Spectral Element Solution of the Navier-Stokes Equations on High Performance Distributed-Memory Parallel Processors

by

Paul Frederick Fischer

B.S., Cornell University, Ithaca, NY (1981)
M.S., Stanford University, Palo Alto, CA (1982)

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Doctor of Philosophy
at the
Massachusetts Institute of Technology

June 1989

Signature of Author [illegible]
Department of Mechanical Engineering
April 27, 1989

Certified by [illegible]
Professor Anthony T. Patera
Thesis Supervisor

Accepted by [illegible]
Professor Ain A. Sonin
Chairman, Department Committee on Graduate Studies

SPECTRAL ELEMENT SOLUTION OF THE NAVIER-STOKES EQUATIONS ON HIGH PERFORMANCE DISTRIBUTED-MEMORY PARALLEL PROCESSORS

by

PAUL FREDERICK FISCHER

Submitted to the Department of Mechanical Engineering on April 27, 1989, in partial fulfillment of the requirements for the Doctor of Philosophy in Mechanical Engineering

Abstract

We present a high-efficiency medium-grained parallel spectral element method for solution of the incompressible Navier-Stokes equations in general two- and three-dimensional domains. The method is based upon naturally concurrent Uzawa and Jacobi-preconditioned conjugate gradient iterative methods; data-parallel geometry-based distribution of work amongst processors; nearest neighbor sparsity and high-order substructuring for minimum communication; general heterogeneous locally-structured (vector) / globally-unstructured (parallel) constructs; and efficient embedding of vector reduction operations for inner product and norm calculations. An analysis is given for the computational complexity of the algorithm on a "native" model medium-grained parallel processor, and the intrinsic communication superiority of high-order discretizations is demonstrated.

The method is implemented on the (fast) Intel vector hypercube, and the performance of this algorithm-architecture coupling is evaluated in a technical and economic framework that reflects the true advantages of parallel solution of partial differential equations. We examine the potential economic benefits of distributed memory parallel processing by measuring the performance/price ratio versus actual optimized performance on several different computers for the solution of an 80,000 degree-of-freedom steady Stokes problem. Comparison of the performance of the 16-node Intel vector hypercube (44 MFLOPS) with that of a (single headed) Cray X/MP-12 (66 MFLOPS) clearly shows that parallel machines can achieve serial-supercomputer speeds at a fraction of serial-supercomputer costs. Parallel machines offer not only better economy, but also the potential of faster speeds; a 312,000 degree-of-freedom spectral element Stokes problem achieves 160 MFLOPS on a 64-node Intel vector hypercube.

Lastly, we remark that the locally-structured/globally-unstructured spectral element discretizations achieve good vector and parallel efficiency at little cost in generality. In particular, parallel solution of the full Navier-Stokes equations can now be considered to be of fluid mechanical, not only numerical, interest. We compare several Naiver-Stokes calculations with previous experimental and numerical results, including external flow past a cylinder at Re=1000, flow between co-rotating spheres, and horse-shoe vortex formation at the base of an end-mounted cylinder. The method is currently being used to investigate hydrodynamic stability phenomena in internal and external flows.

Thesis Supervisor: Professor Anthony T. Patera
Title: Associate Professor of Mechanical Engineering

Acknowledgements

I would like to acknowledge Prof. A. Patera who has brought together a critical mass of researchers to make this an exciting place to work, and who has shown never failing enthusiasm for the advancement of computational fluid dynamics. I also would like to thank Drs. E. Rønquist and E. Bullister for their continual assistance and insightful comments, G. Anagnostou with whom I have had many interesting conversations regarding general algorithm development, and Drs. C. Amon, L. Ho, G. Karniadakis, H. Kozlu, and C. Mavriplis for their collaborative efforts. Special acknowledgement goes to Dr. David Scott of Intel Scientific Computers who has had a significant impact on this work and without whose efforts many of the results presented herein would not be possible.

I would like to thank P. Bar-Yoseph, M. Cruz, and E. Renaud for their assistance in producing geometry and results for some of the sample calculations, and D. Zeritis for enthusiastic assistance with the nested dissection algorithm.

TABLE OF CONTENTS

ABSTRACT 2

ACKNOWLEDGEMENTS 4

TABLE OF CONTENTS 5

1. INTRODUCTION 7

2. ECONOMICS 11

3. SPECTRAL ELEMENT DISCRETIZATIONS 16
   - 3.1 Elliptic Problems 18
   - 3.2 Extension to the Navier-Stokes Equations 22

4. ITERATIVE SOLUTION PROCEDURES FOR ELLIPTIC PROBLEMS 30
   - 4.1 Evaluation of Spectral Element Operators 32
   - 4.2 Conjugate Gradient Iteration 33

5. PARALLEL SPECTRAL ELEMENT METHODS 35
   - 5.1 Conceptual Parallel Architecture 36
   - 5.2 Computational Complexity 40
   - 5.3 Comparison with *h*-Type Substructure Methods 51
   - 5.4 Architecture Mappings: Message-Passing Hypercube 54

6. IMPLEMENTATION OF PARALLEL CONSTRUCTS 57
   - 6.1 Distributed Memory Parallel Processing 57
   - 6.2 General Implementation 59
   - 6.3 Vector Reduction 61
   - 6.4 Direct Stiffness Summation 65
   - 6.5 Element Face-Matching 66
   - 6.6 Direct Stiffness Exchange Sequence 72

6.7 Special Node Treatment 76
6.8 Element-to-Processor Mapping 77

7. MEASURED PERFORMANCE ANALYSIS 78
7.1 Intel Hypercube Timings 78
7.2 Predicted Performance for Larger Systems 89
7.3 Economic Performance Analysis 93

8. PARALLEL NAVIER-STOKES COMPUTATIONS 97
8.1 Steady Stokes Flow 98
8.2 Rotating Flows 99
8.3 Natural Convection 103
8.4 Grooved Channel Flows 105
8.5 External Flow Past a Cylinder 108
8.6 Three-Dimensional Horseshoe Vortices 114

APPENDICES

A. Speed and Cost Data for Several Modern Computer Systems 123

B. Timing Results for 80,000 Degree-of-Freedom Stokes Problem 124

C. Isoparametric Discretizations in $\mathbf{R}^3$ 125

BIBLIOGRAPHY 130

Chapter 1
Introduction

Over the last several decades, computational fluid dynamics (CFD) has proven to be a feasible means of analysis for many problems of fundamental and applied interest in fluid mechanics. Yet, because of the mathematical difficulty of the problem and the complexity of the resulting solutions, little headway has been made in solving the governing Navier-Stokes equations for many commonplace engineering flows. In terms of the principal measure of difficulty of a given flow problem, the Reynolds number, several orders of magnitude separate the problems which are currently tractable and those which are representative of most engineering applications. As a consequence, continued emphasis is placed on the development of new algorithms. Although CFD comprises both algorithm development and the study of physics, the maturity of the discipline is such that advances in the study of fluid motion are typically the result of advances in computational algorithms.

Algorithm development in CFD is spurred on both by advancing computer architectures and by the changing, singular, nature of the governing equations as the spatial complexity and Reynolds number is increased. While progress in fluid mechanics analysis is influenced directly by increasing computer speeds, it would be wrong to conclude that the increasing output of CFD is inversely proportional to the precipitous clock cycle time of modern computers. The advances go far beyond that. Vector, and now parallel, architectures have spawned the reorganization of many antiquated codes to result in orders of magnitude reduction in problem solution times. The increasing mathematical difficulty of problems which can be addressed as a result of new found computational power necessitates new discretization techniques and solution algorithms. While low-order methods coupled with direct solvers are adequate for low Reynolds number problems in one- and two-dimensions, it is clear that high-order discretizations and iterative solvers have advantages for three-dimensional problems at higher, transitional Reynolds numbers [1]. As a result of such architecture and algorithm advances, solutions of simple geometry, two-dimensional laminar flows are now superseded by complex geometry, three-dimensional transitional and turbulent flow calculations.

The most dramatic change in computer architectures in the last decade has been the development of large scale parallel processors which achieve increased performance by assigning multiple computers to the execution of a single task. Such an approach offers the potential of increased speed at reduced cost, as a few (expensive) computers are replaced by many less powerfull (and less expensive) processors. The cost benefits derive from the simplicity of the architecture, the use of relatively low density VLSI technology produced in large volumes, and the fact that the CPU cost is not the dominant component of the total system cost. Unfortunately, the availability of parallel processors does not necessarily imply their efficient usage, and care must be taken in developing numerical algorithms that are appropriate for parallel implementation.

In this thesis we present a spectral element algorithm for the Navier-Stokes equations which exploits with high parallel efficiency the highly economical parallel computers currently available. The work builds extensively on past work on parallel partial differential equation solution in the choice of an iterative solver, as well as in the underlying strategy of load balancing, communication, and topological embeddings. In particular, the schemes are founded on the following well-developed precepts: use of iterative solvers that exploit sparsity and minimize non-concurrent operations, e.g. [2,3]; geometry-based distribution of work amongst processors, e.g. [4,5,6,7]; exploitation of nearest-neighbor sparsity and substructuring to minimize communication, e.g. [8,9]; efficient embedding of vector reduction operations to allow for more general and implicit solution algorithms, e.g. [10,11].

The methods used represent an extension of these well-estabished ideas in the following ways. First, the spectral element discretizations [12] employed are high-order, leading not only to improved accuracy but also to a more efficient, work-intensive "native" medium-grained parallelism. Second, the discretizations, solvers, and parallel constructs are built upon the general foundation of heterogeneous, locally-structured/globally-unstructured, representations, thus allowing for efficient implementation in arbitrary geometries. Third, the equations solved are the full Navier-Stokes equations describing viscous fluid flow [7], as opposed to (second-order elliptic) subsets of the Stokes problem; all potentially non-concurrent hazards are therefore addressed. Lastly, the methods are implemented on a fast vector parallel processor, thus providing a useful and economical fluid mechanics analysis tool.

The outline of this thesis is as follows. We review in Chapter 2 some of the economic concerns with numerical solution of the Navier-Stokes equations on advanced supercomputers. In Chapter 3 we introduce the spectral element discretizations for elliptic problems, and describe the extension of these discretizations to the Navier-Stokes equations. In Chapter 4 we present a representative iterative solution procedure for the elliptic problems. In Chapter 5 we present the spectral element solution approach for parallel architectures, and give theoretical performance estimates for performance of the method on various architectures. In Chapter 6 the parallel and vector implementation of the methods is described, both in terms of general software constructs and for the particular case of the Intel vector hypercubes. In Chapter 7 computational results and performance measures are presented which demonstrate that the potential advantages of parallel processing are in fact, realizable. In Chapter 8, numerous physical results are presented which illustrate the utility of parallel processing as a fluid mechanics analysis tool.

Chapter 2
Economics

The solution of incompressible fluid dynamics problems by numerical simulation has advanced rapidly in recent years due to simultaneous improvements in algorithms and computers. However, despite these advances, the large number of degrees-of-freedom required to resolve even relatively simple three-dimensional laminar flows, let alone transitional or turbulent flows, has prevented computational fluid dynamics from addressing many problems of fundamental and practical importance. In essence, large-scale fluid mechanics calculations are still too costly in terms of human and computational resources to assume the role of "primary means of analysis."

A promising approach to reducing the costly nature of fluid dynamics calculations is to solve problems not on a single (expensive) computer, but rather to distribute the work amongst many less powerful (and less expensive) processors. The potential increase in efficiency due to the economies of parallel processing derive not only from decreases in direct costs, but also from improvements in productivity and creativity brought about by a more local and interactive computing environment. Unfortunately, the availability of parallel processors does not necessarily imply their efficient usage, and care must be taken in developing numerical algorithms that are appropriate for parallel implementation.

Given the complexity of parallel computation as compared to its serial counterpart, it is imperative to verify that there is a sound economic basis for the notion that parallelism will lead to improved computational "efficiency". To this end, we briefly review an economic caricature of the costs associated with numerical simulation. The particular physical problem of interest is fixed, and the maximum error that can be tolerated in the numerical solution, ε, is specified. We then choose an algorithm and architecture/machine with which to solve the problem: the former is characterized by W, the number of floating point operations (in millions, say) required to attain the specified accuracy; the latter is characterized by the usual "fully-utilized/will-not-exceed" speed rating, MFLOPS, and a purchase cost, \$. The wall-clock time to perform the calculation is then given by $\tau = W/\hat{\eta}$ MFLOPS, and the direct computer costs are proportional to $C = W/(\hat{\eta}e)$. Here $\hat{\eta}$ is an algorithm-architecture efficiency parameter, and e = MFLOPS/\$ is a measure of the resource efficiency of a computer.

Although it is not appropriate in this context to introduce any particular cost function, it is clear that an unambiguous condition for reduction in cost (i.e., improvement in performance) is a simultaneous decrease in both the time to compute, τ, and the cost of the solution, C. From the relationships between (τ, C) and (W, MFLOPS, e) we conclude that any algorithm-architecture coupling that corresponds to a decrease in W, an increase in MFLOPS, and an increase in efficiency e constitutes a real increase in performance. There are two different avenues to improving performance. First, a better numerical algorithm can be be devised, corresponding to a decrease in operation count, W, at fixed accuracy; this decreased operation count may be achieved either through improvements in discretization or through improvements in solution method. Second, a "better" computer can be found, in which both the MFLOPS and resource-efficiency, e, are increased.

To illustrate more clearly the cost reduction due to computer performance, we plot in Figure 2.1 the (MFLOPS,e) operating points of several modern computers (the MFLOPS and \$ data is given in Appendix A). It follows from the arguments given above that for a fixed algorithm, and a fixed algorithm-architecture coupling, $\hat{\eta}$, a computer A is better than all computers B which are in the third quadrant with respect to computer A. (If we were to make the further requirement that an unambiguous cost improvement must be accompanied by a lower purchase cost, computer A will only be better than computers B in the lower half of the third quadrant.) It is seen from Figure 2.1 that supercomputers have made great strides in reducing τ, however they have had little impact as regards C; this is consistent with the fact that supercomputers are typically used only where the potential profit is large, and the analysis alternatives (e.g., experiment) are expensive.

In order to render the calculation of complex three-dimensional flows quotidian we will require significantly more resource-efficient computers. In fact, these machines now exist; within the past few years computers have emerged which are characterized by an efficiency $e = 1$ MFLOPS/\$10,000, corresponding to a resource-efficiency rating which is a full factor of ten better than the previous norm of $e = 1$ MFLOPS/\$100,000. This progress has been effected by basic hardware advances at the low MFLOPS limit of the $e = 1$ MFLOPS/\$10,000 curve, followed by parallel architecture advances which extend the performance envelope to the high-MFLOPS limit. In terms of the "quadrant of improvement" there now exist machines that represent clear improvements in performance over current mainframe and supercomputers alike.

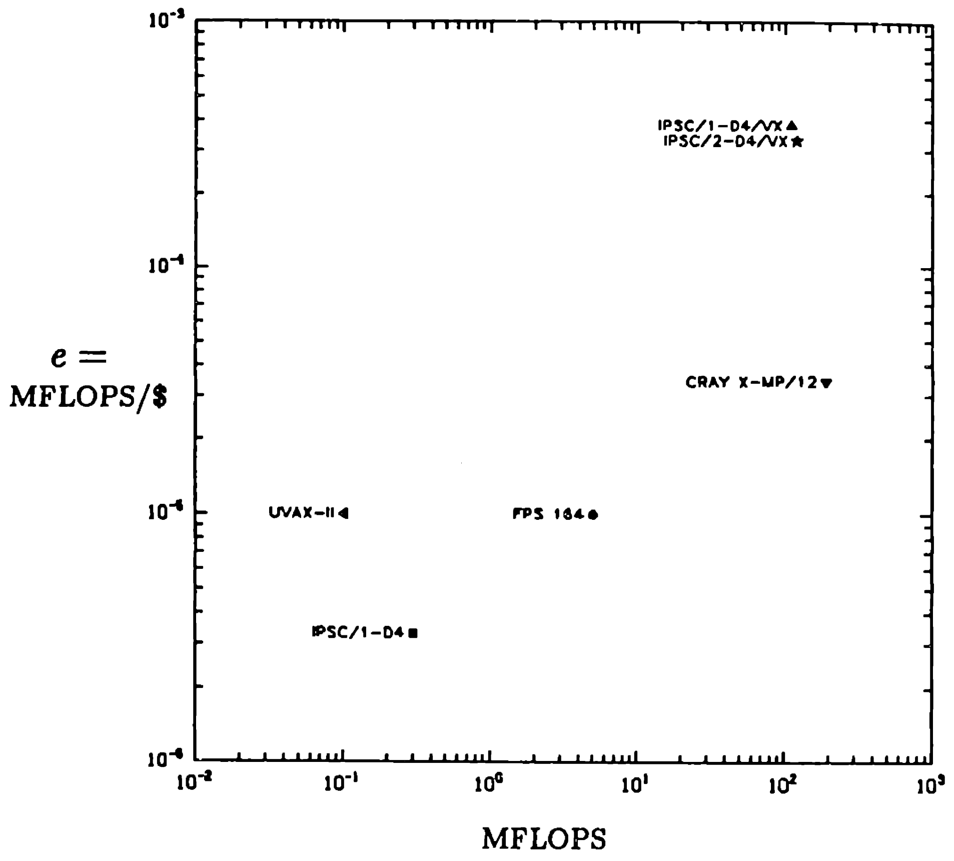


Figure 2.1: Operating points (peak theoretical) of several modern computers in MFLOPS-e space.

The fact that computer manufacturers are able to increase the number of processors in a system, M_{max}, with only a slightly faster-than-linear increase in cost is indicative of the fact that most of the high-MFLOPS, $e = 1$ MFLOPS/\$10,000 machines consist of processors which are largely independent, coupled by a rather sparse (albeit sophisticated) connection/routing network. The burden is thus placed on the algorithm to be sufficiently concurrent and "uncommunicative" to realize this ideal algorithm-independent performance; that is, the numerical algorithms must attain a sufficiently high algorithm-architecture efficiency $\hat{\eta}$ so as not to erode the savings in τ and C due to increases in MFLOPS and e. Note that if we ignore other architectural issues such as vectorization, $\hat{\eta}$ reduces to the usual definition of parallel efficiency, $\eta = S_\tau/M$, where M is the number of processors used in a calculation, and S_τ is the parallel speedup, defined as $S_\tau = \tau_1/\tau_M$.

Chapter 3
Spectral Element Discretizations

To address many problems in fluid mechanics, it is necessary to solve the fully viscous, incompressible, Navier-Stokes equations governing fluid motion at low Mach numbers:

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} &= -\nabla p + \frac{1}{Re} \nabla^2 \vec{u} - \vec{f} \quad in\,\Omega \\ \nabla . \vec{u} &= 0 \quad in\,\Omega \\ \vec{u} &= \vec{u}_{\partial\Omega} \quad on\,\partial\Omega, \end{aligned} \tag{3.1}$$

where $Re = UL/\nu$ is the Reynolds number based on a characteristic velocity, length, and viscosity. Examples include stability analysis, transitional and fully separated flows, heat transfer applications where boundary layer resolution is important, and direct simulation of turbulence in which convective effects determine the large scale motion while viscous effects dominate the smaller scales. In such instances, high-order techniques have advantages because of their good boundary layer resolution capabilities, their ability to accurately resolve a broad solution spectrum, and their minimal numerical dispersion and dissipation properties [13,14,15].

Traditionally, the arguments against the use of high-order discretizations are that they yield high-bandwidth, non-sparse, linear operators and hence require much more computational effort than their low-order counterparts, and that they do not provide sufficient geometric flexibility for most problems of engineering interest. However, as computational hardware performance continues to increase, it is becoming feasible to address fully three-dimensional problems, in which case it proves economical to employ iterative solvers for both low- and high-order methods. While the high-order operators are still full, it has been known for some time that the tensor-product operators associated with spectral methods can be factored into a series of sparse operators - resulting in operation counts which are competitive with low order methods [16]. In addition, at higher Reynolds numbers, high-order methods become more attractive because the correct, physical, dissipation can be obtained with fewer grid points in each spatial direction; the implication for three-dimensional calculations is a significantly reduced number of equations. As regards geometric flexibility, high-order spectral element and p-type finite element methods are capable of handling complex geometries by virtue of iso-parametric mappings and are an appropriate approach to discretization for problems in which the length scales of the solution are much smaller than those of the associated geometry, as is typically the case with fluid mechanics problems. Finally, in a parallel processing context, heterogeneous high-order, locally-structured/globally-unstructured, discretizations attain signficantly improved computation to communication ratios in comparison to low-order methods as will be shown in Chapter 5.

