method for Solution of Complex
Mathematical Models with Parallel Comput-
er

The continuous demand for higher computational power motivated the other objective
undertaken in this work. Higher computation power, which corresponds to faster
simulations, has the advantages of expanding the limits of practical simulations and
of enabling more thorough parametric studies, as the studies performed in Chapter 2
demonstrate.

The robust CFS algorithm was developed for direct solution of linear equation
sets which arise in the context of finite and spectral element discretizations of physi-
cochemically complex problems on MIMD parallel computers. The combination of
domain decomposition and nested dissection allows efficient incorporation of the spar-
sity of the matrix, as well as efficient exploitation of parallelism. Also, the algorithm
incorporates partial pivoting for accurate solution of the indefinite and asymmetric
equation sets that result from discretization of large classes of differential equations.
CFS algorithm is hierarchical in all of its steps: ordering, symbolic factorization, for-
mulation, numeric factorization, and triangular solutions. Task scheduling is based
on the subtree-to-subcube mapping, which is the most communication-efficient map-
ping that has been developed. A hybrid of fan-in/fan-out updating strategy is used for reducing the communication overhead.

The performance of CFS was demonstrated on a 32-node Intel iPSC/860 parallel computer. Three representative test problems on two-dimensional grids were examined: Poisson's problem, natural convection in an enclosed cavity heated from the sides, and lid-driven flow in a cavity. At each Newton iteration step, the formulation and $LU$-decomposition steps demonstrated good speedups which scaled well with the increasing number of processors. The performances of forward and backward elimination steps were lower. The performance of backward elimination improved in all cases for large problems as the number of processors increased. Unfortunately, the performance of forward elimination did not improve with the increasing number of processors in all cases, not even for the larger problems. Notwithstanding this fact, the poor parallel performance of the forward elimination had a very small effect on the overall performance of the algorithm because of the very small amount of time spent in this step of the algorithm. If many triangular solutions are required for each factorization step, for example when modified Newton's method is used, the cost of triangular solutions may become a significant fraction of the overall solution procedure cost. In that case, the scheme proposed in Section 4.3.7 may be employed to increase the performance of the forward elimination step.

Applicability of the algorithm to physical problems was demonstrated through the solution of general flow problems involving natural and forced convection. The performance of the algorithm degraded minimally for solution of these flow problems. The performance comparisons of CFS with two state-of-the-art solvers, Nekton on the Intel hypercube and a serial frontal algorithm on the Cray, demonstrated the nearly order-of-magnitude savings in time by CFS on the Intel hypercube. The savings in costs were two orders-of-magnitude when compared with the serial frontal algorithm on the Cray X-MP, but only because the Cray costs were already an order-of-magnitude higher than the Intel hypercube costs. Finally, it was demonstrated that
the maximum number of unknowns scaled almost linearly with increasing processors, indicating efficient memory usage by the CFS algorithm.

6.3 Future Directions

The investigations in this work lay the foundations for much exciting and fruitful future research. Below, some suggestions are outlined for future extensions of the current work. Some of these suggestions are straightforward extensions of the current work. Some require more effort, forethought, and preliminary investigations.

6.3.1 Thermosolutal Convection Coupled with morphology in Binary Alloy Solidification

Strong couplings among the flow field, concentration field, temperature field, and interface morphology result in highly deformed melt/crystal interfaces, as demonstrated in Chapter 2. Calculations of nonlinear solution families could not be carried out for bulk concentrations more than a few percent above the critical onset because of the high deformations of the interface. To study these nonlinear states, more general numerical techniques, such as the mapping techniques developed and used by Brown and Tsiveriotis [256, 257, 258], need to be combined with the numerical methods used in Chapter 2. Because the problem of thermosolutal convection coupled with a highly deformed interface results in large numbers of unknowns and large computational complexities, it is recommended that CFS be used to facilitate faster and cheaper computation of the solutions.

6.3.2 CFS Code Optimization

Despite its good performance, CFS code is not optimized for use on one particular processor architecture. Optimization of the code for better vectorization and pipelining will increase the speed. Furthermore, the improvement of the forward elimination
algorithm as suggested in Chapter 4 may prove worthwhile for transient computations which use a modified Newton's method requiring many triangular solutions per factorization step. Finally, integer addressing should be optimized to reduce the overhead.

6.3.3 Transient Simulations Using CFS Algorithm

Currently, time integration is not incorporated in the CFS algorithm. Since many interesting practical situations involve solution of time-dependent differential equations, it is imperative that the time integration procedure be built into CFS. This extension is quite straightforward. Time integration is built around the steady-state algorithm and performed sequentially. No new ideas need to be developed because time integration has been implemented successfully in sequential environments in conjunction with Newton's or modified Newton's methods for complex problems; examples include work by Brown and Kim [30, 163] and by Brown and Tsiveriotis [256, 257].