In this chapter we present a high-order spectral element discretization for solution of the incompressible Navier-Stokes equations. The spectral element method [12,17] is a generalized variational scheme which exploits the rapid convergence rates of spectral methods while retaining the geometric flexibility of the finite element techniques. It is based upon a macro- (spectral) element discretization in which the degrees-of-freedom internal to elements are coefficients of global (elemental) expansion functions with C^0 continuity imposed across element boundaries. With an appropriate choice of interpolants and quadrature formulae, it is can be proven that the error for problems having smooth solutions will decrease exponentially fast as the order of the expansion, N, is increased [17].

3.1 Elliptic problems

Our numerical methods for the Navier-Stokes equations are premised upon a 'layered' approach, in which the discretizations and solvers are constructed on the basis of a hierarchy of nested operators proceeding from the highest to the lowest derivatives. For incompressible viscous flow equations the linear self-adjoint elliptic Laplace operator represents the 'kernel' of our Navier-Stokes algorithm insofar that it involves the highest spatial derivatives. This operator governs the continuity requirements, conditioning and stability of the system. The fully discretized Navier-Stokes equations are typically solved at each time step by performing a series of elliptic solves and preconditioning steps.

We outline the basis of the discretization by considering the solution to a two-dimensional Poisson equation on the domain shown in Figure 3.1.

$$\begin{aligned} -\nabla^2 u &= f \quad in\ \Omega, \\ u &= 0 \quad on\ \partial\Omega. \end{aligned} \tag{3.2}$$

Equation (3.2) can be equivalently expressed as: Find $u \in \mathcal{H}_0^1(\Omega)$ such that

$$\int_\Omega \nabla\phi\nabla u\, d\Omega = \int_\Omega \phi f\, d\Omega \quad \forall\phi \in \mathcal{H}_0^1(\Omega), \tag{3.3}$$

where the space $\mathcal{H}_0^1$ is the space of all functions which are zero on the boundary and have a square integrable first derivative. The variational form has the significant advantage that it reduces the required level of continuity on the solution from C^1 to C^0, which in turn has implications as regards parallel communication.

Discretization of the variational statement proceeds by restricting the admissible solutions and trial functions in (3.3) to a finite-dimensional subspace, X_h, of the infinite-dimensional space, $\mathcal{H}_0^1$. For the spectral element method we choose the space X_h to be:

$$X_h \;=\; \{\phi|_{\Omega^k} \in \mathbf{P}_N(\Omega^k)\} \cap \mathcal{H}_0^1(\Omega) \tag{3.4}$$

where $\mathbf{P}_N(\Omega^k)$ is the space of all polynomials of degree $\leq N$ in each spatial direction on element k. The spectral element method is thus characterized by the discretization pair $h = (K, N)$, where K is the number of subdomains (elements) and N is the order of the polynomial approximations. For reasons of efficiency (tensor products, [16]) the subdomains are taken to be quadrilaterals in $\mathbf{R}^2$ and hexahedra in $\mathbf{R}^3$. The spectral element discretization of (3.2) thus corresponds to numerical quadrature of the variational form (3.3) restricted to

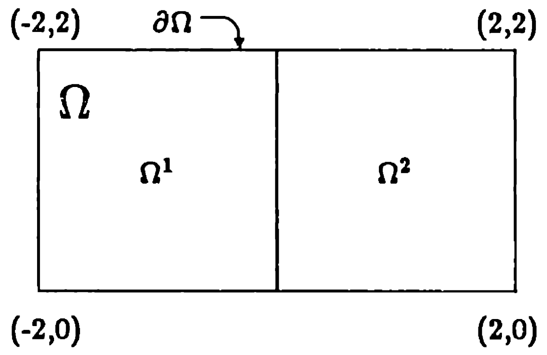


Figure 3.1: Computational domain Ω consisting of $K = 2$ subdomains.

the subspace X_h: Find $u_h \in X_h$ such that:

$$\int_\Omega \nabla\phi_h \nabla u_h \, d\Omega = \int_\Omega \phi_h f_h \, d\Omega \qquad \forall \phi_h \in X_h(\Omega), \tag{3.5}$$

where f_h is the interpolant of f in the space X_h.

While (3.5) is a statement of the type of solution which we seek, it does not indicate the form in which our solution will be represented, i.e., the choice of basis functions to be used for the polynomials in X_h. Traditionally, the availability of fast transforms have led to the use of Chebyshev polynomials as basis functions for spectral methods. However, the order of the polynomial used in spectral element decompositions is rarely large enough to benefit from fast transform techniques. In addition, it is desirable to have symmetric operators when using iterative solvers. This motivates the choice of Legendre based polynomials which result in symmetric operators due to their orthogonality with respect to a unity weighting function.

The present spectral element method employs a tensor product form of Lagrangian interpolants based on a local, elemental, mapping of $\mathbf{x} \in \Omega^k \rightarrow \mathbf{r} = (r, s) \in [-1, 1]^2$. We consider for illustration the case where the elemental decomposition consists of the union of squares with sides of length 2. Within each element, u_h has the form:

$$u_h(x, y)|_{\Omega^k} = \sum_{p=0}^{N}\sum_{q=0}^{N} u^k_{pq} h_p(r) h_q(s) \quad , \tag{3.6}$$

where $u^k_{pq} = u^k(r_p, s_q)$ are the unknown grid values of the approximate solution in element k. The interpolants, $h_i(\xi)$, satisfy:

$$\begin{aligned} h_i(\xi) &\in \mathbf{P}_N[-1, 1] \\ h_i(\xi_j) &= \delta_{ij} \quad , \end{aligned} \tag{3.7}$$

where the grid points, ξ_j, are chosen to be the Gauss-Lobatto-Legendre quadrature points [14,18]. The use of Lagrangian interpolants greatly increases the sparsity of the resultant system matrices. In particular, the mass matrix is diagonal.

With the explicit representation of functions given by (3.6), it is straight forward to evaluate the discrete variational form (3.5). Integration is performed using Gauss-Lobatto quadrature:

$$\int_\Omega f(x,y)d\Omega \;\rightarrow\; \sum_{k=1}^{K} \left\{ \sum_{i=0}^{N}\sum_{j=0}^{N} f_{ij}^k \rho_{ij} \right\} \quad ,$$

$$\rho_{ij} = \int_{-1}^{1} h_i(r)dr \int_{-1}^{1} h_j(s)ds \quad .$$

where ρ_{ij} is the quadrature weight associated with the nodal point $\mathbf{r}_{ij}$. The derivatives at the quadrature points are computed as:

$$\begin{aligned} \left.\frac{\partial u}{\partial x}\right|_{\mathbf{x}_{ij}^k} \;&\rightarrow\; \left.\frac{\partial u^k}{\partial r}\right|_{\mathbf{r}_{ij}} = (D_{ip}\, u_{pj}^k), \\ D_{ij} \;&\equiv\; \left.\frac{dh_j(r)}{dr}\right|_{r=r_i} \quad , \end{aligned} \tag{3.8}$$

where, for notational convenience in this and the following two equations, we use (.) to imply summation over the repeated index within the parentheses. The variational statement (3.5) therefore takes the form:

$$\sum_{\Omega^k} \left\{ \sum_{i=0}^{N}\sum_{j=0}^{N} \rho_{ij} \left[(D_{ip}\,\phi_{pj}^k)(D_{iq}\,u_{qj}^k) \;+\; (D_{jp}\,\phi_{ip}^k)(D_{jq}\,u_{iq}^k) \;-\; \phi_{ij}^k\, f_{ij}^k \right] \right\} = 0 \quad , \tag{3.9}$$

where it remains to specify what are the admissible values of ϕ_{ij}^k.

Since Eq. (3.9) holds for arbitrary $\phi_h \in X_h$, the requisite discrete system of equations can be generated by setting $\phi_{ij}^k = \delta_{ii'}\delta_{jj'}\delta_{kk'}$, for all (i', j', k') corresponding to unique points in the domain interior. Note that the outermost summation in equation (3.9) implies that in the case where ϕ^k_{ij} has a physical counterpart in another element, k', (i.e. ϕ^k_{ij} lies on the interface between elements k and k'), the contributions to the integral (sum) from the adjacent elements must be added together. We refer to this operation as direct stiffness summation and denote it by $\sum'_{\Omega^k}$. The final system to be solved for u is therefore:

$$\sum_{\Omega^k}{}' \left\{ \sum_{p=0}^{N} D_{pi}\,(D_{pq}u^k_{qj})\rho_{pj} + \sum_{p=0}^{N} D_{pj}\,(D_{pq}u^k_{iq})\rho_{ip} \right\} = \sum_{\Omega^k}{}' f^k_{ij}\rho_{ij} \qquad (3.10)$$

Although the discrete Laplacian in (3.10) is never evaluated in standard matrix form, it is convenient to express equation (3.10) as:

$$\mathbf{A\,u} = \mathbf{B\,f} \quad , \qquad (3.11)$$

where $\mathbf{A}$ is the global stiffness matrix and $\mathbf{B}$ is the mass matrix. The global vectors $\mathbf{u}$ and $\mathbf{f}$ represent the unknowns and data at all unique points $\vec{x}^k_{ij}$.

Although the discrete Laplacian for curvi-linear, three-dimensional geometries is more complex than that presented in (3.10), the basic derivation is the same. An efficient procedure for evaluating the general three-dimensional Laplacian is presented in Appendix C.

3.2 Extension to the Navier-Stokes Equations

Two alternative time advancement schemes are currently implemented for solution of the unsteady Navier-Stokes equations. The first of these is based upon a consistent choice of approximation spaces for the pressure and the velocity in order to ensure a well-posed formulation for the saddle Stokes problem [19-22]. The extension to Navier-Stokes follows by treating the nonlinear convective terms explicitly and solving the resultant Stokes problem at each time step with an Uzawa iterative procedure [1]. The second scheme uses the well known fractional step method [12,14] in which the *same* $\mathcal{H}^1$ approximation space is used for both the pressure and velocity. The splitting procedure is attractive as it is both accurate and efficient for sufficiently high Reynolds number flows: it is accurate because the discretization error due to inconsistent pressure boundary conditions decreases as the Reynolds number increases; it is efficient because it involves only the standard discrete $\mathcal{H}^1$ Laplace operator **A** (16), rather than the mixed $\mathcal{L}^2 - \mathcal{H}^1$ pressure operator that results from the use of consistent approximation spaces [18].

We begin by presenting the Uzawa algorithm for the steady Stokes problem and its extension to the Navier-Stokes equations. Consider the incompressible steady Stokes equations:

$$\begin{aligned} \nabla^2\vec{u} + \nabla p &= \vec{f} \;\; in\, \Omega \\ -\nabla.\vec{u} &= 0 \;\; in\, \Omega \end{aligned}$$

subject to boundary conditions

$$\vec{u} = 0 \;\; on\, \partial\Omega, \tag{3.12}$$

Following the variational procedures of the previous section, the discrete form of (3.12) in $\mathbf{R}^2$ is:

$$\begin{bmatrix} \mathbf{A} & & -\mathbf{D}_1^T \\ & \mathbf{A} & -\mathbf{D}_2^T \\ -\mathbf{D}_1 & -\mathbf{D}_2 & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{0} \end{Bmatrix} , \tag{3.13}$$

where **A** is the discrete Laplacian of equation (3.11) and $\mathbf{D}_i$ is the derivative operator associated with the ith direction. To honor the the Brezzi-Babuska-Ladyzenskaya (inf-sup) condition, the function space for p and the associated trial functions are in $\mathbf{P}_{N-2}(\Omega^k)$ rather than $\mathbf{P}_N$, and are based upon Gauss rather than Gauss-Lobatto quadrature points which are used for the velocity [1,17]. This consistent formulation has the significant feature that the approximation for the pressure does not extend to the element boundaries. Hence, as is physically the case, no apriori pressure boundary conditions are required when solving (3.13).

The Uzawa solution scheme for the Stokes problem is as follows. Block Gaussian elimination is performed on (3.13) to decouple the pressure and velocity into $d+1$ systems of equations. The pressure can be solved directly from:

$$\mathbf{Sp} = \sum_{i=1}^{d} \mathbf{D}_i \mathbf{A}^{-1} \mathbf{f}_i \quad , \tag{3.14}$$

where

$$\mathbf{S} = \sum_{i=1}^{d} \mathbf{D}_i \mathbf{A}^{-1} \mathbf{D}_i^{\mathbf{T}} \quad . \tag{3.15}$$

The matrix $\mathbf{S}$ is full, positive-definite, symmetric, and, when preconditioned by the mass matrix $\mathbf{B}$, of near unity condition, so iterative procedures such as conjugate gradient methods are an appropriate means of solving the system (3.14). It does imply that the $\mathbf{A}$ system be solved d times at each iteration; these also are solved iteratively using techniques described in the next section. Once the pressure is known, the d velocity components can be computed from (3.13) with d additional $\mathbf{A}$ solves.

We next consider the Uzawa algorithm applied to the full unsteady Navier-Stokes equations in non-dimensional form:

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} - \frac{1}{Re}\nabla^2 \vec{u} + \nabla p &= -\vec{u}\cdot\nabla\vec{u} + \vec{f} \ \ in\, \Omega \\ -\nabla.\vec{u} &= 0 \ \ in\, \Omega \\ \vec{u} &= 0 \ \ on\, \partial\Omega, \end{aligned} \tag{3.16}$$

where $Re = UL/\nu$ is the Reynolds number based on a characteristic velocity, length, and viscosity. The similarity between (3.16) and the Stokes problem (3.14) is apparent. We discretize (3.16) in time by treating the viscous term $\frac{1}{Re}\nabla^2\vec{u}$ implicitly, and the nonlinear convective term explicitly. The divergence free constraint is imposed at each time step, so the pressure must also be computed implicitly, using the Uzawa algorithm as before. The discrete system for the velocity and pressure at time step $n+1$ is:

$$\begin{bmatrix} \frac{1}{Re}\mathbf{A} + \frac{1}{\Delta t}\mathbf{B} & & -\mathbf{D}_1^T \\ & \frac{1}{Re}\mathbf{A} + \frac{1}{\Delta t}\mathbf{B} & -\mathbf{D}_2^T \\ -\mathbf{D}_1 & -\mathbf{D}_2 & 0 \end{bmatrix} \begin{Bmatrix} \mathbf{u}_1{}^{n+1} \\ \mathbf{u}_2{}^{n+1} \\ \mathbf{p}^{n+1} \end{Bmatrix} = \begin{Bmatrix} \frac{1}{\Delta t}\mathbf{B}(\mathbf{u}_1{}^n + AB_1^n) \\ \frac{1}{\Delta t}\mathbf{B}(\mathbf{u}_2{}^n + AB_2^n) \\ 0 \end{Bmatrix} \tag{3.17}$$

where,

$$AB_i^n \equiv \sum_{q=0}^{2} \alpha_q (\vec{u}^{n-q} \cdot \nabla \vec{u}^{n-q} - \vec{f}^{n-q}) \cdot \hat{e}_i \quad i = 1,2,...,d \tag{3.18}$$

$$\alpha_0 = 23/12 \qquad \alpha_1 = -16/12 \qquad \alpha_2 = 5/12 \quad .$$

Although the above scheme is only first order accurate in time, the third order Adams-Bashforth treatment of the convective operator is used because of the relatively large portion of the imaginary axis which lies within its stability region [23].

Solution of (3.17) proceeds in the same manner as for the Stokes problem. At each time step we solve:

$$\mathbf{E}\mathbf{p} = -\left(\frac{1}{Re}\mathbf{A} + \frac{1}{\Delta t}\mathbf{B}\right)^{-1} \mathbf{D}_i^T AB_i^n \quad , \tag{3.19}$$

where

$$\mathbf{E} = \sum_{i=1}^{d} \mathbf{D}_i \left(\frac{1}{Re}\mathbf{A} + \frac{1}{\Delta t}\mathbf{B}\right)^{-1} \mathbf{D}_i^T \quad . \tag{3.20}$$

The velocity is computed with d additional solves of the elliptic Helmholtz operator, $\left(\frac{1}{Re}\mathbf{A} + \frac{1}{\Delta t}\mathbf{B}\right)$.

Unfortunately, whereas $\mathbf{S}$ is very well conditioned, $\mathbf{E}$ is very ill-conditioned, implying slow convergence in the outer conjugate gradient iteration. As a result, the splitting method presented below has been the preferred solution scheme for high Reynolds number calculations. However, a preconditioner based upon additional $\mathbf{A}$ solves has been recently developed by Rønquist [24] which makes the Uzawa algorithm more attractive due to its superior accuracy.

The splitting formulation, or fractional-step method [25-27] is comprised of three computational steps. Starting with explicit treatment of the nonlinear step using the third-order Adams-Bashforth method, compute $\hat{u}$:

$$\frac{\hat{u} - \vec{u}^n}{\Delta t} = \vec{AB}^n \quad . \tag{3.21}$$

This is followed by the pressure step:

$$\begin{aligned} \nabla^2 p &= \frac{1}{\Delta t}\nabla \cdot \hat{u} \quad in\ \Omega \\ \frac{\hat{\hat{u}} - \hat{u}}{\Delta t} &= -\nabla p \quad in\ \Omega \\ \nabla p \cdot \hat{n} &= \frac{1}{\Delta t}\hat{u} \cdot \hat{n} \qquad on\ \partial\Omega. \end{aligned} \tag{3.22}$$

Finally, we compute the implicit viscous correction:

$$\begin{aligned} \frac{\vec{u}^{n+1} - \hat{\hat{u}}}{\Delta t} &= \frac{1}{Re}\nabla^2 \vec{u} \\ \vec{u}^{n+1} &= \vec{u_b}(t) \quad on\ \partial\Omega \quad . \end{aligned} \tag{3.23}$$

Here the $\vec{AB}_n$ is the third-order Adams-Bashforth discretization of (3.18).

Of the three steps, the elliptic solve for the pressure (3.22) is the most time consuming. Care must be exercised when formulating (3.22) to ensure proper treatment of the boundary conditions. If the velocity is specified everywhere on the domain boundary, $\partial\Omega$, the pressure equation is a pure Neumann problem having an unspecified degree-of-freedom corresponding to the average pressure in the domain. When using iterative solvers, this extra degree-of-freedom can be eliminated by first orthogonalizing the right-hand-side with respect to a constant (discrete) vector. The resultant solution vector, comprised of linear combination of eigenvectors of the discrete operator and the right-hand-side, will also be orthogonal to the constant vector and hence have zero average. If a Neumann, or outflow, condition is specified for the normal component of the velocity on some fraction of the domain boundary, $\partial\Omega_o$, then Dirichlet conditions are specified for the pressure on $\partial\Omega_o$.

We close this section by illustrating the exponential convergence rate attainable with spectral methods. Kovasznay gives an analytical solution to the Navier-Stokes equations which is similar to the two-dimensional flow field behind a periodic array of cylinders [28]:

$$u = 1 - e^{-\lambda x}\cos(2\pi y) \tag{3.24}$$

$$v = \frac{\lambda}{2\pi} e^{-\lambda x}\sin(2\pi y) \tag{3.25}$$

$$\lambda \equiv \frac{1}{2}Re \pm \sqrt{\frac{1}{4}Re^2 + 4\pi^2}\quad , \tag{3.26}$$

where Re is the Reynolds number based on the mean flow velocity and separation between the vortices. We solve this steady state boundary value problem for the case $\lambda = \frac{1}{2}Re + \sqrt{\frac{1}{4}Re^2 - 4\pi^2}$, Re=40, by marching in time from an initial state: $\vec{u}_h = 0$ in Ω, $\vec{u}_h = \vec{u}$ on $\partial\Omega$, where $\vec{u}_h$ and $\vec{u}$ are taken to be the spectral element and analytical solutions, respectively. The streamlines are shown in Figure 3.2.

We compare the spectral element solution, $\vec{u}_h$, with the analytical solution, $\vec{u}$, by plotting $||\vec{u} - \vec{u}_h||_{\mathcal{H}^1,\mathcal{L}^2}$ for K=8 and varying N in Figure 3.3. Exponential

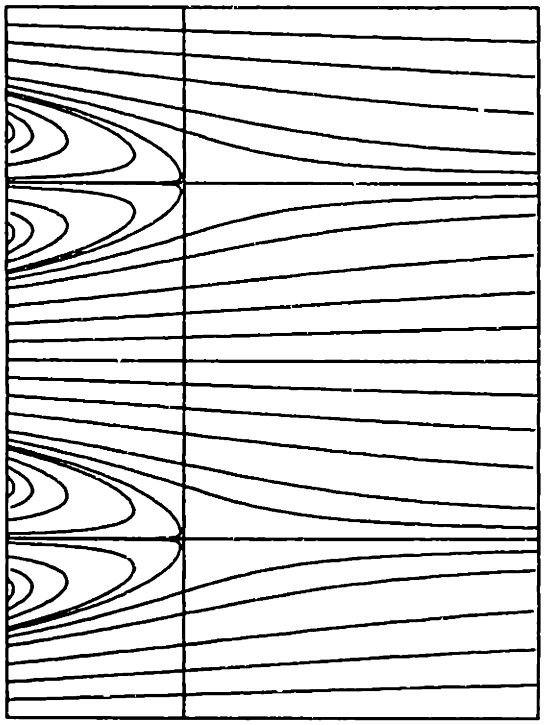


Figure 3.2:. Streamlines for Kovasznay's case. The mean flow is from left to right. Reynolds number based on vortex separation, mean flow speed, and kinematic viscosity is $Re = 40$. The domain is subdivided into $K = 8$ elements.

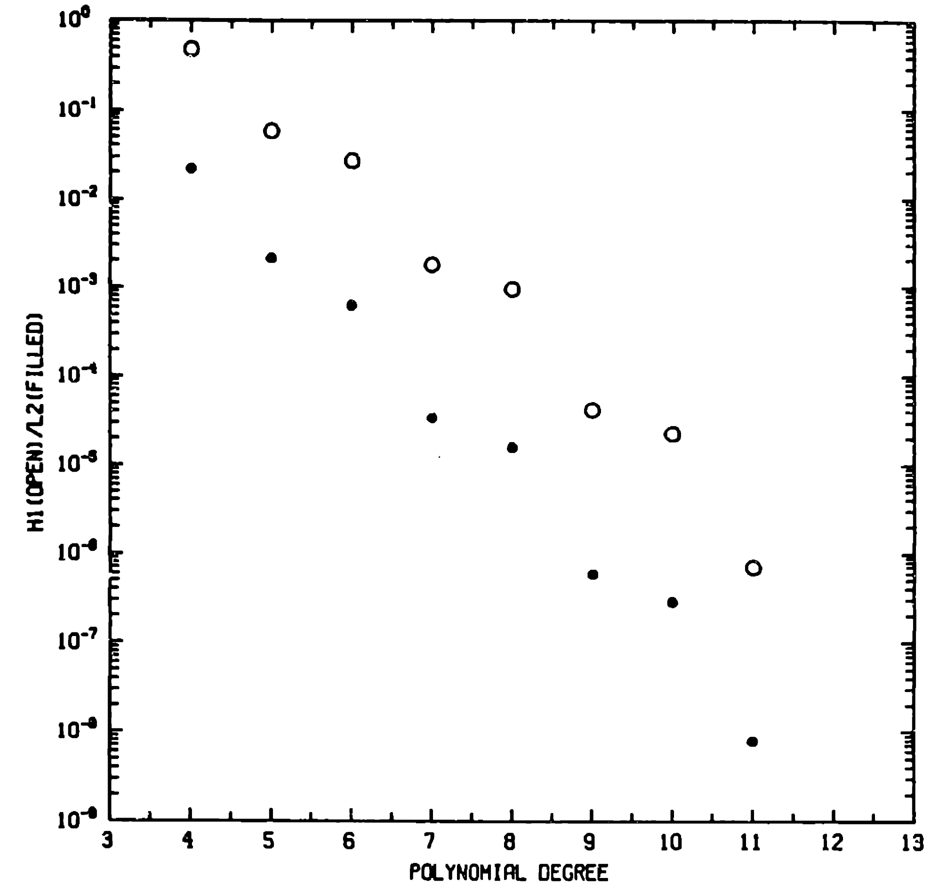


Figure 3.3: Spectral element convergence $||\vec{u} - \vec{u}_h||$ for the problem of Figure 3.2. The graph depicts the $\mathcal{H}^1$ and $\mathcal{L}^2$ norms for the Uzawa solution algorithm. (Courtesy E.M. Rønquist)

convergence in the velocity is evident for both norms. Rønquist has also shown that the pressure converges exponentially for the Uzawa algorithm but not for the splitting scheme [24].

Chapter 4
Iterative Solution Procedures for Elliptic Problems

It is clear from the previous chapter that both the Uzawa and the fractional-step formulations of the Navier-Stokes problem rely heavily upon the ability to perform fast A-solves. For this reason, our efforts have to a large degree been concentrated on the solution of elliptic problems such as the Poisson as Helmholtz equations. The choice of a solution scheme is dependent upon many factors - the sparsity and bandwidth of the linear system of equations, whether the system is positive definite or not, whether the coefficients are constant or varying in time, the architecture of the computer upon which the system of equations is to be solved - all these have a strong influence on the type of solution scheme used. Our targeted class of problems are large, three-dimensional elliptic equations which yield large bandwidth, sparse, systems of equations. Care is taken in the discretization through the choice of a variational formulation to ensure that the system is positive definite. Because the systems are large, fast (concurrent) computers are required. Under such conditions, iterative methods are an appropriate means of solving the discretized elliptic problems. The choice of iterative solution schemes is motivated by three issues: memory, operation count, and concurrency. The first of these alone is sufficient reason to choose iterative solvers over direct methods, particularly for three-dimensional problems.

Consider a problem of general deformed geometry in $\mathbf{R}^d$. The discrete Laplacian in (3.10) couples all points local to a given spectral element which implies that, if **A** is explicitly computed and stored, there will be at least N^d entries on each row. Given that there are KN^d unknowns, the total storage would be $O(KN^{2d})$, just to generate the matrix. If we take an example where $d = 3$, $N = 10$, and five elements are used in each spatial direction, $K = 125$, the minimum required storage would be 125 Mwords for the **A** matrix alone. Such a problem would either have to be solved out of core, or would have to be solved on one of the few, large-memory, supercomputers such as the Cray-2.

Following similar arguments, it has been demonstrated that iterative schemes have the potential of being faster than direct methods [29] as the amount of work in factoring the **A** matrix is certainly greater than KN^{2d}. Finally, the advent of parallel processing has brought renewed interest in iterative procedures because such methods are naturally concurrent. Direct methods have the characteristic of being inherently sequential; one methodically eliminates in the forward direction, row after row, then performs the backsolve, line by line. By contrast, iterative methods have no characteristic of forward and back, and no notion of bandwidth. There are simply KN^d vector inner-procducts to be computed at each iteration, each more or less independent of the other.

In this section we outline the basis of one particular iterative spectral element solver, conjugate gradient iteration, and analyse the associated computational complexity for a serial processor implementation. Extensions to parallel processor implementations are discussed in Chapter 5.

4.1 Evaluation of Spectral Element Operators

At the heart of any iterative solver is the evaluation of matrix-vector products of the form **Au**. It is clear from equation (3.10) that for discretizations in $\mathbf{R}^d$ these products can be efficiently calculated in $O(KN^{d+1})$ operations using sum-factorization methods [16]. Additional computational efficiency can be gained by exploiting the regular structure of the spectral (intra-element) operators and recognizing that the inner most kernel of the matrix-vector product, $(D_{ip}\, u_{pj})$, is exactly equivalent to a *matrix-matrix* product which can be rapidly evaluated on many vector architectures. In addition to the $O(KN^{d+1})$ computational effort, the direct stiffness summation will require $O(KN^{d-1})$ operations, corresponding to the number of degrees-of-freedom lying on the element interfaces. It thus follows that the number of clock cycles required to evaluate the left side of (3.10) on a single processor is

$$Z_1^{e'} = c_1 K N^{d+1} + c_2' K N^d + c_3 K N^{d-1}, \tag{4.1}$$

where the constants c_1, c_2', and c_3 depend only (weakly) on spatial dimension. The $O(KN^d)$ contribution to $Z_1^{e'}$ is only present in the case of complex geometry or non-separable coefficients. It should also be noted that only $O(KN^d)$ storage is required to evaluate **Au**.

The proper choice of spectral element basis is directly reflected in the "good" computational complexity estimate $Z_1^{e'}$. First, the sum-factorization (3.10) and the operation count (4.1) applies to general-geometry isoparametric spectral element discretizations of non-separable equations [17], due to the tensor product spaces (3.4), tensor product quadratures (3.8), and tensor product bases (3.6) described in Section 3.1. Second, the direct stiffness summation contribution to Z_1^{el} is only $O(KN^{d-1})$, rather than $O(KN^{d+1})$, due to our choice of basis in which the number of test functions which are nonzero on the elemental boundary is minimal. Although the fact that the direct stiffness summation work is small does not appear particularly important in the single-processor estimate (4.1), in the parallel case the direct stiffness contribution will be the leading-order *communication* term.

4.2 Conjugate Gradient Iteration

We next consider simple Jacobi (diagonal)- preconditioned conjugate-gradient iterative solution [30] of the multi-dimensional elliptic equation (21). The conjugate gradient algorithm is given by,

$$\begin{aligned}
\mathbf{r}_0 &\equiv \mathbf{Bf} - \mathbf{Au}_0 \qquad (4.2)\\
\textit{while } |\mathbf{r}^T\mathbf{Br}| &> \epsilon,\\
\tilde{\mathbf{p}}_k &= \tilde{\mathbf{A}}^{-1}\mathbf{r}_{k-1}\\
\rho_k &= \mathbf{r}_{k-1}\cdot\tilde{\mathbf{p}}_k\\
\beta_k &= \rho_k/\rho_{k-1}\\
\mathbf{p}_k &= \beta_k\mathbf{p}_k + \tilde{\mathbf{p}}_k\\
\mathbf{w}_k &= \mathbf{Ap}_k\\
\alpha_k &= \rho_k/(\mathbf{p}_k\cdot\mathbf{w}_k)\\
\mathbf{u}_k &= \mathbf{u}_{k-1} + \alpha_k\mathbf{p}_k\\
\mathbf{r}_k &= \mathbf{r}_{k-1} - \alpha_k\mathbf{p}_k
\end{aligned}$$

where $\mathbf{u}_0$ is an initial guess for the solution $\mathbf{u}$. Note that the diagonal preconditioner, $\tilde{\mathbf{A}} = \text{diag}(\mathbf{A})$, can be formed without constructing the entire $\mathbf{A}$

operator.

From (4.2) we see that, per iteration, the conjugate gradient scheme requires: one matrix-vector evaluation - $Z_1^{e'}$ cycles; several local collocation operations - $O(KN^d)$ cycles; and two inner products - $O(KN^d)$ cycles. If we denote by N_ϵ^A the number of iterations required to bring the error in the solution down to $O(\epsilon)$ in some appropriate norm [31,32], the number of clock cycles for conjugate gradient solution of (3.10) is:

$$Z_1^e = N_\epsilon^A \left\{ c_1 KN^{d+1} + c_2 KN^d + c_3 KN^{d-1} \right\} \quad , \tag{4.3}$$

where c_2 accounts for the additional conjugate gradient operations.

Although in the evaluation of parallel performance in Chapter 5 the number of iterations, N_ϵ^A, will scale out, it is nevertheless appropriate to comment on the number of iterations required to achieve convergence. It can be shown that the condition number of the preconditioned $\mathbf{A}$ system is $O(K_1^2 N^2)$ [33], which implies that $N_\epsilon^A \sim O(K_1 N)$ [30], where K_1 is the number of spectral elements in a *single* spatial direction. This convergence rate, though respectable for a high-order method, is clearly not order-independent, and will deteriorate significantly for large problems. The convergence rate can be improved by the use of recently developed spectral element multigrid algorithms [33,34].

Chapter 5
Parallel Spectral Element Methods

The development of any numerical algorithm entails a thought process in which one considers the scope of the problem, the class of problems to be addressed, and the machine architecture for which the algorithm is ultimately targeted. Given the capability of today's desk top computer workstations, architectural issues are not of great concern for an increasing number of problems of relatively small size or limited scope, as such problems can be solved in a time frame which is acceptable to the end user. However, even at the workstation level, high-end performance is only obtained through exploitation of basic vectorization techniques - so the fundametals of vector architectures must be understood. For larger problems, such as three-dimensional Navier-Stokes calculations, performance demands are well beyond readily available processing capabilities, so algorithms must be developed which exploit advanced architectures in order to render many difficult problems tractable. It is clear that future generations of supercomputers will rely more and more on large scale parallelism and that future high-performance software will necessarily incorporate parallel algorithm constructs, so it is of value to develop an understanding of concurrent computing and the associated algorithmic concerns.

Our code development has followed a top down process in which architecture independent issues such as discretizations and solvers are addressed first [1,35,36]; followed by identification of architecture dependent constructs regarding vectorization, concurrency, and interprocessor communication; and ultimately, by machine specific implementation. There is of couse feedback in the development process via iterations at several levels. For instance, the development of the solvers is not wholly independent of the architecture in that care is taken from the beginning to ensure that concurrency is maintained at all levels. The actual parallel implementation is comprised of two steps. We first postulate a model architecture to permit analysis of the computational complexity associated with our particular algorithm/architecture coupling and to develop general parallel constructs in a machine-independent context. We then address the actual implementation details for a particular machine. Viewing the parallel implementation as a two step process promotes greater generality and portability of the resultant computer program. In this chapter we outline the implementation of the spectral element discretizations and solvers on a conceptual parallel processor and examine the associated computational complexity for both high and low order discretizations. Detailed implementation issues are presented in the following chapter.

5.1 Conceptual Parallel Architecture

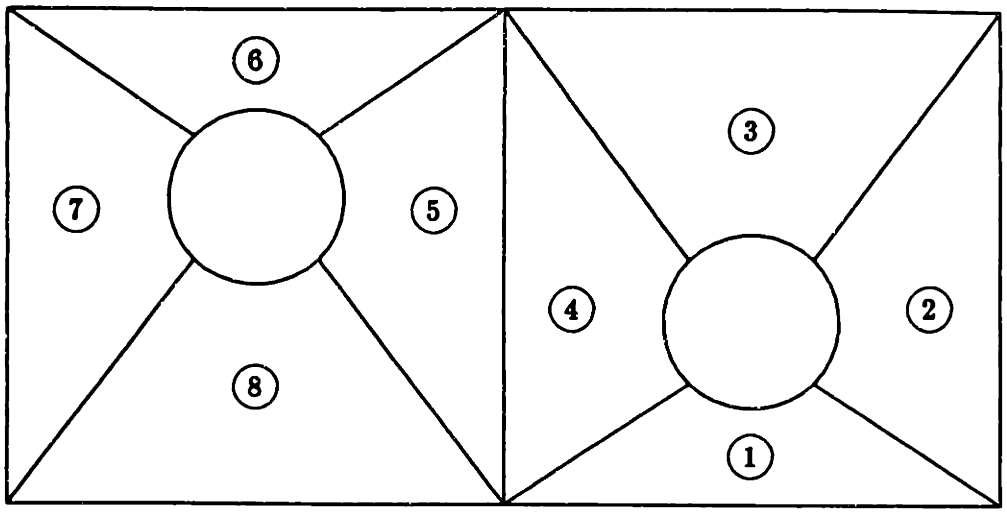
Our discretizations, algorithms, and data structures are constructed so as to admit a natural, element-based, parallel work decomposition, in which each spectral element (or group of spectral elements) is mapped to a separate processor/memory, with the individual processor/memory units being linked by a relatively sparse communications network. This conceptual architecture is naturally suited to the spectral element discretization in that it provides for tight, structured coupling within the dense elemental constructs, while simultaneously maintaining generality and concurrency at the level of the unstructured macro-element skeleton. The locally structured/globally unstructured spectral element parallel paradigm is closely related to the concept of domain-decomposition by substructured finite elements [37,5,6] and many of our results are generic to both computational models. This latter point will be discussed in greater detail in Section 5.3.

To illustrate the issues associated with parallel spectral element solution techniques, we consider the performance of the conceptual medium-grained parallel processor shown in Figure 5.1, in which K spectral elements are partitioned amongst $M \leq K$ independent processor/memory units. (Our terminology will be two dimensional, however the methods readily extend to three space dimensions, as will be demonstrated by examples in Chapter 8.) In essence, each processor contains a subset of the entire domain, comprised of several elements. We assume load balance in the sense that all processors have an equal number of elements. The communications network of the model parallel processor is assumed to satisfy two constraints:

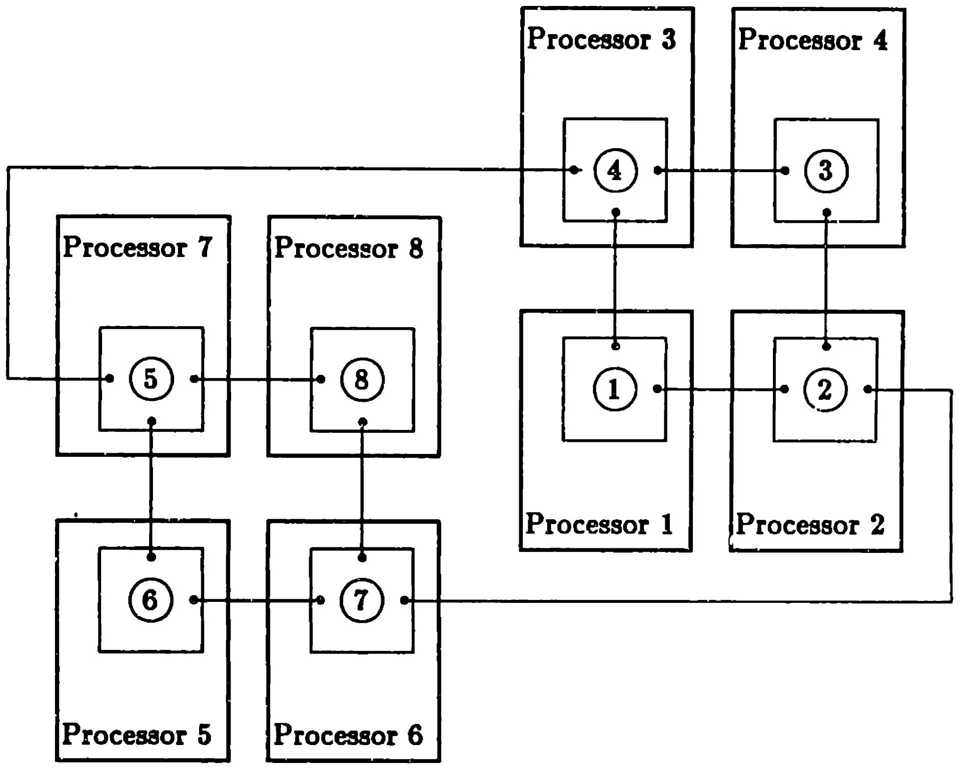
A distinct, direct link must exist between two processors for each distinct pair of adjacent elements which is divided between the processors. (5.1)

A summation of M values distributed over M processors can be performed in $O(\log M)$ communication steps. (5.2)

These two requirements relate directly to the two communication constructs central to our algorithm, direct stiffness summation and vector reduction, respectively. (Note, all logarithms are taken to be base 2.)



(a)



(b)

Figure 5.1: (a) Spectral element decomposition for a multiply connected domain with element numbers denoted by ⊙; (b) "native" model parallel processor.

We characterize the "hardware" associated with the processors and communication networks in Figure 5.1 by a basic clock cycle for calculation, δ, and the time-per-word required to send n words across a direct link, $\Delta(n)$. It is assumed that data transfer can occur simultaneously over all distinct links. The ratio Δ/δ is denoted $\sigma(n)$; $\sigma(n)$ is assumed to be a decreasing function of n, with $\sigma(1)$ appreciably greater than $\sigma(\infty)$ due to message startup overhead. Messages travelling more than one link (or "hop") can be penalized in terms of both longer transmission time and potential contention. (Contention represents network imbalance/saturation, and arises when more than one potentially parallel communication requires the same link.)

We emphasize that our model architecture is inherently (though not restricted to) a distributed memory system in recognition of the fact that the majority of the computational effort and hence, memory access, is associatied with *intra*-element calculations. Since memory access time is frequently a rate limiting step on vector architectures it is critical that, as the number of elements and processors grows, the primary memory bandwidth increases proportionally. One easy way to ensure this is by having a tight, one-to-one, correspondence between each processor and a uniquely assigned local memory. Such an architecture allows for concurrent memory access as well as concurrent processing.

If no real machines were similar to the hypothetical architecture described above, the resulting analysis would, of course, be of fairly limited value. However, there are many architectures which are identical to, or at least very similar to, the conceptual architecture of Figure 5.1. In particular, reconfigurable lattices readily satisfy constraints (37), and lattice or hypercube message-passing architectures satisfy all constraints save the assurance of nearest neighbor, contention-free communication. We will discuss these mapping issues and other real-world corrections in the following chapter once the performance of our algorithm on the basic model of Figure 5.1 is understood.