6.3.4 More Sophisticated Domain Decomposition

The automatic nested dissection algorithm presented in Chapter 4 works only for rectangular domains. Real physical problems often involve complicated geometries. More general nested dissection algorithms need to be incorporated into CFS so that the solution of problems with more complicated geometries will be possible. Different mathematical models represent different parts of the domain in multiphase problems. Load balancing for these problems, which have different numbers of variables and equations associated with each section, needs to be considered.

Many heuristic algorithms have been developed for automatic nested dissection [107, 110, 111] and automatic domain partitioning [267, 264, 126] of general domains, which can be implemented in CFS. The best results are obtained by applying these domain partitioning algorithms only to divide the complex geometry into large rectangular subdomains. The rectangular subdomains, each of which consists of a single phase,
are chosen so as to balance the load and to minimize exterior separator lengths. Then,
the ordering or decomposition of each of these subdomains is performed with the fast
heuristic nested dissection algorithm presented in Section 4.3.3. In this manner, the
overall domain partitioning can be performed by a sophisticated algorithm in a short
time because the domain decomposition is coarse at this level, and the number of the
resulting rectangular, single-phase subdomains is not large. The nested dissection
of the resulting rectangular subdomains is performed with the fast heuristic algo-
rithm presented in Section 4.3.3. The level of decomposition at the first stage by
the general domain partitioning algorithm may be chosen independent of the number
of processors. The partitioning at this stage may be even coarser than the domain
decomposition for allocation of the subdomains to the processors.

6.3.5 Parallel Solution of Multiphase Problems

Application of CFS to directional solidification problems requires the solution of mul-
tiphase problems. This may be achieved in one of the following ways.

The first way is to use mapping techniques as follows. Subsequent to imple-
mentation of a heuristic algorithm for domain decomposition of multiphase domains,
mapping techniques such as the ones used by Brown and Tsiveriotis [256, 257, 258]
are implemented to determine the position of the mesh nodal points. Here the mesh
is fixed to the interface. The implementation involves adding mapping equations and
unknowns for the position of the finite element mesh points. These mapping equa-
tions are solved together with equations for other field variables and for interface
position at each Newton iteration step to determine the position of the mesh. Since
CFS is implemented to incorporate multiple variables and equations, the addition of
mapping equations does not modify the algorithm at all.

A second way to extend CFS to multiphase systems involves using phase field
theory. Recent progress has been made in the application of phase field theory to
directional solidification systems [127]. Briefly, a field equation is introduced for the
phase field $\phi$ which determines the phase of the system. The limits of $\phi$ correspond to liquid ($\phi = 1$) and solid ($\phi = 0$) phases. The changes in $\phi$ correspond to phase changes. The equations for the field variables such as temperature and concentration are coupled to the phase field. Also, the phase field gives the correct limits for the physical properties of the system, for example, the thermal conductivity, which is expressed as

$$k = \phi k + (1 - \phi)k_s$$

where the thermal conductivity of the system approaches that of the liquid $k$ and that of the solid $k_s$ in the appropriate limits. The advantage of the phase field is that only one field equation need be written for the whole system. For example, a single field equation for temperature governs heat transport in the whole system. This equation approaches the appropriate limit in the liquid and in the solid. The mesh is not fixed to the interface. Therefore, sophisticated mapping techniques need not be employed to handle highly deformed interfaces. One challenge here is the development of a phase field formulation for flow problems. The other challenge is choosing the appropriate parameters for the phase field equation to describe the phase change appropriately. Overall, the phase field approach is less straightforward and more uncertain, but it may prove a much more powerful technique than the mapping techniques described above.

CFS can be applied to simulations of complex directional solidification systems subsequent to implementations of a general automatic domain partitioning algorithm and mapping techniques, or of phase field formulation for multiphase problems. The extensions to CFS described below will be worthwhile in that they will increase the power and speed considerably.
6.3.6 Spectral Element Methods and $hpr$-Refinement

The superior performance of CFS with spectral elements was demonstrated in Chapter 5. Also, it was noted that spectral element discretizations have smaller complexities when solutions with high accuracies are desired. Oden [211] argued that for realistic simulations it is necessary to develop algorithms which not only refine the mesh ($h$-refinement), but also refine the polynomial order ($p$-refinement), and furthermore, move the element positions ($r$-refinement) automatically and intelligently to produce a solution with a certain, predetermined accuracy. The traditional mesh refinement by looking at the solution and deciding where the refinement is necessary is inefficient and simply not sufficient for highly intensive computations. Implementation of a "smart" $hpr$-refinement strategy in CFS should prove very valuable in the future.