5.2 Computational Complexity

We consider here M-parallel solution of the elliptic problem (3.2) by conjugate gradient iteration (4.2). As described in Chapter 4, the performance of the conjugate gradient iteration is determined by the following representative computational kernels:

$$\mathbf{r} = \mathbf{A}\mathbf{u} \tag{5.3}$$

$$\mathbf{r} = \tilde{\mathbf{A}}\mathbf{u} \tag{5.4}$$

$$\alpha = \mathbf{r}\cdot\mathbf{r} \quad , \tag{5.5}$$

corresponding to operator evalution, diagonal-matrix collocation, and norm (or inner product) calculation, respectively. We now discuss how each of these operations is performed in parallel, and present computational complexity estimates [9] for the resulting algorithms .

We begin with the evaluation of a representative term of $\mathbf{r} = \mathbf{A}\mathbf{u}$, (3.11), which, by construction, admits the following simple concurrency. First, we calculate an "incomplete" residual $\tilde{r}^k \equiv A^k u^k$ in each element separately and concurrently. For our model problem of Figure 3.1, this would be:

$$\tilde{r}^k_{ij} = \sum_{p=0}^{N} D_{pi}\,(D_{pq}u^k_{qj})\rho_{pj} + \sum_{p=0}^{N} D_{pj}\,(D_{pq}u^k_{iq})\rho_{ip} \;\; \forall i,j \in \{1,2,...,N\}^2 \tag{5.6}$$

(This residual is incomplete in the sense that the element-boundary-node displacements are not admissible; direct stiffness is the process by which appropriate contributions are added from neighboring-element test functions.) This operation is communication-free, and will require $c_1 K N^{d+1}/M$ clock cycles, where c_1 is defined by (4.1). Next, we perform direct stiffness summation of the edge residuals to account for contributions from neighboring elements:

$$r_{ij}^k = \sum_{\Omega^k}{}' \tilde{r}_{ij}^k \quad , \tag{5.7}$$

where the direct stiffness procedure is described in Figure 5.2. The flow of information at vertices in Figure 5.1 is clearly not coincident with the possible single-hop flow of information along links in our parallel processor described by Figure 5.1b, as the variational formulation requires that contributions from each element be summed together at a shared vertex. However, an efficient direct stiffness procedure based on nearest-neighbor use of the edge-based communication network can be constructed, as we now describe.

We first consider the simple spectral element mesh shown in Figure 5.2a in which data is given on each element. In Figures 5.2c through 5.2f we show diagrammatically how direct stiffness summation can be performed by local directional splitting of the operation into a sequence ψ of d=2 element exchanges. It can be seen that the nodal values at the vertices are, indeed, correct; that is the sequence ψ ensures that the contributions from each element at the shared corner are summed together. The advantages of this splitting method over, say, a bi-directional parallel edge pass followed by vertex-specific operations are: the splitting method is algorithmically clean; the splitting method avoids costly short messages; the splitting method avoids non-nearest-neighbor communication and contention. These advantages are even clearer in three space dimensions.

For many spectral (or substructured finite) element decompositions it is difficult, if not impossible, to find an edge pass sequence ψ which results in correct nodal values at all vertices. However, it is often relatively simple to find a sequence ψ for which the number of vertices with incorrect values, denoted "special" nodes, is small. This suggests the following strategy: first, a vector

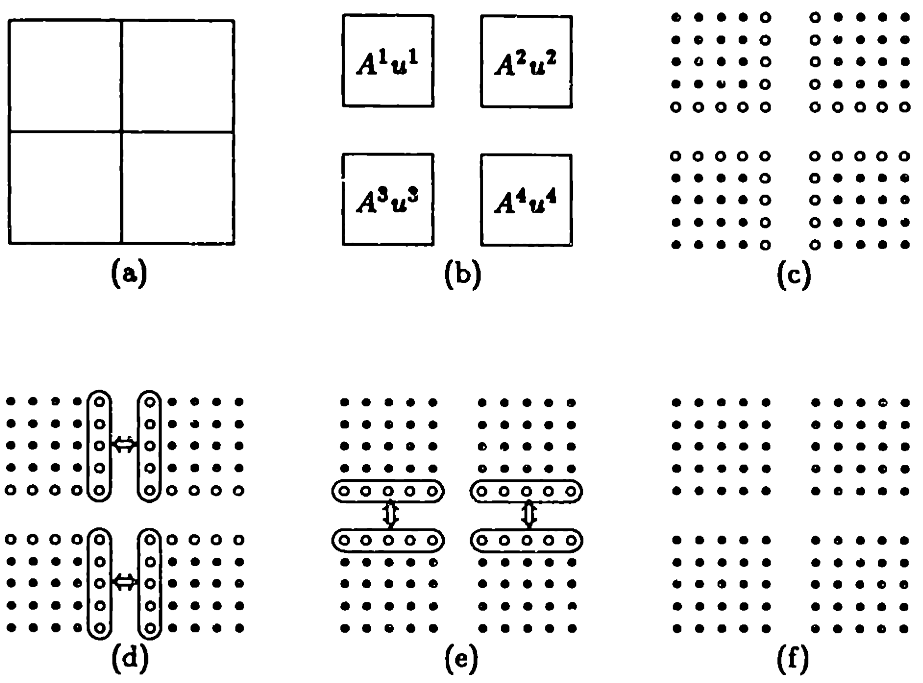


Figure 5.2: Computation of residual vector for regular geometry in $\mathbb{R}^2$: (a) four element mesh; (b) simultaneous (parallel) computation of incomplete residual, $\tilde{r}^k = A^k\tilde{u}^k$; (c) nodal content of $\tilde{r}^k$ is denoted by circles - solid circles indicate correct residuals, open circles indicate values requiring contributions from neighboring elements; (d) and (e) bi-directional exchange and sum sequence, ψ, which effects the completed residual at all nodes including the corner nodes; (f) completed residual, $\tilde{r}^k = \sum' A^k\tilde{u}^k$.

reduction (sum) operation is performed to accumulate the contributions of all elments to the residuals associated wtih the special nodes; second, a standard d-pass sequence is performed, ψ; third the results of the vector reduction are redistributed to the special nodes. The summaton strategy is illustrated diagrammatically in Figure 5.3.

If we (suggestively) denote the number of special nodes by ϵ, the number

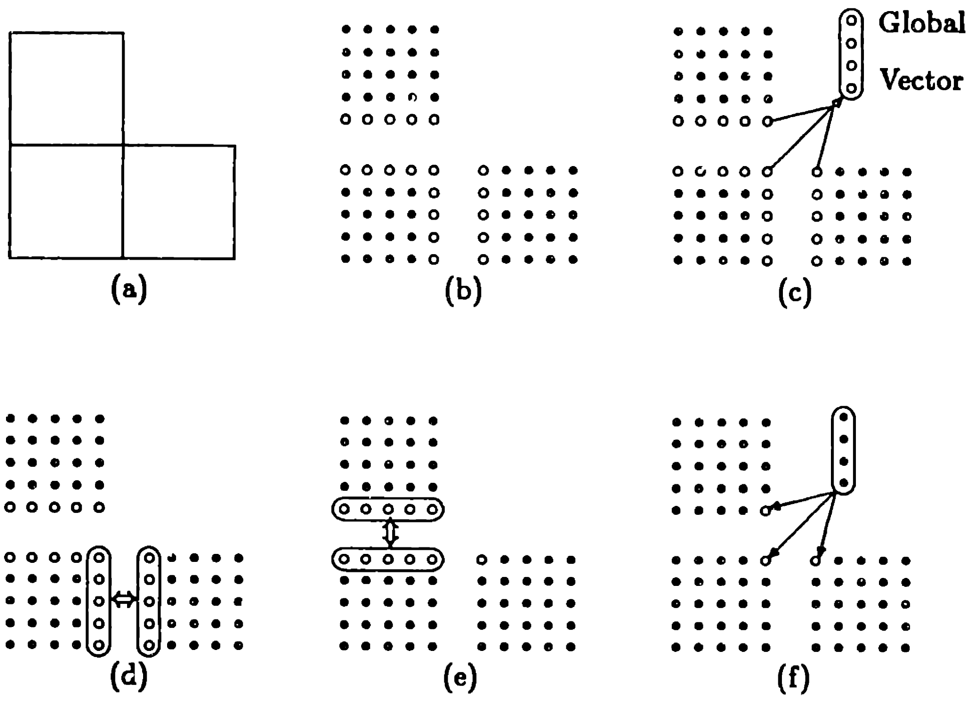


Figure 5.3: Direct stiffness summation for irregular geometry in $\mathbb{R}^2$: (a) three element mesh; (b) nodal content of incomplete residual, $\tilde{r}^k$; (c) $O(\log M)$ map-and-sum vector reduction to gather special nodes onto a global data structure; (d) and (e) standard bi-directional exchange and sum sequence, ψ; (f) final map ($O(\log M)$ fan out) which overwrites local data at special nodes with correct, complete residual.

of clock cycles required to perform direct stiffness summation is then

$$c_3 \frac{KN^{d-1}}{M} + c_4\sigma(N^{d-1})N^{d-1} + c_5\sigma(\epsilon)\epsilon \log M \quad , \tag{5.8}$$

where the c_3 term represents the summation of all edge values, the c_4 term represents the communication of edge values between processors, and the c_5 term accounts for the special-node treatment. The $c_5 \log M$ term can be further reduced for architectures which admit contention-free $\log M'$ vector reductions for "local" subsets of $M' < M$ processors.

This summation algorithm is both general and relatively efficient, and is guaranteed tc effect the direct stiffness operation by virtue of the special node treatment. Furthermore, the regularity and special nodes associated with a candidate sequence ψ can readily be determined by computing the result of a multiplicative ψ for a field comprising (locally) distinct primes, and comparing with the correct result of such a direct stiffness operation. (Direct stiffness multiplication is defined as in Figure 5.2 with arrows implying multiplication, not addition.) The analysis of our parallel direct stiffness method is however, not complete, in that no studies have been perfromed to determine minimally irregular sequences.

The residual calculation (5.3) is the most complicated operation in conjugate gradient iteration. The diagonal-collocation operation (5.4) is completely concurrent and communication-free, with compuational complexity c_6KN^d/M . Similarly, the inner product (5.5) is a standard vector reduction, which is evaluated by first performing intra-element/intra-processor sums, and then evoking a $\log M$ inter-processor vector reduction (5.2). The resulting computational complexity is

$$c_7 KN^d/M + c_8\,\sigma(1) \log M \quad , \tag{5.9}$$

where the first and second terms reflect intra- and inter-processor summation, respectively. Note that for inner products of functions $u_h \in X_h$, the elemental partial sums must take into account the "multiplicity" of boundary nodes.

On the basis of the preceding analysis we arrive at an approximate expression for the number of clock cycles required to solve (3.11) by conjugate gradient iteration on M processors,

$$\begin{aligned} Z_M^A &= N_\epsilon^A \left\{ \frac{c_1 K N^{d+1}}{M} + \frac{c_2 K N^d}{M} + \frac{c_3 K N^{d-1}}{M} \right. \\ &+ \left. c_4 \sigma(N^{d-1}) N^{d-1} + c_5 \sigma(\epsilon) \epsilon \log M + c_8 \sigma(1) \log M \right\} \quad . \end{aligned} \tag{5.10}$$

We identify the c_1-, c_2-, and c_3-terms of our serial estimate (4.3), however there are now three new terms associated with communication and (only) $\log M$-parallelizable operations. From (5.10) and (4.3) we can derive the (inverse) parallel speedup, $S_\tau^{-1} = Z_M^A / Z_1^A$, in which we keep only the leading order terms in computation and communication,

$$S_\tau^{-1} = \frac{1}{M} + \frac{\alpha}{K N^2} \sigma(N^{d-1}) + \frac{\beta}{K N^{d+1}} \sigma(1) \log(M) \tag{5.11}$$

The estimate (5.11) is general not only as regards geometry, but also in the fact that it extends to the full Navier-Stokes equations by virtue of the Uzawa and splitting methods. Note that (5.11) is the inverse speedup for the same algorithm on the same basic processor/memory unit; vectorization is assumed to occur within the spectral elements, and thus scales out of the speedup analysis. Vectorization can be efficiently applied to (5.6) given the local structure internal to elements.

We make several comments concerning the computational complexity (5.10) and speedup (5.11). First, in the limit of vanishing communication, $\sigma = 0$, we achieve unity-parallel-efficiency performance; this is due to the choice of an intrinsically concurrent iterative procedure. Second, even in the case of non-negligible communication time, $\sigma \neq 0$, the method can maintain good performance due to the $1/N^2$ and $1/N^{d+1}$ algorithmic ratio of communication to computation; this favorable ratio derives from the geometric decomposition of work, the intrinsic substructuring associated with spectral elements, and the correct choice of boundary-minimal polynomial bases. Third, for a fixed discretization $h = (K, N)$ there is an optimal number of processors M_{opt}, $1 < M_{opt} < K$, that maximizes speedup by trading off concurrency and communication through "super-substructuring" of spectral elements. We find that $M_{opt} = \max\{1, \min[K, M'_{opt}]\}$, where M'_{opt} is the local minimum $M'_{opt} = KN^{d+1} \ln 2/\beta\sigma(1)$.

We illustrate the speedup analysis by plotting in Figure 5.4 S_τ^{-1} versus M for some representative parameter set (K, N, d, σ). The parallel overhead costs are depicted by the dashed curves which represent the direct stiffness summation and inner-product communication times, denoted c_{ds} and c_{ip}, respectively. (The direct stiffness curve is shown to grow slightly with increasing M in anticipation of contention and slight M dependence of direct stiffness communication on real machines.) It is clear from Figure 5.4 that the inner-product time can become dominant as the number of processors is increased, and that there results a minimum obtainable solution time at a finite number of processors, M'_{opt}.

By virtue of the fact that M'_{opt} scales with K and that M is bounded by K (the spectral element being an "indestructible" data unit), we conclude that the high-order spectral element parallel paradigm is intrinsically medium-grained in character; the number of processors and speedup grows with problem size (in contrast to fixed speedup in coarse-grained processing), however the number

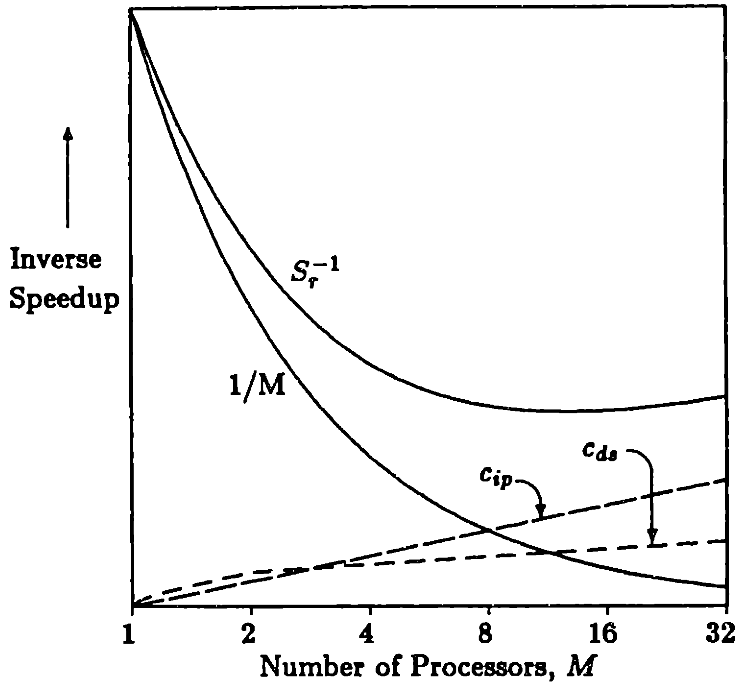


Figure 5.4: Illustration of computing costs associated with the parallel conjugate gradient algorithm, measured in wall clock time. The growth of the inner-product time illustrates the limiting nature of *global* operations, resulting in a minimum solution time for a number of processors $M_0 < \infty$.

of degrees-of-freedom per processor remains "large" (in contrast to fine-grained processing). In the case where $M_{opt} = M'_{opt} < K$ it can be argued that the implicit granularity of the spectral element discretization is not the limiting factor in speedup; in the case where $M'_{opt} > K \rightarrow M_{opt} = K$, it is clear that the spectral granularity is potentially limiting performance.

We remark briefly on the implications of the above analysis as regards memory requirements. Given that the spectral element iterative solvers require c_9KN^d memory, it follows that under optimal conditions, for which $M = M'_{opt}$, the requisite memory-per-processor is $c_9\beta\sigma(1)/N$; more memory is "unneccesary", and less memory will degrade performance by forcing the calculation to $M > M_{opt}$ and higher τ. It will thus often be the case that a problem will only fit on $M \approx M_{opt}$ processors, preventing direct measurement of speedup for $M < M_{opt}$. This is not a serious problem, as the goal of parallel processing is not the generation of complete speedup curves; furthermore, one can readily fit models to available performance data in order to predict performance for memory-excluded M (see Chapter 7). Note that in any real computer the number of processors in a system is fixed at M_{max}, and thus memory-per-processor should be larger than that required at M'_{opt} in order to accomodate solution of large problems on $M_{max} < M_{opt}$.

We next analyse the parallel efficiency, $\eta \equiv S_\tau/M$. From equation (5.11) we have, to leading order:

$$\eta \approx 1 - \alpha\frac{M}{KN^2}\sigma(N^{d-1}) - \beta\frac{M}{KN^{d+1}}\sigma(1)\log(M) \qquad (5.12)$$

We note that the constants governing the degradation in efficiency in (5.12) are directly proportional to $\sigma \equiv \Delta/\delta$, thus implying that a straightforward approach to improving system performance is to decrease Δ. While this certainly yields better processor utilization, it does so at a cost, since a reduced Δ implies a faster, more expensive, network. The difficulty is that the efficiency, η, only measures processor utilizaton and does not account for the cost of the associated network. In addition, parallel efficiency is strongly dependent upon the choice of algorithm and the scalar-vector mix of the particular application (changing the effective δ). It cannot be taken as a measure of "goodness" of any particular algorithm-architecture coupling, as it relatively easy to obtain high efficiency on a system with slow processors, implying that the reduction in σ is brought about by an increase in δ rather than a decrease in Δ. Moreover, used as a performance measure it can in fact be misleading, since any algorithm or architecture improvements which directly reduce the time to compute on a single processor, τ_1, will also reduce the time on M processors, τ_M, but will result in a decreased efficiency.

It is nonetheless of interest to analyse the efficiency versus number of processors for a fixed algorithm-architecture coupling and for a fixed ratio, K/M [38]. Such an analysis addresses the question, "What performance will be obtained if the problem size is doubled and the number of processors is doubled?" This is readily computed from (5.12), yielding:

$$\eta\Big|_{K/M} \approx \eta_0 - \frac{\beta M}{K N^{d+1}} \sigma(1) \log(M) \tag{5.13}$$

where η_0 is the efficiency obtained on a smaller system for some $M > 1$. From (5.13), it is clear that there is only a small logarithmic degradation in the efficiency as the problem size and machine size grow simultaneously.

Remark 1: Although we have focussed primarily on the variational structure of our methods, there is, of course, an equivalent linear-algebra interpretation of the work decomposition. To illustrate this algebraic viewpoint we consider the one-dimensional spectral element discretization of Figure 5.5a, for which the matrix structure (3.10) of the discrete Laplacian is shown explicitly in Figure 5.5b. It is seen that $\mathbf{A}$ comprises four ($K = 4$) full submatrices, $[A^k] = A^k_{pq}$, with each pair of adjoining submatrices coupled by a single overlapping row and column associated with the C^0 continuity condition.

We then consider the elemental "incomplete" residual $(\tilde{r}^k) = \tilde{r}^k_p = A^k_{pq} u^k_q = [A^k](u^k)$. At internal nodes ($p = \{1, 2, ..., N-1\}$) $\tilde{r}^k_p$ is complete ($= r^k_p$), as there is no overlap between the $[A^k]$ for these nodes. However, at the element interface between two elements k and $k+1$, the complete residual must be the sum of two inner products: (u^k) with the last row of $[A^k]$; and (u^{k+1}) with the first row of $[A^{k+1}]$. The sum of these two inner products is precisely the sum of the incomplete residuals $(\tilde{r}^k)$ at the interface nodes, thus showing how the incomplete residual/direct stiffness procedure is related to the underlying matrix structure.

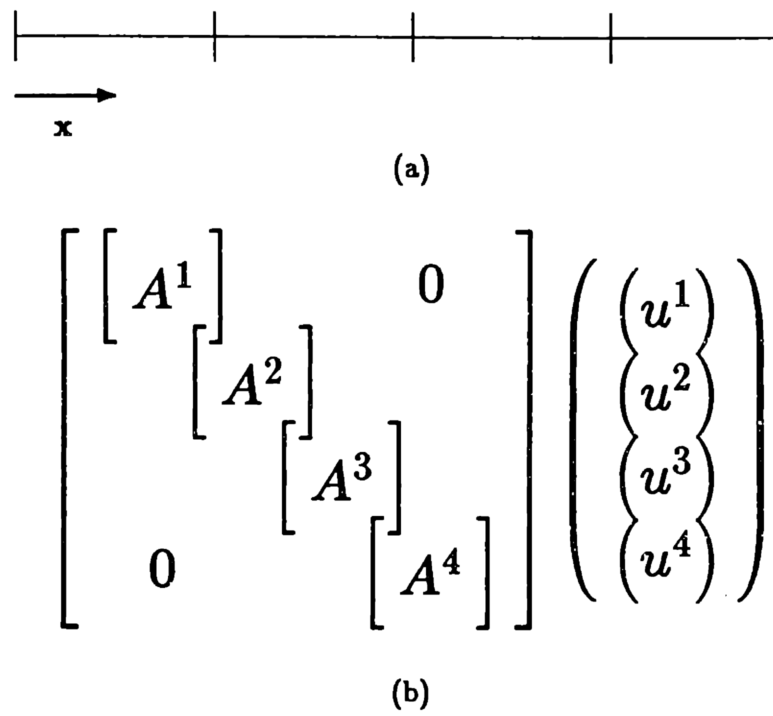


Figure 5.5: Structure of the discrete Laplacian operator, **A**, for the four element discretization in $\mathbf{R}^1$ shown in (a). The full submatrices in (b) have a single overlapping row and column corresponding to the degree-of-freedom which is shared at element interfaces.

5.3 Comparison with h-Type Substructure Methods

The description of spectral element discretizations in Chapter 3, of iterative solvers in Chapter 4, and of parallel constructs in the previous sections can be readily extended to h-type finite element substructure techniques [37,5,6]. Following the spectral element derivation, we take our discretization pair $\tilde{h} = (\tilde{K}, \tilde{N})$ to imply that the discretization is comprised of $\tilde{K}$ regular subdomains, each consisting of $\tilde{N}^d$ linear elements. The global $\mathcal{H}^1$ condition and Dirichlet conditions are the same as for the spectral element basis; this fact will be critical in evaluating computational complexity. The resulting finite element equations are very similar to (3.10), however there is now a great deal of intra-substructure sparsity in the matrix due to the locally compact support of the low-order finite element space. That is, the submatrix blocks in Figure 5.5 which are full for the spectral element discretization are now sparse (in fact, tri-diagonal).

The extension to the multi-dimensional case closely parallels that of the spectral element development. We use tensor product spaces and bases, and direct stiffness procedure shown in Figure 5.2. Note that the tensor product forms are not important in h-type methods as regards sum factorization (5.6) ($\mathbf{Au}$ products are evaluated in terms of local stencils), however they are important in maintaining local structure. The finite element substructure equations and spectral element equations are readily defined in terms of the same quantities due to their common variational foundation; in fact, substructure h- and spectral element methods can be used simultaneously in the same calculation using nonconforming "mortar" methods [35,39].

We can now consider the computational complexity, $\tilde{Z}_M^A$, associated with conjugate gradient solution of substructured finite elements on the model medium-grained processor of Figure 5.1. For the discretization parameter $\tilde{h} = (\tilde{K}, \tilde{N})$ and M processors we find to leading order (assuming for simplicity no "special" nodes),

$$\tilde{Z}_M^A = \tilde{N}_\epsilon^A \left\{ \frac{\tilde{c}_1 K \tilde{N}^d}{M} + \tilde{c}_4 \sigma(\tilde{N}^{d-1}) \tilde{N}^{d-1} + c_8 \sigma(1) \log M \right\} \quad . \tag{5.14}$$

where the significant difference between (5.14) and (5.10) is the reduced work needed to evaluate the h-type residual. Of interest is comparing the time to compute, τ_M, for the spectral element and finite element approximations, $\rho_M = \tau_M / \tilde{\tau}_M = Z_M^A / \tilde{Z}_M^A$:

$$\rho_M = \frac{N_\epsilon^A \left\{ \frac{c_1 K N^{d+1}}{M} + c_4 \sigma(N^{d-1}) N^{d-1} + c_8 \sigma(1) \log M \right\}}{\tilde{N}_\epsilon^A \left\{ \frac{\tilde{c}_1 K \tilde{N}^d}{M} + \tilde{c}_4 \sigma(\tilde{N}^{d-1}) \tilde{N}^{d-1} + c_8 \sigma(1) \log M \right\}} \tag{5.15}$$

where we assume that the same number of spectral element/substructures, $K = \tilde{K}$, and processors, M, are used in each calculation.

To begin we assume $\tilde{N}_\epsilon^A = N_\epsilon^A$, and take $\tilde{N} = \mu N$; $\mu > 1$ as the spectral element approximation will always be at least as good as the linear h-type approximation (recall that the error, ϵ, is fixed). We start by considering only the first terms in the numerator and denominator; this ratio is the usual serial work comparison of high-order and low-order methods, $\rho_1 = Z_1^A / \tilde{Z}_1^A = N/\mu^d$. It can be shown that ρ_1 is significantly less than unity for an interesting class of problems, in particular for smaller ϵ and higher space dimension d [1]. Of interest in the context of the current parallel analysis is the fact that the two remaining (communication) terms in the work estimate (5.15) are at least as large for the low-order method ($\tilde{c}_4-$, $\tilde{c}_8-$ in the denominator) as for the high-order method (c_4-, c_8- terms in the numerator); in particular, if the direct stiffness summation offset is important, the low-order-method communication terms can be larger than their spectral element counterparts by the factor μ^{d-1}. We thus see that the relative advantage of high-order methods improves in a medium-grained parallel environment, due to the fundamental fact that communication is order-independent if proper boundary-minimal bases are chosen; this is a general argument for high-order methods, and need not be restricted to p-type ($N \rightarrow \infty$) convergence strategies.

We return briefly to the assumption that $\tilde{N}_{\epsilon}^{A} = N_{\epsilon}^{A}$. In fact, for the diagonal preconditioner on "uniform" meshes $\tilde{N}_{\epsilon}^{A} = \mu N_{\epsilon}^{A}$. Although this might be used as a further argument in favor of high-order methods, optimal (K, N)-independent multigrid schemes can, in fact, be found for both finite element [40] and spectral element [33,34] approximations, and thus this point is not relevant. In particular, new intra-element spectral element multigrid methods achieve nearly (K, N)-independent convergence while maintaining approximately the same computational complexity as conjugate gradient iteration; with this modification, the algorithms described here are optimal as regards both work per iteration and number of iterations required.

Lastly, it should be noted that the $\tilde{h} = (\tilde{K}, \tilde{N})$ description of the finite element discretization is artificial in that it imposes a coarser granularity on the problem than is actually present. The preceding analysis is thus only directly relevant when $M_{opt} < K$. When this condition is not satisfied, the finite element substructure approach can be readily refined due to the homogeneity of the approximation, whereas extension of the spectral element method to intra-element parallelism is less straight-forward. Our conclusions for the medium-grained paradigm should, therefore, not be applied to the fine-grained case without additional analysis.

5.4 Architecture Mappings: Message-Passing Hypercube

In this section we consider how the model parallel processor defined in Figure 5.1 and analyzed in Sections 5.1-5.3 maps to message-passing hypercube architectures. We recall that a medium-grained hypercube network is defined by $M = 2D$ "large" processors, $P_p, p = 1, ..., M$, with a direct link between any two processors P_p and P_q for which $p-1$ and $q-1$ differ only in only bit in their binary representation). The topological properties of hypercubes are summarized in [10], and numerous applications of hypercubes are described in [41,42].

We assume that the spectral elements have been distributed amongst the processors according to some partition E_q. If we compare an arbitrary partition on the hypercube to the ideal partition on our model parallel processor of Figure 5.1, our communication estimates (41) will be modified by the introduction of non-nearest-neighbor communication (non-unity-dilation mappings) and possible contention (network load imbalance/saturation). In general it will not be possible to find a mapping for which there exists a direct link in the hypercube for every direct link in our model processor. That is, the hypercube architecture violates assumption (5.1).

The first, and most obvious, effect of not satisfying (5.1) is that the direct stiffness summation will require, at best, $\lambda_1 = (2d)K/(MD/2)$ more communication steps, and at worst, $\lambda_1 = O(K)$ more communication steps, due to the lack of direct links between element pairs assumed in the ideal model of Figure 5.1. The second effect, with which we associate a multiplier λ_2, derives from the fact that a particular hypercube partition Ep may give physically adjacent spectral elements non-nearest-neighbor positions in the hypercube network. This will potentially increase the transmission time between these spectral elements in the direct stiffness summation procedure; the magnitude of the deterioration will depend on the message-passing protocol. For the case of store-and-forward we expect a maximum increase in transmission time of $O(\log M)$; for the case of wormhole or pipeline routing we expect substantially less deterioration. The third effect, with which we associate a multiplier λ_3, is the fact that, in the absence of direct links between communication elements network, contention can occur during routing through the hypercube. This effect can be quite difficult to quantify, in particular for general partitions on large cubes.

We note that all of these effects are associated with the direct stiffness term of (41); the $\log M$ communication terms are unaffected by the hypercube mapping as the hypercube architecture honors (5.2) by virtue of simple binary-tree-like embeddings [10]. We thus arrive at our new estimate for speedup for the hypercube system,

$$S_r^{-1} = \frac{1}{M} + \frac{\lambda_1 \lambda_2 \lambda_3 \alpha}{K N^2} \sigma(N^{d-1}) + \frac{\beta}{K N^{d+1}} \sigma(1) \log(M) \qquad (5.16)$$

in which only the direct stiffness term is modified. This speedup model will serve to interpret the hypercube computational results to be presented in Chapter 7.

The above considerations suggest that the spectral element-to-hypercube partition can lead to computational inefficiencies. Although on computers with fast communication and direct routing these effects may not by leading order, it is likely that computation speeds will always outpace off-board communication rates, and these mapping issues should therefore not be ignored. We briefly discuss here several fairly standard mapping strategies. The first strategy, an intra-processor strategy, S_{1intra}, attempts to partition elements such that members of E_q share edges; this reduces λ_1. Furthermore, this intra-processor strategy promotes inter-element nearest-neighbor mappings, S_{1inter}, which reduce λ_2 and λ_3. The second intra-processor strategy, S_{2intra}, randomly partitions the elements to form the E_q; the motivation behind this strategy is to render the calculation load-balance-insensitive with respect to local mesh refinement [43]. Although we do not consider refinement-induced load imbalance in this thesis, it is certainly an important issue. The strategy S_{2intra} does not preclude subsequent attempts at S_{1inter}, however it certainly makes the task difficult, and one must conclude that S_{2intra} will tend to increase not only λ_1, but also λ_2 and λ_3. Heuristics for achieving these strategies are described in [43-46].

Chapter 6
Implementation of Parallel Constructs

This chapter describes in detail several of the kernels required for efficient implementation of the parallel algorithms introduced in the previous chapter. Some background regarding distributed memory parallel processing is provided first. The key communication dependent routines are presented, including an in depth discussion of the direct stiffness implementation. Finally, the heuristics of the element-to-processor mapping algorithm are outlined.

6.1 Distributed Memory Parallel Processing

Distributed memory parallel computers offer tremendous potential as a scalable parallel architecture because both the memory and the primary memory-processor bandwidth grow in direct proportion to the number of processors in the system. Distributed memory systems are comprised of independent computers, or nodes, each containing a processor which is directly connected to *local* memory containing both data and source code. Processors typically access non-local data via a slow, indirect, network and data transfer protocol, where the network is characterized by its topology, e.g., ring, mesh, or hypercube. This direct/indirect memory hierarchy favors algorithms in which the majority of the computational effort is localized within a processor.

Typically the nodes are attached to a front-end machine, or host, which serves to download the executable image and data to the nodes, and to initialize the problem to be solved. Each node has a unique identification number so that the host can selectively download information. The host acts as the interface to the local area network and, in some cases, to the secondary and tertiary storage.

During execution, nodes run asynchronously, following locally resident source code and processing locally resident data. In practice, distributed memory machines are typically programmed in what has been termed "single program, multiple data" (SPMD) fashion where the nodes have identical copies of the source code but different sets of data [47]. Data is transferred between processors via a message passing protocol whereby data packets are explicitly sent from one processor to another. The data is identified by a label so that the receiving processor can discriminate incoming messages. Issuing a receive-wait command will cause a processor to halt execution until the desired data is received. Processor synchronization is inherent in this type of protocol since receipt of data implies that the data is in fact ready for processing by the receiving node.

The independence of each processor/memory unit implies that there is an additional hierarchy to data structures which is not present in serial or shared memory architectures. In a program constructed according to the SPMD model, there will be M distinct copies of every data element - one on each processor; the contents can of course vary from processor to processor. The data on each node derives its unique identity either as a result of different initial conditions as supplied by the host, or through program branch points which are dependent upon the particular node identification number. If a variable which is changing in value during program execution is to have the same value on all processors, interprocessor communication will be required to make a comparison or condensation of the distinct copies of that particular variable. This concept is fundamental to understanding vector reduction on distributed memory systems.

6.2 General Implementation

Our methods are implemented in an essentially machine-independent fashion. First, we construct a spectral element code in a standard high-level language in which each spectral element is treated as a "virtual parallel processor". In particular, each spectral element is treated as a separate entity, and all data structures and operations are defined and evaluated at the elemental level. The only procedures which require communication are, by construction, direct stiffness summation and vector reduction, which are relegated to special subroutines which effect data transfer based on the local (element-based) algorithms described in previous sections.

It is clear that the virtual-parallel-processor spectral element code will achieve the full parallel potential of the underlying algorithm on our model system of Figure 5.1 if we simply unroll the elemental index, and descend identical (save data) copies of the code to M processors. Each processor p is then responsible for a single (or group) of spectral elements corresponding to the partition E_q. It follows that the virtual-parallel-processor code can be readily ported to any computer whose architecture is sufficiently "similar" to the hypothetical model of Figure 5.1; the only machine-dependent code comprises "device drivers" which enact the low-level communication required by the direct stiffness summation and vector reduction subroutines. The class of architectures "similar" to our model processor is at least as large as the class of message-passing multiple-instruction multiple-data architectures. (Note that the class of architectures "similar" to our native system of Figure 5.1 is larger than the class of architectures which map well to our native system; for instance, the virtual-parallel-processor code is readily ported to a ring architecture, however the ring will be susceptible to significant contention.)

Following the SPMD paradigm, each processor is programmed as though it were responsible for the entire computational domain Ω. Since the total domain is comprised of a group of independent elements, a subdomain is readily defined as a subset of the elements. As there are no constraints on the element numbering scheme (e.g., no bandwidth minimization is required when using iterative solvers), the elements within each subset can be renumbered from 1 to K_p, where K_p is the number of elements on a particular processor, p. Element identity consists of geometry, material properties, boundary conditions and element-to-element connectivities. Of these, all can be renumbered arbitrarily, with the exception of the element-to-element connectivity which must be preserved in order to retain the original domain topology. A look up table on each processor provides a means for reconstructing that topology; its use is required only when performing direct stiffness summation.

In developing the algorithms, three guiding criteria are used to optimize the communication operations. The first is of critical importance: due to message startup costs, it is preferable to send a single message of length n, rather than to send n messages of length 1 (a message is a packet of data transferred from a sending node to a receiving node). The second optimization is to "cover" communication whenever possible, i.e. keep the processors busy while messages are in transit. The final concern is to avoid communication schemes that result in a dead-locked state wherein processors are waiting for messages which are never sent. An example of a dead-locking scheme is presented in the direct stiffness summation section below.

We close this section by noting that for general purpose programs that have been constructed algorithmically to exploit parallelism, the additional work required to code the method in a parallel-compatible fashion is relatively minor using the strategy described above. The importance of automatic parallelizing compilers for large-scale general-purpose partial differential equation solvers is not clear.

6.3 Vector Reduction

We begin with a discussion of vector reduction operations as their parallel implementation is relatively straightforward and serves as a good introduction to distributed memory processing. Vector reductions can be classified as any operation (generally commutative and associative) in which $q \geq 2$ values are reduced to a single result. In the spectral element algorithms, we are primarily concerned with cases in which q is a multiple of K, i.e., values which are distributed across elements. Examples of vector reductions are computation of inner products, $\alpha = \mathbf{u} \cdot \mathbf{u}$; computation of the Courant number, $C = \max(\vec{u} \cdot \vec{u} / \vec{u} \cdot \Delta\vec{x})$, or determination of any *global* parameter which is dependent upon *local* data, for instance, a logical flag whose value is set according to the presence of some boundary condition. The general approach is to compute the result of M vector reductions from the constituents on each processor independently, then evaluate the final result by considering the M remaining values.

To illustrate the vector reduction we consider computation of an inner-product. When evaluating inner-products on the element based data structures, it is necessary to account for the multiple representation of nodes lying on shared interfaces between two or more elements. This can be done by multiplying each term in the inner product by the inverse of the number of elements which share a particular node. The inner product kernel therefore has the form:

On each processor p:

$$\alpha^p = \sum_{\kappa=1}^{K_p} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} a_{ijk}^{\kappa} \, a_{ijk}^{\kappa} \, m_{ijk}^{\kappa} \quad , \tag{6.1}$$

then,

$$\alpha = \sum_{p=1}^{M}{}'' \alpha^p \quad , \tag{6.2}$$

where m_{ijk}^{κ} is the inverse multiplicity defined at each point. The $\sum''$ (6.2) indicates a global summation across the M processors utilizing $\log M$ communication cycles according to (5.2). The optimal global summation algorithm will be dependent upon the network topology; an efficient algorithm for the particular case of hypercube networks is presented below.

Consider the case in which eight processor/memory units, $p = \{1, 2, ..., 8\}$ are arranged on a hypercube network as depicted in Figure 6.1a. The eight values of α^p are condensed to a single value according to the data exchange sequence depicted in Figure 6.1b-6.1d. The equivalent coding sequence has the form:

On processor p:

```
                $\delta = M/2$
startloop:      if p $\leq \delta$ then:
                   receive $\alpha^{p+\delta}$
                   add $\alpha^{p+\delta}$ to $\alpha^p$
                   $\delta \Leftarrow \delta/2$
                   goto startloop
                else
                   send $\alpha^p$ to processor $p - \delta$
                   exit routine
endloop:
```

A similar (inverse) procedure is used to redistribute the final result α^1 to all the processors. The above procedure is specific to hypercube network topologies in that it assumes that if δ is a power of 2, then there will always exist a direct link between a processor p and a processor $p + \delta$. It can be seen that this procedure does reduce the set of values α^p to a single result in $D = \log M$ communication cycles [10]. However, such $\log M$ reductions are not restricted to hypercube topologies; in fact, a $\log M$ scheme has been found for more general message passing lattices [48].

Remark 2: We make the additional comment that the $\log M$ vector reduction software to compute $\alpha = \sum^{n} \alpha^p$ is provided with the system on the Intel iPSC hypercubes, so that efficient programming of this fundamental communication operation is not required. Identification and provision of such basic software tools is important in this new and rapidly changing technology in that it allows

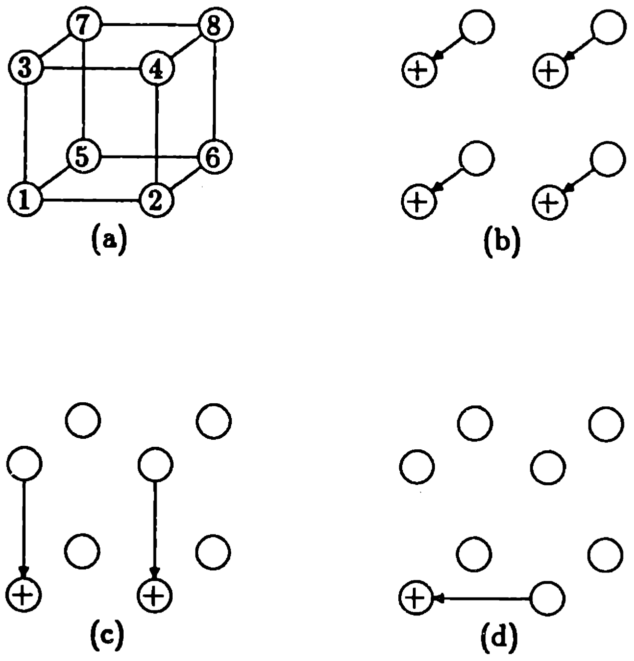


Figure 6.1: Inter-processor vector reduction for hypercube of dimension $D = 3$. Processors (1-8) and associated communication network are shown in (a). D-cycle data exchange with active communication lines denoted by arrows and active processors denoted by "+" is shown in (b-d). At each stage, values are sent along the communication lines to active processors where they are added to the locally resident value.

application developers to concentrate on algorithmic issues rather than machine specific constructs while still obtaining optimum performance.