6.3.7 Solution of Three-Dimensional Transport Problems

Simulations which include three spatial dimensions are quite costly. However, since the real, physical systems exhibit three-dimensional behavior except in very limited instances, it is imperative to increase the computational power so as to lower the computation times and the associated costs for three-dimensional computations. The CFS algorithm has performed well for two-dimensional problems, but the question remains whether the generalization of CFS for solution of three-dimensional problems will perform as efficiently.

To find out the answer to this question, consider a comparison for the cost of computations in two and three dimensions for discretizations which give similar accuracies, that is an $n \times n$ grid in two dimensions and an $n \times n \times n$ grid in three dimensions. In the simplest case, the cost of computations is proportional to the total operation count and the amount of required computer storage. These two variables are compared for the two- and three-dimensional computations in the analysis below.
To the leading order, the operation count for solution of a linear system, which is ordered with the nested dissection method, is proportional to the operation count for $LU$-factorization of the largest separator. The largest separator has a size of $n$ for the $n \times n$ two-dimensional problem and has a size of $n^2$ for the $n \times n \times n$ three-dimensional problem. That is, the largest separator corresponds to an $n \times n$ dense matrix for the two-dimensional problem, and to an $n^2 \times n^2$ dense matrix for the three-dimensional problem. Therefore, the ratio of the operation count for the three-dimensional problem $C_{3-D}$ to the operation count for the two-dimensional problem $C_{2-D}$ to the leading order is

$$\frac{C_{3-D}}{C_{2-D}} \sim \frac{n^6}{n^5} = n^3 \quad (6.1)$$

It is noteworthy that the band ordering gives exactly the same ratio. However, the operation count for the band ordering is $O(n)$ larger for both the two- and the three-dimensional computations.

Consider the lid-driven cavity problem to see how the above ratio affects the computation time. The largest problem that was tested on 16 processors of the Intel hypercube in Chapter 5 was for a $68 \times 68$ mesh with 42,300 unknowns. The length of the largest separator is 343 for this problem, where two biquadratic degrees-of-freedom for the velocity and one bilinear degree-of-freedom for the pressure exist per finite element. The $68 \times 68 \times 68$ problem in three dimensions has 5.5 million unknowns, and the length of the largest separator is 42,300. Therefore, the ratio of the operation counts from Eq. (6.1) is about 2 million for these examples! If the three-dimensional problem is distributed among $4 \times 4 \times 4 = 64$ processors, the time for the solution will be 500,000 times longer for the three-dimensional problem. Alternatively, if the three-dimensional problem is to be solved in a comparable amount of time, each processor needs to operate half-a-million times faster. In concrete terms this means that a speed of 5 tera FLOPS per processor is required to compute the solution for the above three-dimensional lid-driven cavity problem on 64 processors in the same amount of time as the two-dimensional problem on 16 processors of the Intel iPSC/860 hypercube.
This incredibly large speed is unlikely to be realized in the near future.

Not only is the operation count large, but also the storage requirement is high for three-dimensional problems. An analysis similar to the presentation in Chapter 4 for storage requirements of three-dimensional problems indicates that the ratio of the memory requirement for the three-dimensional problem $M_{3-D}$ to that for the two-dimensional problem $M_{2-D}$ is

$$\frac{M_{3-D}}{M_{2-D}} \sim n^2 \frac{P_{2-D} \log_2 P_{3-D}}{P_{3-D}^{4/3} \log_2 P_{2-D}}$$  \hspace{1cm} (6.2)$$

where $P_{2-D}$ is the number of processors employed to solve the two-dimensional problem and $P_{3-D}$ is the number of processors for the three-dimensional problem. This ratio is about 1400 for the above example of the lid-driven cavity problem, where $P_{2-D} = 16$, $P_{3-D} = 64$, and $n = 68$ elements. That is, about 9 Gbytes of memory are required per processor for the three-dimensional problem.

The above estimations indicate that CFS will not be directly useful for large, three-dimensional computations. However, the utility of CFS will be as a preconditioner for iterative solution methods which do not require as much storage and have lower complexities. CFS will be useful for solving the problem on a coarse grid and applying the L and the U factors for conditioning a finer discretization [95]. CFS will be an especially valuable preconditioner because of its robustness.

Combining CFS with iterative solvers is not the ultimate answer. The above large estimates for operation counts and computer storage requirements of three-dimensional problems indicate the importance of $hpr$-methods, as discussed by Oden [211], for producing solutions with the desired accuracy by using the smallest number of unknowns. This is an area open for future investigation.
Bibliography


[63] S. R. Coriell, B. T. Murray, G. B. McFadden, and K. Leonartz. Convective and morphological stability during directional solidification of the succinonitrile-