6.4 Direct Stiffness Summation

One of the critical ingredients to general three-dimensional spectral element algorithms is an efficient implementation of the direct stiffness summation operation. As data within elements is well structured, the flexibility to handle unstructured domains derives from the ability to couple the subdomains together in an arbitrary manner. In this thesis we restrict our attention to the cases where the subdomain coupling is conforming, i.e. each interface point on the surface of a given element has a one-to-one correspondence with an interface point on the adjacent element. Recently developed non-conforming couplings which allow for greater geometric flexibility are discussed in [35,38].

Development of the direct stiffness summation algorithm for conforming discretizations in higher space dimensions is primarily a problem of enumeration; a subset of an array of elemental data is to be added to a corresponding subset in another element and a compact description of the correspondence is required. In addition, the presence of shared vertices in $\mathbf{R}^2$ (edges in $\mathbf{R}^3$) implies that data must be exchanged and summed amongst several elements which share a given vertex. The connectivity at such points can be very complex and an efficient means of handling the vertices is required which retains the full generality of the spectral element discretization.

Our approach to the conforming direct stiffness summation is two-fold:

first, an edge (face) exchange sequence, ψ, is employed to update the majority of the interface residuals for each element; second, a local-to-global mapping operation is employed for all remaining vertices, so called "special nodes", which are not correctly treated by the first step. This two-step direct stiffness procedure is depicted in Figures 5.2 and 5.3. In the limit that no vertices are treated correctly in the first pass, the local-to-global map will correctly treat all nodes, just as in a standard finite element fully-indirect-addressing scheme. However, in many fluid mechanics problems a large portion of the computational domain is filled by a regular array of elements for which the number of special nodes can be reduced to zero, so that an efficient algorithm results.

Parallel implementation of direct stiffness summation is more complex than the inner product evaluation and is in fact the key communication algorithm developed for the distributed memory/spectral element implementation. We discuss three issues: developing the face data packing (or face-matching) algorithm; establishing the element face exchange sequence, ψ; and implementing the special node treatment for arbitrary (conforming) element configurations. Of these issues, only the exchange sequence is specific to distributed memory processors; the other issues are generic to efficient, general-topology, direct stiffness summation on both serial and parallel processors.

6.5 Element Face-Matching

We next address the enumeration or "face-matching" problem for conforming topologies. To establish a framework for discussion, we begin with some notations used for three-dimensional spectral element data structures. We

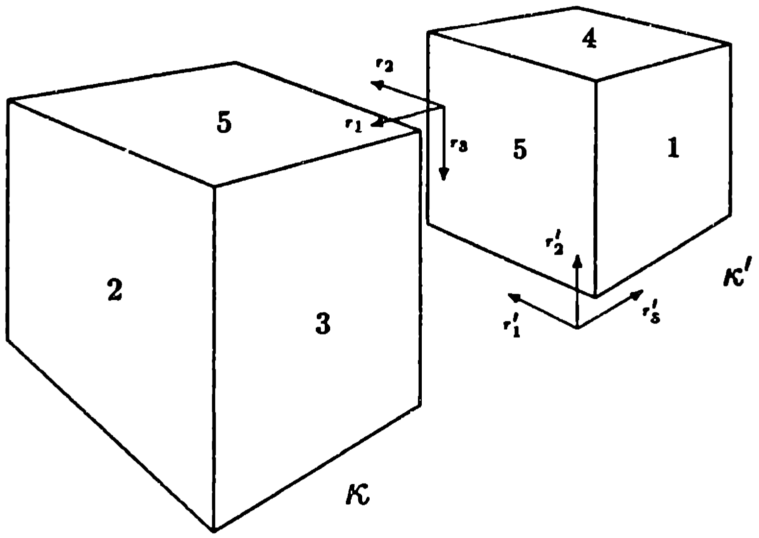


Figure 6.2: Two elements in $\mathbf{R}^3$ illustrating relative rotation, $r = r' = 3$, for direct stiffness summation of faces $F = 1$ and $F' = 5$. Fiducial nodes are denoted by local coordinate axes.

consider the elements shown in Figure 6.2 Recall that in three-dimensions, the spectral elements are mapped from physical space, $\mathbf{x} = (x, y, z)$ to their local cartesian coordinate space $\mathbf{r} = (r, s, t) = (r_i, i = 1, 2, 3)$. In an element based scheme, it only makes sense to refer to local coordinates since elements can be deformed and oriented such that there is no well defined "x-, y-, or z-direction". We number the element faces with the symmetry preserving scheme depicted in Figure 6.2, in which a face with an outward pointing normal in the direction r_i is denoted as $F = 2i$ and in the direction $-r_i$ as $F = 2i - 1$. The vertices are numbered from 1 to 8; advancing along an element edge in the direction $+r_i$ takes one from vertex v to vertex $v + 2^{i-1}$.

The nodal points within an element are numbered in a Cartesian ordering scheme, u_{ijk} : $i,j,k \in \{1,2,...,N\}^3$, or with an equivalent sequential single index notation, u_l : $l \in \{1,2,...,N^3\}$. The relationship between the two numbering schemes is given by:

$$l = i + N(j-1) + N^2(k-1) \quad . \tag{6.3}$$

This scheme correlates directly to the FORTRAN array addressing scheme, so (6.3) essentially establishes how the array offsets are computed. The single index notation allows for reduced loop nesting when marching on a $d-1$ dimensional surface in a d dimensional space. Note that using this numbering scheme, the order of the polynomials in the space X_h is in fact $N-1$. We use N for both notations, the true order of the polynomial being described is generally clear in the context of the reference.

The element to element matching is established according to our element based paradigm. Associated with each element is a list describing the orientation of the adjacent elements. The list for an element κ is referenced by face number F, and contains the adjacent element number κ', the adjacent face number F', and the rotation of the face r'. Relative rotation of two faces is established via a fiducial node orientation of the faces. We denote as the fiducial node the lowest numbered vertex on each face. The rotation is the number of edges on face F' which are traversed in the counter-clockwise direction to move from the fiducial node on F to the fiducial node on F' when face F' is oriented with its outward facing normal toward the viewer. In general, the rotation is symmetric, i.e., $r = r'$, and ranges from 0 to 3 for configurations in $\mathbf{R}^3$.

We illustrate the complexity of the face matching problem by considering the two element configuration shown in Figure 6.2 for the case of $N = 5$. We seek to establish the correlation between element node numbers required to carry out direct stiffness summation of an array $\mathbf{u}$ between elements κ and κ'. The faces which are to be summed are $F = 1$ and $F' = 5$; the relative rotation is $r = 3$. According to the single index scheme, the node numbers on the two faces would appear (with r_1 pointing towards the viewer) as:

1	6	11	16	21		21	22	23	24	25
26	31	36	41	46		16	17	18	19	20
51	56	61	66	71		11	12	13	14	15
76	81	86	91	96		6	7	8	9	10
101	106	111	116	121		1	2	3	4	5
		F						F'		

Denoting the ordered sequences to be correlated as $\mathbf{S}_\kappa$ and $\mathbf{S}_{\kappa'}$, we have:

$$\mathbf{S}_\kappa \leftrightarrow \mathbf{S}_{\kappa'} \tag{6.4}$$

$$\mathbf{S}_\kappa = \{1,6,11,16,21,26,31,36,41,46,51,56,61,66,71, 76,81,86,91,96,101,106,111,116,121\} \ , \tag{6.5}$$

$$\mathbf{S}_{\kappa'} = \{21,22,23,24,25,16,17,18,19,20,11,12,13,14,15, 6,7,8,9,10,1,2,3,4,5\} \ . \tag{6.6}$$

The correlation $\mathbf{S}_\kappa \leftrightarrow \mathbf{S}_{\kappa'}$ implies that direct stiffness summation of $\mathbf{u}$ will result in u_1^κ being added to $u_{21}^{\kappa'}$, u_6^κ to $u_{22}^{\kappa'}$, and so on. Note that the the sequences (6.4) correspond to non-adjacent memory locations, implying that a packing operation is required to minimize the buffer length of communicated data. We therefore invoke the transitive property of the correspondance, "$\leftrightarrow$", and introduce an intermediate sequence $\mathbf{S}_I$ to indicate the correlation between the temporary packing array $\mathbf{v}$ and the two data arrays $\mathbf{u}^\kappa$ and $\mathbf{u}^{\kappa'}$:

$$\mathbf{S}_I \leftrightarrow \mathbf{S}_\kappa \tag{6.7}$$

$$\mathbf{S}_{\kappa'} \leftrightarrow \mathbf{S}_I \quad , \tag{6.8}$$

where

$$\mathbf{S}_I = \{1,2,3,...,24,25\} \quad . \tag{6.9}$$

The complete direct stiffness exchange sequence is: move $\mathbf{u}^{\kappa}\big|_{\mathbf{S}_{\kappa}}$ to $\mathbf{v}\big|_{\mathbf{S}_I}$, send $\mathbf{v}\big|_{\mathbf{S}_I}$ to the processor associated with element κ', and add the contents of $\mathbf{v}\big|_{\mathbf{S}_I}$, to $\mathbf{u}^{\kappa'}\big|_{\mathbf{S}_{\kappa'}}$.

We next introduce the additional notation required to derive the sequences (6.4) for arbitrary order N, arbitrary face matchings (F, F'), and arbitrary rotation r. We first define the operator $<\mathbf{a},\mathbf{b}>_F$ to imply a sequence of numbers comprised of the sum of the elements of sequences $\mathbf{a}$ and $\mathbf{b}$, generated by first incrementing the elements of $\mathbf{a}$, then $\mathbf{b}$:

$$<\mathbf{a},\mathbf{b}>_F \equiv \{a_1+b_1,\ a_2+b_1,\ ...,a_N+b_1,a_1+b_2,...,a_N+b_N\} \tag{6.10}$$

where the F-dependent contents of $\mathbf{a}$ and $\mathbf{b}$ are to be defined shortly. We further take $-<\mathbf{a},\mathbf{b}>_F \equiv <\mathbf{b},\mathbf{a}>_F$, and $-\mathbf{a} \equiv \{a_N, a_{N-1}, a_{N-2}, ..., a_1\}$. Using these tools, we can express (6.4) as:

$$<s_1,s_2>_1 + f_1 \leftrightarrow <s_1, -s_2>_5 + f_5 \tag{6.11}$$

The fiducial node offset f_F has been added to each of the array sequences in (6.12). The face-dependent contents of the sequences s_1 and s_2 is now clear. For face $F = 1$ on the right hand side of (6.12) we have $s_1 = \{0,5,10,15,20\}$, $s_2 = \{0,25,50,75,100\}$, and $f_1 = 1$; for the left hand side, $s_1 = \{0,1,2,3,4\}$, $s_2 = \{0,5,10,15,20\}$, and $f_5 = 1$. Following the example (6.12), the general expression for a matched pair of element faces (F, F') is given by:

$$<s_1,s_2>_F + f_F \leftrightarrow (-1)^{G+G'+r+1} < (-1)^{R_1}s_1, (-1)^{R_2}s_2 >_{F'} + f_{F'} \quad , \tag{6.12}$$

Face F	s	*elements of* s_1, s_2				
3,4,5,6	s_1	0	1	2	...	$(N-1)$
1,2	s_1	0	N	$2N$	...	$(N-1)N$
5,6	s_2	0	N	$2N$	...	$(N-1)N$
1,2,3,4	s_2	0	N^2	$2N^2$	...	$(N-1)N^2$

Face F	f_F	G
1	1	0
2	$1+(N-1)$	1
3	1	1
4	$1+(N-1)N$	0
5	1	0
6	$1+(N-1)N^2$	1

G'	r	R_1	R_2
0	0	0	0
0	1	0	1
0	2	1	1
0	3	1	0
1	0	0	0
1	1	1	0
1	2	1	1
1	3	0	1

Table 6.1: Look up tables for computing strides - $(s_1, s_2)_F$, fiducial nodes - f_F, and index ordering exponents - G, R_1, R_2, for arbitrary face-to-face pairings - (F, F', r) in $\mathbf{R}^3$.

where $(s_1, s_2, f_F, G, R_1, R_2)$ depend upon (F, F', r) as presented in Table 6.1. From these look up tables, it is easy to construct the sequence (6.12) for any (F, F', r). Note that the sequences s_1 and s_2 are readily interpreted as loop indices in any high-level language, and that (6.12) therefore provides a means of determining the correct start address, stride, and loop ordering for conforming direct stiffness summation of any element pairing in $\mathbf{R}^3$.

6.6 Direct Stiffness Exchange Sequence

Description of the direct stiffness exchange sequence, ψ, is straightforward for a regular array of elements having principal axes (r_1, r_2, r_3) aligned with the physical axes (x, y, z). The x-, y-, (z-) exchange sequence described in Figure 5.2 corresponds to each processor pair first exchanging data on faces 1 and 2, then on faces 3 and 4, (followed by faces 5 and 6 in $\mathbf{R}^3$). Such an ordering will yield the correct residuals at the vertices for any element structure which is topologically equivalent to the regular array in Figure 5.2; the ring topology for the natural convection problem of Chapter 8 is one example. We denote this exchange sequence as the nominal exchange sequence for each element:

$$\psi_\kappa = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 6 \end{bmatrix} . \qquad (6.13)$$

The non-zero entries in ψ_κ are the face numbers, F associated with each pass in the exchange sequence for element κ. The exchange is executed in a semi-lock step fashion. For $i = 1, .., d$ passes, each element sends the contents of the faces which are on row i of the array ψ to the appropriate element/processor. Upon completion of the sending phase, each processor awaits reciept of the reciprocal element face data to be summed with the respective face data denoted in row i. Faces are updated as soon as the data is received, regardless of the order in which they appear on a row in (6.13). Up to $2d$ faces can be sent in the ith pass, a zero indicates that no data is sent or received. Note that synchronization is imposed only in moving from row to row, the exchanges denoted by the entries on a given row are executed asynchronously.

In many instances it will be necessary to modify the nominal exchange sequence (6.13) for a given element. Consider the two-dimensional array of elements depicted in Figure 6.3. The majority of the elements are well ordered, with the exception of a single element (7) which is rotated 90°. Clearly the nominal exchange sequence would lead to a locked communication state; on the first pass, elements 6 and 8 would be waiting for faces 3 and 4 of element 7, while element 7 was in turn awaiting data from elements 3 and 11. Since global synchronization is imposed in moving from row to row of ψ, elements 3 and 11 would never move to the second pass of (6.13), until *all* elements had satisfied the entries on row one of their specific exchange sequence, ψ_κ. One obvious solution to this dilemma is to rotate the element (7) back to its "natural" orientation. However, this has ramifications regarding redefinition of the original problem, recreating the data set provided by the preprocessor, and determining what is "natural" in all cases. An easier approach is to simply redefine the exchange sequence for element pairs having face matchings which deviate from the nominal pairing. For each element pair (κ, κ') only one sequence needs to be restructured.

In the example of Figure 6.3, the reordered sequence would be:

$$\psi_7 = \begin{bmatrix} 0 & 0 & 3 & 4 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 6 \end{bmatrix} . \qquad (6.14)$$

Our heuristic for determining which element to reorder is to count the number of "odd" face-face pairings for each element and change the sequence ψ for the element which has the most "odd" matches. We define "odd" to be any face pair (F, F') which cannot be expressed as $(2i-1, 2i)$ or $(2i, 2i-1)$, for i=1,2,

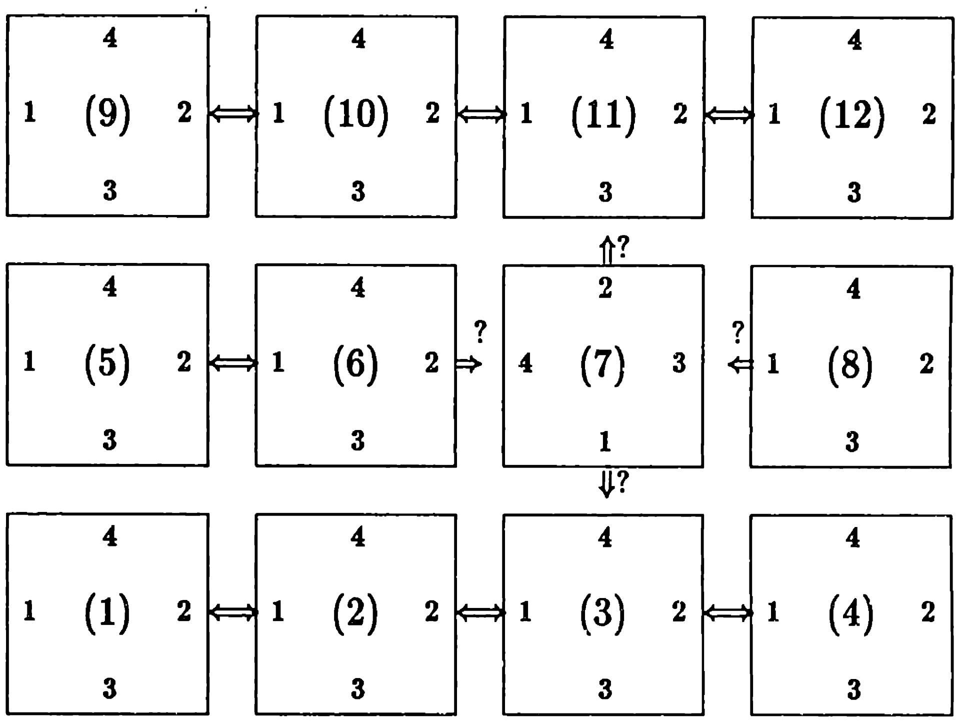


Figure 6.3: Example of an exchange sequence, ψ, for twelve elements (processors) which results in communication deadlock because of the odd orientation of element (7).

or 3.

We discuss the advantages of the above sequencing scheme over possible alternatives. One far less complex scheme would be to have each element send the data on all $2d$ faces to the respective element/processors, then collect the reciprocal data and sum it to the respective local face values. However, such a scheme would fail to update the vertex values correctly for any vertex shared by more than 2 elements, implying that a large number of single word messages would need to be exchanged to complete the direct stiffness summation. An alternative, synchronous, scheme is to have each element send a face, and immediately wait for the reciprocal data for that particular face. Such a scheme also has drawbacks. First, if the reciprocal element is not on the same schedule as the initiating element, dead-lock will result (consider an ordered periodic chain of elements in which every element sends out face 2 and waits for its neighbor to reply with face 1...). Second, even if dead-lock is avoided, one can envision that a forced synchronization of exchanges will result in waves of activity/inactivity propagating through the system of processors if they are not allowed to move onto the next calculation/face-exchange in an asynchronous manner. Finally, the synchronized send-and-wait approach violates our second optimization criteria of covering communication with other operations; the current algorithm allows the processor to commence packing and sending the next face in a given row i of ψ while the previous message is in transit.

6.7 Special Node Treatment

The special node algorithm has been adopted to address the complex issue of developing an efficient and robust direct stiffness algorithm. The implementation is trivial once the list of special nodes is constructed. A copy of a global array is established on each processor to hold the contents of the summed special node residuals; a global vector reduction is used to condense the copies into a single, summed, vector; and an $O(\log M)$ fan out broadcast the summed residual vector to each processor. Each processor maps elemental data to and from the global vector according to a local-to-global mapping which is constructed in a preprocessing phase of the calculation. The basic sequencing strategy has been described in Section 5.2, so we turn now to the problem of special node detection.

The most straightforward way to finding which nodes are "special" in a given configuration is to resort to the original definition of their characteristics: "special nodes are those which fail to be updated correctly by the direct stiffness exchange sequence ψ". We merely pose a direct stiffness problem for which the answer is known, and which is arrived at via a unique sequence, ψ, then check all nodes to verify the result. Nodes which have the incorrect value are appended, along with their counterparts adjacent elements, to the local-to-global map. We choose as our test problem direct stiffness multiplication of distinct primes, as this ensures that the correct result cannot be obtained by multiple contributions of a single vertex.

6.8 Element-to-Processor Mapping

The element-to-processor map is constructed to minimize processor load imbalance and communication overhead [44]. Since the spectral element method is inherently a macro-element decomposition, load balance is achieved by ensuring that the number of elements on any given pair of processors differs by no more than one. The communication time associated with vector reduction operations is fixed for any problem topology and hence does not influence the mapping strategy. In contrast, the overhead associated with element interface communication is directly influenced by the element-to-processor mapping; it is governed by the number of "exposed" faces on each processor, i.e., those interfaces for which adjacent elements are assigned to different processors, and by the distance on the network which separates two, physically adjacent, elements. Our current decomposition algorithm employs a nested-dissection scheme [42,45,46] to minimize the number of exposed element interfaces and thereby reduce the data communication traffic.

Element-to-processor mapping optimizations which seek to ensure that only "nearest neighbor" processors are communicating are not of interest for several reasons. First, such optimizations are strongly architecture dependent, and hence not necessarily portable. Second, it is not generally possible to find such a nearest-neighbor mapping for arbitrary domains. Third, spectral element test problems have shown such optimizations to affect solution times by at most twenty percent on the iPSC/1-VX [49], a machine which has a particularly high σ and which would benefit most from nearest-neighbor mapping strategies.

Chapter 7
Measured Performance Analysis

We have implemented our methods on the Intel vector hypercubes, the iPSC/1-VX/dD and its successor, the iPSC/2-VX/dD. The iPSC/1-VX is a 286-based system with store-and-forward message-passing; the iPSC/2-VX is a 386-based system with pipelined communication routing. In both cases the same vector hardware is used, capable of a peak speed of 10 MFLOPS/board. The two machines differ primarily in scalar speed and communication speed and robustness, with the iPSC/2 representing a significant improvement in both capabilities due to advances in technology and architecture. The iPSC/1 (iPSC/2) 286-based (386-based) mother board achieves .02 (.06) MFLOPS, and communication rates of $\Delta(1), \Delta(\infty) = 5960\mu s, 33\mu s (300\mu s, 1.4\mu s)$. These Intel message-passing hypercubes are clearly similar to our model system of Figure 5.1, and therefore represent a relatively "simple" port of the virtual-parallel-processor code described in the previous section.

7.1 Intel Hypercube Timings

We now analyze the spectral element-Intel iPSC/1-VX algorithm-architecture coupling based on the framework of Section 5.1 and the complexity estimates of Section 5.2. We begin by analyzing the simple three-dimensional "chain"

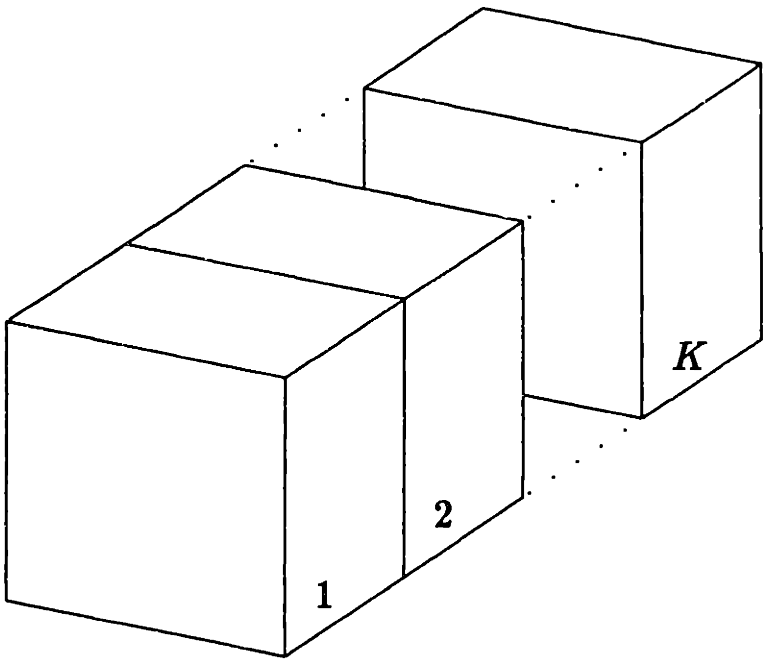


Figure 7.1: Periodic chain of K elements ($N = 10, d = 3$) used for multiprocessor timing analysis.

shown in Figure 7.1 with periodic boundary conditions imposed on all sides. We consider six problems of increasing size, K= 1,2,4,8,16, and 32, respectively, with $N = 10$ in all cases; the partitions E_q for each problem are given in Table 7.1. Note that for a particular K the number of processors that can be used is limited by three factors: memory constraints preclude $M < K/2$; machine size precludes $M > M_{max}$; and algorithm granularity precludes $M > K$. By virtue of the gray-code mapping [10] used for the partitions E_q the hypercube implementation maps exactly to our model processor system, $\lambda_1 = \lambda_2 = \lambda_3 = 1$. (Note that communication between faces of elements on the same processor do not pass through the network.)

Processor	K/M=1					K/M=2				
Number	K=1	K=2	K=4	K=8	K=16	K=2	K=4	K=8	K=16	K=32
1	1	1	1	1	1	1,2	1,2	1,2	1,2	1,2
2		2	2	2	2		3,4	3,4	3,4	3,4
3			4	4	4			7,8	7,8	7,8
4			3	3	3			5,6	5,6	5,6
5				8	8				15,16	15,16
6				7	7				13,14	13,14
7				5	5				9,10	9,10
8				6	6				11,12	11,12
9					16					31,32
10					15					29,30
11					13					25,26
12					14					27,28
13					9					17,18
14					10					19,20
15					12					23,24
16					11					21,22

Table 7.1: Gray code processor-element partition for a periodic chain.

K/M	Time (sec)	Time$_{ca}$ (sec)	Time$_{ds}$ (sec)	Time$_{ip}$ (sec)
1/1	9.7	8.9	0.40	0.40
2/2	14.0	9.0	2.8	2.3
4/4	24.8	8.5	9.1	7.5
8/8	31.7	8.6	9.0	14.0
16/16	37.1	8.6	8.5	19.8
2/1	18.5	17.7	0.40	0.37
4/2	22.9	17.6	2.6	2.3
8/4	33.4	17.2	7.3	7.7
16/8	40.1	17.4	8.5	14.2
32/16	46.6	17.4	8.1	20.2

Table 7.2: iPSC/1-VX timing results for 250 **A** iterations.

We tabulate the results of our numerical experiments in Table 7.2 as a table of $\tau(K,M)$, $\tau_{ca}(K,M)$, $\tau_{ds}(K,M)$, $\tau_{ip}(K,M)$. Here τ is the time to calculate 250 conjugate gradient iterations for the **A** system, 3.11, and $\tau_{ca}(K,M)$, $\tau_{ds}(K,M)$, $\tau_{ip}(K,M)$, represent the breakdown of $\tau(K,M)$ in terms of calculation time, direct stiffness communication time, and inner product communication time. In order to calculate speedup on the basis of this limited dataset we use the analysis of the previous sections to motivate a functional form for τ,

$$f_\tau(K,M) = aK/M + (b + c\log M)\cdot(1-\delta_{1M}) \quad , \tag{7.1}$$

where a, b, and c are constants assumed independent of K and M. We then fit these constants (via least squares) to the total time data τ of Table 7.2, finding a = 9.2 sec, b = 3.1 sec, and c = 6.2 sec; these values are not inconsistent with the direct breakdown of $\tau(K,M)$ into τ_{ca} (a-term), τ_{ds} (b-term), and τ_{ip} (c-term), which serves to verify the form of (7.1). Note also the constancy of τ_{ds} for $M \geq 4$.

From (7.1) we calculate the inverse speedup, $S_\tau^{-1} = f_\tau(K,M)/f_\tau(K,1)$, which is plotted in Figure 7.2; also plotted are the measured speedups for the data of Table 7.2, $\tau(K,M)/f_\tau(K,1)$. The reasonably good fit of (7.1) to the data is further verification of the model. We make several comments concerning the speedup curve of Figure 7.2. First, the optimal number of processors, M_{opt}, is less than K; furthermore, the ratio M_{opt}/K is roughly constant, as predicted by the models of the last section. The fact that $M_{opt} < K$ implies that for this machine, which is a fast calculator and a slow communicator (σ relatively large), the spectral element granularity is more than sufficient. Second, the speedup grows with problem size, as must be the case. Third, the maximum speedup on the largest problem is roughly 5.0, corresponding to a parallel efficiency of η=.3.

To investigate "non-idealities", we have considered two additional tests for the N=10, K=32, M=16 problem. In the first test, we replace the partition of Table 7.1 with the partition E_q=$\{2q-1, 2q\}$, in which we now have a non-gray ordering, but the amount of data passed across the network is unchanged (that is, λ_1 is still unity, but λ_2, λ_3 are potentially greater than unity). In this case τ_{ds} (and hence τ) are increased by 9 seconds to 26 seconds (55 seconds), resulting in a 17% decrease in speedup. In the second test, we replace the partition of Table 7.1 with the S_2-intra partition E_q: $E_1, E_2, ..., E_{16}$ ={1,3}, {2,4}, {5,7}, {6,8}, {9,10}, {11,13}, {12,14}, {15,16}, {17,18}, {19,21}, {20,22}, {23,24}, {25,26},

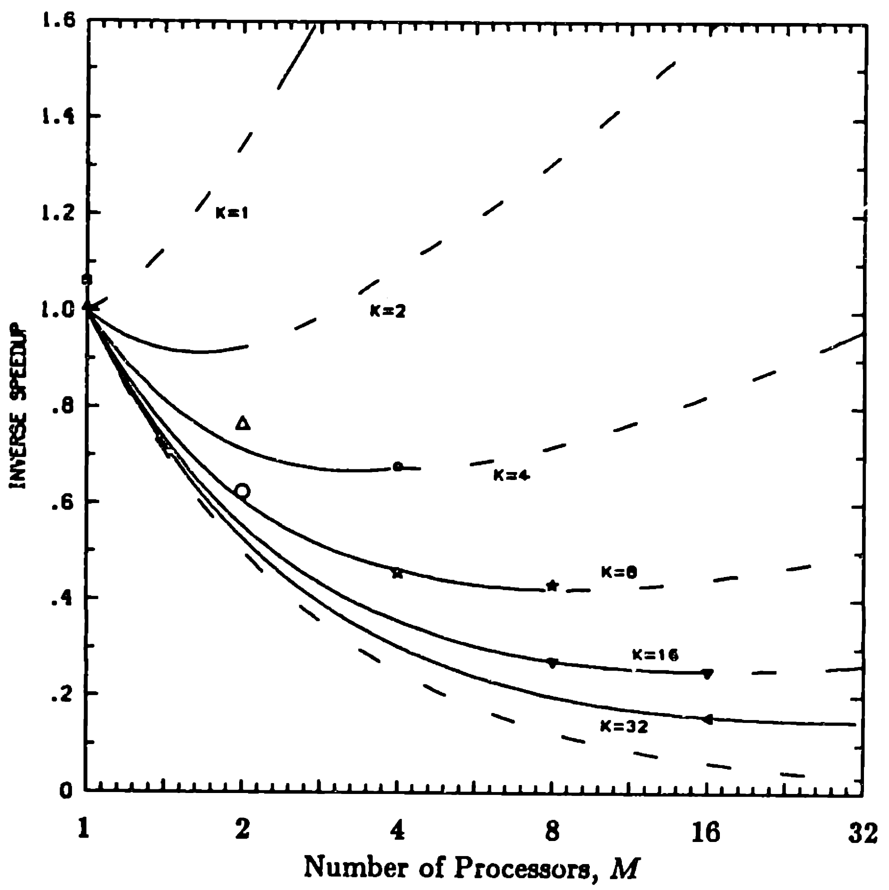


Figure 7.2: Inverse speed-up on the iPSC/1-VX for 250 A matrix iterations of the spectral element configuration in Figure 7.1 for problems defined by K=1,2,4,8,16,32. The solid line indicates the fit $f_T(K,M)/f_T(K,1)$ to the data of Table 4; the symbols represent the actual data. Open symbols indicate the data points for the $M = 2$ cases which are anomalous due to the message passing protocol; these points are not used in computing the fit. The upper dashed lines indicate the (unobtainable) operating regime where $M > K$. The lower dashed line is the peak theoretical speedup, $1/M$.

{27,28}, {29,30}, {31,32}, in which we now not only have a non-gray ordering, but also required twice the amount of data to flow across the network (that is, λ_1, λ_2, and λ_3 are all potentially greater than unity). In this case τ_{d_s} (and hence τ) are increased by 11 seconds to 28 seconds (57 seconds), resulting in a 20% decrease in speedup. We conclude that non-idealities as regards mappings are significant but not dominant.

We perform a similar analysis for the spectral element-Intel iPSC/2-VX coupling. We return to the the chain problem of Figure 7.1 for $N = 10$, $K = 1, 2, 4, ..., 32, 64$ with gray-code ordering on M=1,2,4,...16,32 processors. The timings for 250 **A** iterations are shown in Table 7.3 from which we find for our functional form of τ (7.1) that a = 11.2 sec, b = 0.45 sec, and c = 0.33 sec. We first notice that, in spite of the increased scalar speed of the iPSC/2, a is larger for this case than for the iPSC/1; the particular code used for this chain problem is not as fully vectorized as the original iPSC/1 code and has a slightly higher operation count to accomodate additional flexibility in the operators. The iPSC/2's significantly reduced message transit time, Δ, is evident in the ten-fold reduction in the value of b, and the twenty fold reduction in c. Using these values of a, b and c, we can again compute the inverse speedup, S_τ^{-1}, which we plot in Figure 7.3. We see that the speedup curves are much closer to the asymptotic limit $1/M$ than in Figure 7.3, and that the maximum speedup obtained is 28.8, corresponding to a parallel efficiency of η=0.9. In addition, the projected values for M_{opt} are now much greater than K, implying that the spectral element granularity is potentially limiting, since more processors $M > K$ could be exploited. Note that, while memory constraints on the iPSC/1-VX precluded analysis of problems having more than two (N = 10) elements per processor, the additional 4 Mbytes of memory on the iPSC/2 provides sufficient

K/M	Time (sec)	Time_{ca} (sec)	Time_{ds} (sec)	Time_{ip} (sec)
1/1	11.4	11.3	0.06	0.06
2/2	12.5	11.3	0.74	0.45
4/4	12.8	11.3	0.74	0.75
8/8	13.1	11.2	0.76	1.1
16/16	13.4	11.3	0.75	1.4
2/1	22.3	22.2	0.06	0.06
4/2	23.6	22.4	0.80	0.45
8/4	24.2	22.3	0.80	0.75
16/8	24.5	22.3	0.80	1.1
32/16	24.5	22.4	0.75	1.4

Table 7.3: iPSC/2-VX timing results for 250 A iterations.

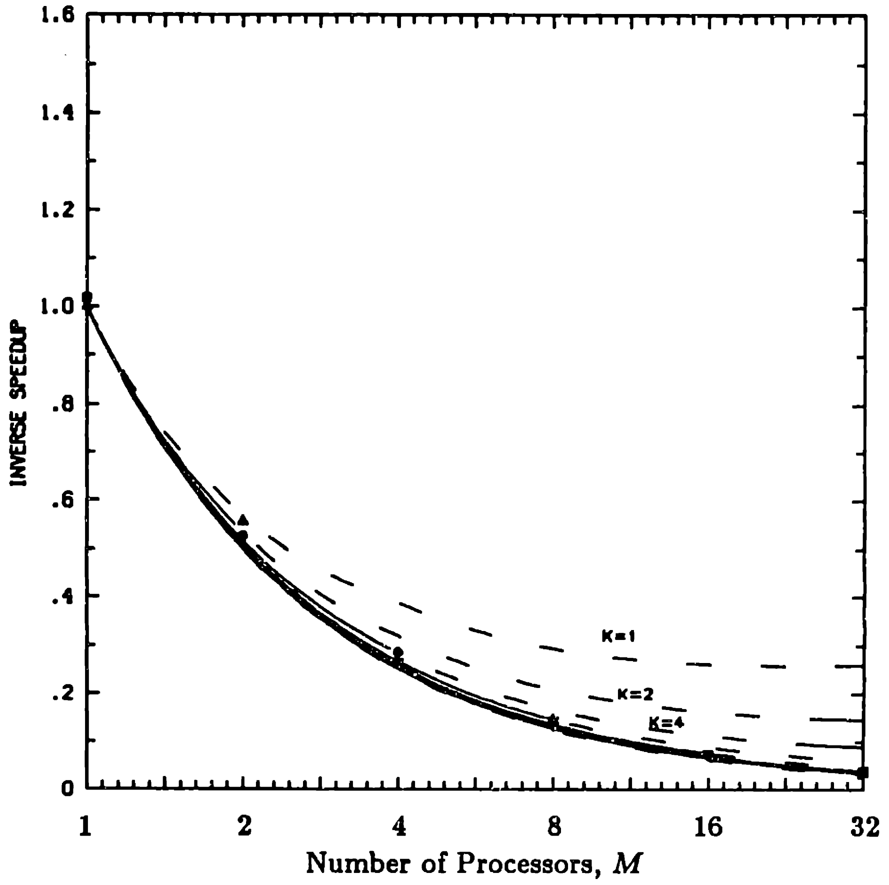


Figure 7.3: Inverse speed-up on the iPSC/2-VX for 250 A matrix iterations of the spectral element configuration in Figure 7.1 for problems defined by K=1,2,4,8,16,32. See previous figure for caption.

space to support six elements per processor.

We next investigate the effects of non-ideal mappings for the iPSC/2-VX system. For the first test, we use the partition $E_q=\{2q-1, 2q\}$, which places adjacent elements together, but requires that the nodes use non-nearest-neighbor communication. In this case τ_{d_s} (and hence τ) are increased by 0.1 seconds to 0.55 seconds (24.6 seconds), resulting in roughly a 0.5% decrease in speedup. In the second test, we use arbitrary partition E_q: $E_1, E_2, ..., E_{16}$ =$\{1,3\}$, $\{2,4\}$, $\{5,7\}$, $\{6,8\}$, $\{9,10\}$, $\{11,13\}$, $\{12,14\}$, $\{15,16\}$, $\{17,18\}$, $\{19,21\}$, $\{20,22\}$, $\{23,24\}$, $\{25,26\}$, $\{27,28\}$, $\{29,30\}$, $\{31,32\}$, which increases the amount of data flow through the network. The timings again show an increase of 0.1 seconds. As expected, the improved communication algorithms of the iPSC/2 significantly reduce the impact of non-optimal element to processor mappings. It is clear that the test problem is not a rigorous communication test for the iPSC/2 as the network is apparently never near saturation. However, these communication patterns are typical of those seen in general spectral element calculations. Furthermore, given the high performance of the vector processors, the communication rate is sufficiently high to maintain a balanced communication/computation work distribution.

We finish our analysis of the chain problem by plotting in Figure 7.4 the results in MFLOPS′$-e'$ space: MFLOPS′ is calculated as $(.1)\cdot\tau(K)_{\mu VAX}/\tau(K,M)$, where $\tau(K)_{\mu VAX}$ is the timing on the DEC μVAX, and .1 is the application-independent MFLOPS rating of the μVAX; e' is calculated from MFLOPS′ and the cost data summarized in Appendix A. It is seen that the hypercube MFLOPS′ $- e'$ point is indeed, interesting, in that it achieves near supercomputer performance at a fraction of the cost. To illustrate the importance of the

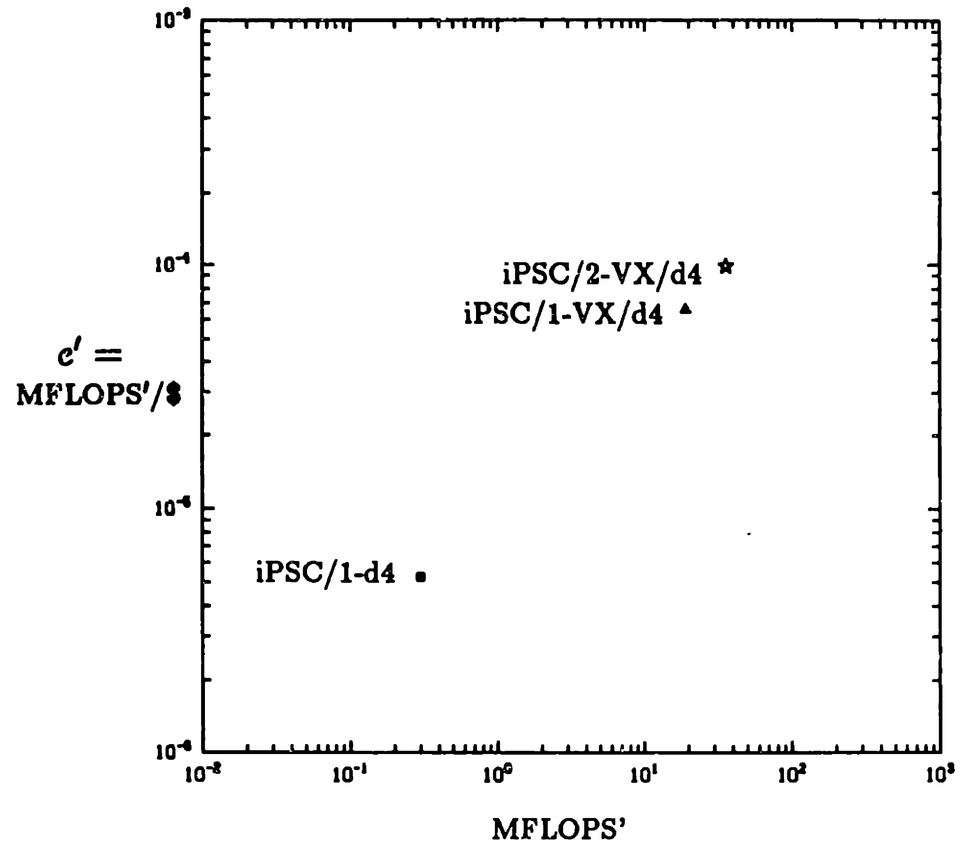


Figure 7.4: Computational resource efficiency for the $K = 32$ chain problem of Figure 7.1 for the iPSC/1, iPSC/1-VX, and iPSC/2-VX hypercubes.

MFLOPS$'$ – e' framework, we have also included the data point for the K=32, M=16 problem on the nonvector iPSC/1; although the parallel efficiency on the nonvector machine is close to unity, the nonvector machine is obviously uninteresting compared to its vector counterpart. This is due to the fact that the nonvector machine achieves high efficiency due to a decrease in σ brought about by an increase in δ, not a decrease in Δ. It is apparent from the nonvector iPSC/1 exercise that vectorization internal to the nodes is important to performance; the nested parallel/vector hierarchy of the spectral element discretization is ideally suited for the task.

7.2 Predicted Performance for Larger Systems

As an important point of analysis is to be able to predict as well as evaluate performance, we use our functional description of τ (7.1) to estimate the extent to which spectral element-distributed memory architecture coupling can be further leveraged through the addition of more processors. As the general fluid mechanics problem is currently far from being solved, we consider the interesting cases where K/M is fixed, i.e., problems of increasing size with an increasing number of processors. We note that in the following analysis the primary cost term is the $\log M$ term associated with the vector reduction operations in the conjugate gradient/direct stiffness algorithms and that the potcntial for contention is not addressed. Thus, we are presuming that the communication network is sufficiently rich to to cope with the direct stiffness communication for these larger problems; this shou!d not be a problem on hypercube multiprocessors having a dimension which is much greater than the spatial dimension of the physical problem of interest.

We use as our model constants in (7.1) the conservative figures a=10.0 sec, b=0.5 sec, and c=0.4 sec. This corresponds to a faster processing rate than currently obtained on the iPSC/2-VX, though not as fast as the fully vectorized code, and to a slightly slower communication rate than actually measured. Using the constant $a = 10.0sec$, the baseline processing rate for a single processor will be 2.78 MFLOPS.

The first issue we address is to find the maximum obtainable speedup as

$M \rightarrow \infty$, $K/M = const$. From (7.1) the speedup is:

$$S_r = \frac{M}{1 + \frac{M}{K}\left(\frac{b}{a} + \frac{c}{a}\log_2 M\right)} \tag{7.2}$$

$$= \frac{M}{1 + \alpha + \beta \ln M} \tag{7.3}$$

where we recognize that b/a, $c/a \propto \sigma$. Differentiating (7.3) with respect to M, we find that a minimum or maximum occurs at:

$$M_S = e^{\frac{\beta-\alpha-1}{\beta}} < 1 \quad . \tag{7.4}$$

Since S_r is growing for $M > 1$ there is no upper-bound on the maximum obtainable speedup for sufficiently small communication penalties (α, β), assuming that the logarithmic cost function is valid.

The second question we address is, "How many processors are required to obtain 1 GFLOPS sustained performance?" In general, the execution rate on M processors will be:

$$MFLOPS(M) = MFLOPS(1) \cdot S_r(M)\Big|_{K/M} \quad , \tag{7.5}$$

from which we can determine M_{1GF} simply by setting MFLOPS(M_{1GF}) = 1000. We consider the cases where K/M = 1,2,4,6, under a series of four different conditions:

i) using the assumed model constants a=10.0 sec, b=0.5 sec, c=0.4 sec;

ii) assuming a doubled clock rate $\Longrightarrow$ a=5.0 sec, b=0.5 sec, c=0.4 sec;

iii) assuming a doubled communication rate $\Longrightarrow$ a=10.0 sec, b=0.25 sec, c=0.2 sec; and finally,

iv) assuming that the direct stiffness effort (b/a) grows in direct proportion to K/M $\Longrightarrow$ a=10.0 sec, $b = 0.5K/M$ sec, c=0.2 sec.

K/M	i	ii	iii	iv
1	507	318	430	507
2	430	246	396	441
4	396	212	378	409
6	384	201	372	398

Table 7.4: Predicted number of processors, M, required to achieve 1 GFLOPS performance based on variations of the current spectral element - iPSC/2-VX coupling.

The results of this analysis are presented in Table 7.4.

It is clear from Table 7.4 that 1 GFLOPS performance should be obtainable with on the order of 512 processors ($D = 9$) for the current values of (δ, Δ) for the spectral element - iPSC/2-VX coupling. We note that as the work per processor increases, $K/M \to \infty$, the number of processors required to achieve 1 GFLOPS is bounded from below by 1000 MFLOPS/MFLOPS($M = 1$) = 360. In addition, we see from column (ii) that substantial gains can be made by increasing the processing rate, implying that the current implementation is not communication bound. Our final analysis (iv) addresses the issue of contention as we now allow for increased communication (though not logarithmic with M) with increasingly large K/M. It is seen that performance is not greatly impacted by this additional communication overhead.

The third question we address is the potential performance on 1024 processors. From (7.2) we find:

$$\begin{aligned} MFLOPS(M=1024, K/M=1) &= 2324 \\ MFLOPS(M=1024, K/M=6) &= 2648 \ . \end{aligned} \tag{7.6}$$

Assuming that system cost scales with the number of processors, the 1024 node system will achieve a respectable resource efficiency of $e' = 11.9 \times 10^{-5}$ MFLOPS/\$, based upon the cost of the iPSC/2-VX/d5. Note that we have substantial progress to make before reaching TeraFLOPS performance.

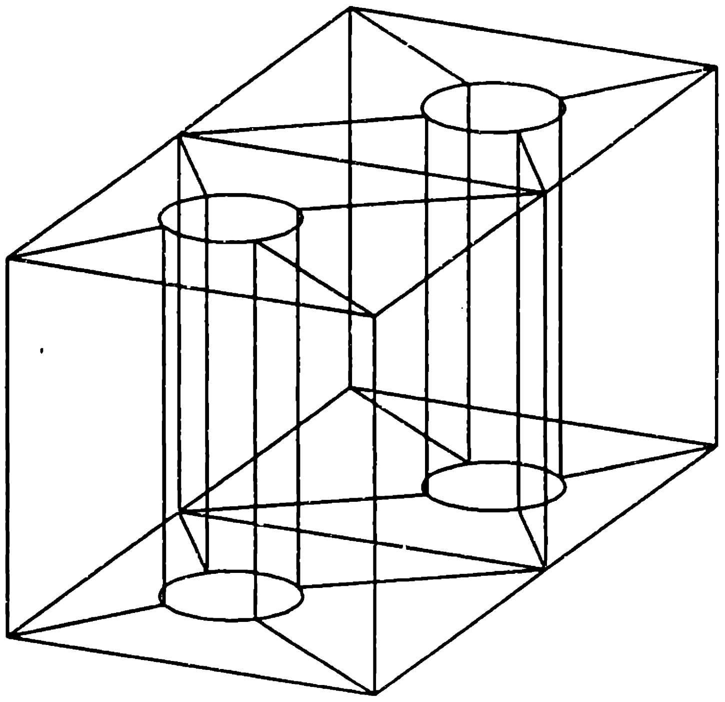
The above analyses are quite general in that they are readily extended to more flexible non-conforming spectral element discretizations [35,39]. The added complexity of such methods primarily results in increased data traffic for low dimension data structures, edges and vertices, implying that properly implemented non-conforming discretizations will incur additional $\log M$ type communication during the direct stiffness summation. If we assume that the $\log M$ term, c, is increased by a factor of 2, we find that 1 GFLOPS performance will be obtained on $M = 650$ processors.

Clear progress in terms of achievable CPU cycles is apparent. However, the truly important performance measures of a particular algorithm-architecture coupling are the solutions and solutions-per-dollar which can ultimately be obtained. We therefore need to reassess our current spectral element Navier-Stokes algorithms in order to understand what are potentially limiting factors in scaling to these very large simulations. One area which needs to be addressed is the implementation of order-independent iterative solvers (e.g. multi-grid [3,34]) which maintain a fixed rate of convergence independent of the problem size. Efficient parallel multi-grid implementation is significantly more complex than either its serial counterpart or parallel conjugate gradient iteration due to non-vanishing communication on the coarse grids ($\lim_{\epsilon \to 0} \Delta(\epsilon) \neq 0$). Another area requiring attention is the explicit treatment of the convective term in the temporal discretization of the Navier-Stokes equations. Currently, increasing spatial resolution imposes stringent restrictions on the time step size due to the Courant stability criteria. To circumvent this problem, implicit spectral element characteristic methods have been recently investigated and a topology preserving (deforming element) scheme shows promise as more efficient means of time advancement [24].

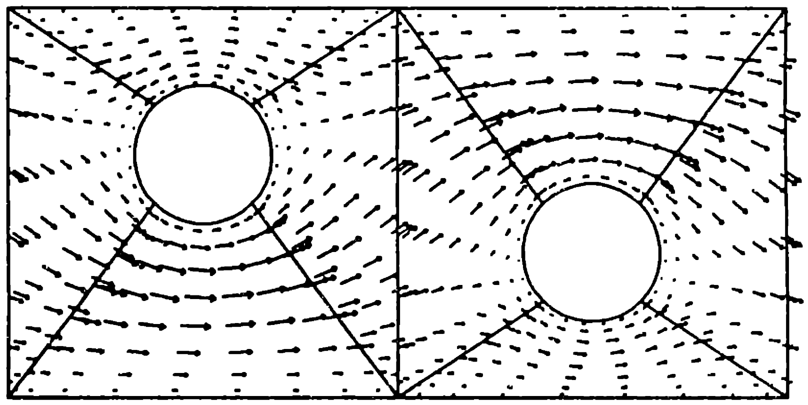
7.3 Economic Performance Analysis

As a major point of this work is the development of general methods for "real" fluid flow problems, we conclude the analysis with the solution of a full three-dimensional Stokes problem. We consider the geometry of Figure 7.5a, with periodic boundary conditions in the flow direction, and no-slip boundary conditions on all solid walls. The discretization parameter is taken to be $h = (K = 32, N = 10)$, and the problem is solved on $M = 16$ processors. The S_{2intra} and S_{1inter} strategies are pursued so as to achieve a nearest-neighbor mapping, thereby minimizing λ_1, λ_2, and λ_3. The element to processor mapping is an extension of that depicted in Figure 5.1; pairs of vertically adjacent elements are placed on each processor, and a copy of the mapping is repeated on processors $P_8, ..., P_{16}$ to effect nearest neighbor communication between the upper and lower levels of elements. The results of the calculation are shown in Figure 7.5b in terms of the velocity field.

On the basis of timings similar to those described for the "chain" problem



(a)



(b)

Figure 7.5: (a) Computational domain consisting of 32 spectral elements (8 are shown for clarity) for the steady Stokes problem of flow past two cylinders in a duct. (b) Velocity vectors at the mid-plane of the domain.

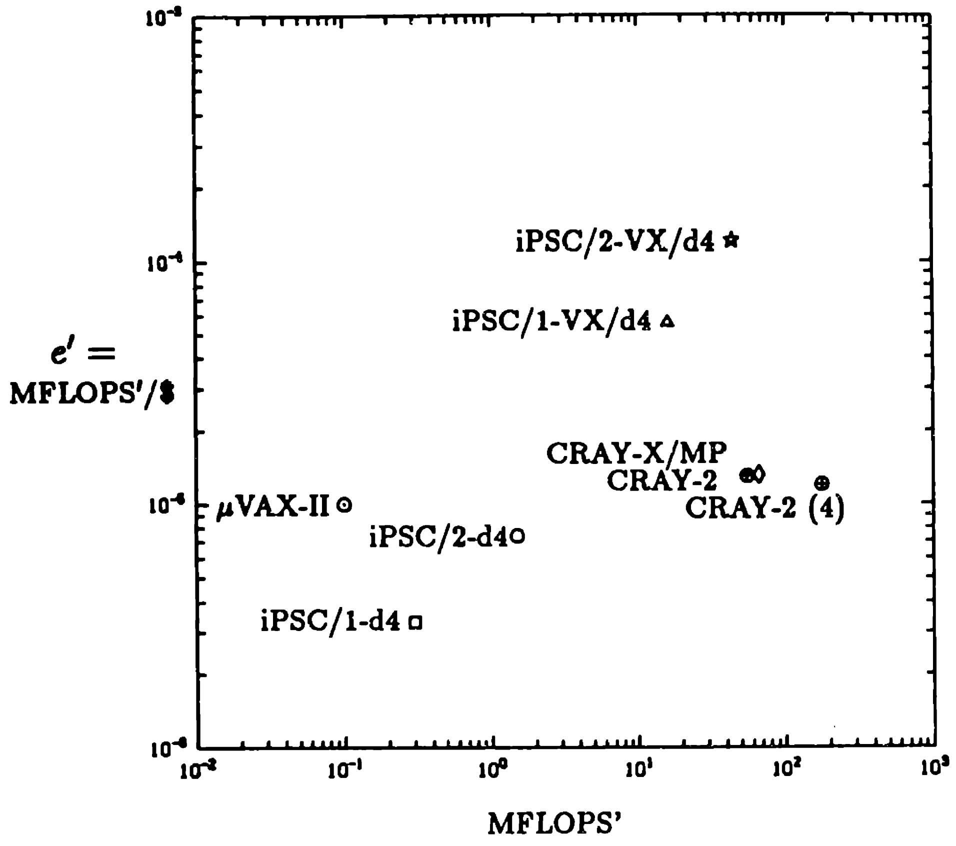


Figure 7.6: Measured computational resource efficiency, e', for the Stokes problem of Figure 7.5.

we plot in Figure 7.6 the MFLOPS$'-e'$ points for this calculation on the μVAX, CRAY X-MP, CRAY 2, IPSC/1-VX/d4, and iPSC/2-VX/d4 computers; the actual timing data is given in Appendix B. The iPSC/2 constitutes a significant improvement over the iPSC/1, primarily due to communication speedup through hardware and architecture, but also partially due to increased scalar speed. The iPSC/2 calculation runs at a parallel efficiency of $\eta = .7$ compared to the iPSC/1 calculation which operates at $\eta = .3$; note how the iPSC/1 and iPSC/2 points are distanced in Figure 7.6 as opposed to Figure 2.

It is instructive to compare the nonvector iPSC/1 and the iPSC/2-VX. The nonvector iPSC/1 is clearly not operating at M_{opt}, as the parallel efficiency is effectively unity. However, even at M_{opt}, the e' of the nonvector iPSC/1 will not be competitive with that of the iPSC/2-VX (or iPSC/1-VX); vectorization represents a significant improvement in performance with only a marginal increase in cost. Note also that from the arguments of Section 5.2, (5.11), even at M_{opt} the nonvector iPSC/1 will be inferior to the iPSC/2-VX due to the fact that $\Delta(1)$ for the iPSC/2 is significantly smaller than for the iPSC/1, and $\sigma(1)$ for the iPSC/2-VX is significantly larger than for the iPSC/1.

The results of Figure 7.6 indicate that properly designed numerical algorithms can solve real problems on parallel processors at serial-supercomputer speeds, using only a fraction of serial-supercomputer resources. Full unsteady Navier-Stokes calculations are presented in the next chapter which illustrate that parallel processing is an effective tool for fluid mechanics analysis.

Chapter 8
Parallel Navier-Stokes Computations

We conclude with the presentation of several example calculations which illustrate the generality of the parallel spectral element implementation, its capabilty in handling laminar and transitional flows, and its utility as a fluid mechanics analysis tool. Many of the examples are preliminary investigations into various classes of problems and in such cases represent only a single data point for a more exhaustive parameter study. However, the examples are instructive in providing insight to the fundamental structure of each flow. The presentation below roughly follows from low Reynolds number, two-dimensional flows to moderate Reynolds number ($Re \simeq 3000$), three-dimensional simulations.

The typical run time for the problems presented ranged from 4 to 40 hours on anywhere from 2 to 32 nodes, depending on the problem size. For the higher Reynolds number three-dimensional flows, the limitation most often encountered was insufficient resolution to capture all the scales which were present. In some cases resolution was limited due to the lack of non-conforming discretizations which allow for local refinement without unnecessary far field refinement. In other cases the Reynold number was sufficiently high that flow in fully separated regimes became chaotic and polluted the solution elsewhere in the domain. Of course, the resolution problem can be remedied simply by increasing the discretization parameters (K, N); memory on the iPSC/2-VX (4 Mbytes

+ 1 Mbyte vector) was sufficient as the largest problem used only half of the available memory on the 32 node system. The current limitation is in fact post-processing of data; high-speed vector graphics computers and software together with efficient means of transferring and storing data are required for analysis of large calculations.

8.1 Steady Stokes Flow

The first calculation is a steady three-dimensional Stokes flow consisting 312,000 degrees-of-freedom ($K = 128$, $N = 10$). The physical problem is that of a spiral groove thrust bearing with ambient pressure at the inner and outer radii. The elemental geometry in Figure 8.1 is comprised of two levels of elements stacked upon one another. The lower layer corrsponds to the bearing grooves which have a depth of .04, normalized with respect to the outer radius. The upper layer corresponds to the gap between the thrust plates having a depth of 0.01. The flow is driven by imposing counter-clockwise plane rotation on the upper bearing surface. Lift is generated by viscous pumping in the channels, and the computed lift is in good agreement with Reynolds equation solutions for similar configurations [50]. The main point of this calculation is to illustrate the scalability of the parallel algorithms described previously. The Stokes problem was run on a 64 node Intel iPSC/2-VX at a sustained execution rate of 160 MFLOPS for the duration of the 16 minute solution time. Previous calculations on a 16 node machine had achieved 44 MFLOPS [49], indicating roughly a ten percent degradation in performance per processor as the number of processors was increased from 16 to 64. As access to the 64 node machine was limited, extensive parameter studies were not feasible; consequently, it is not clear whether

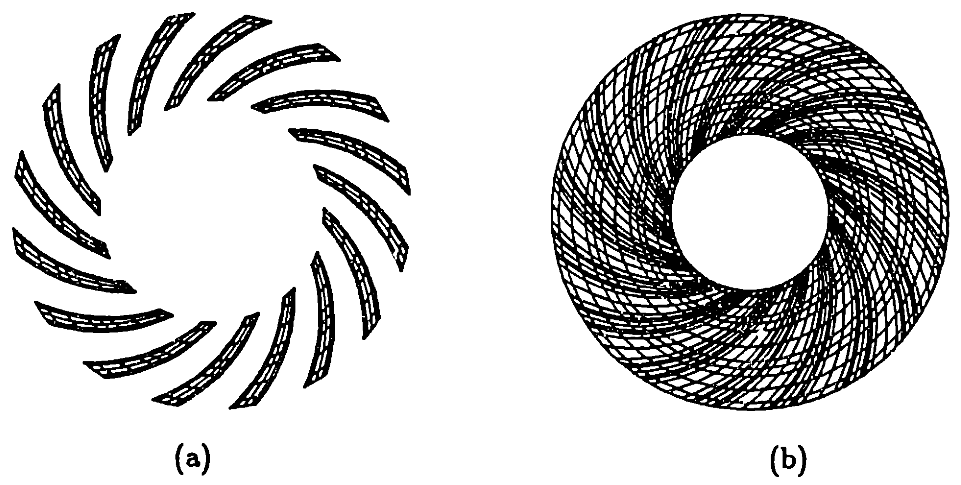
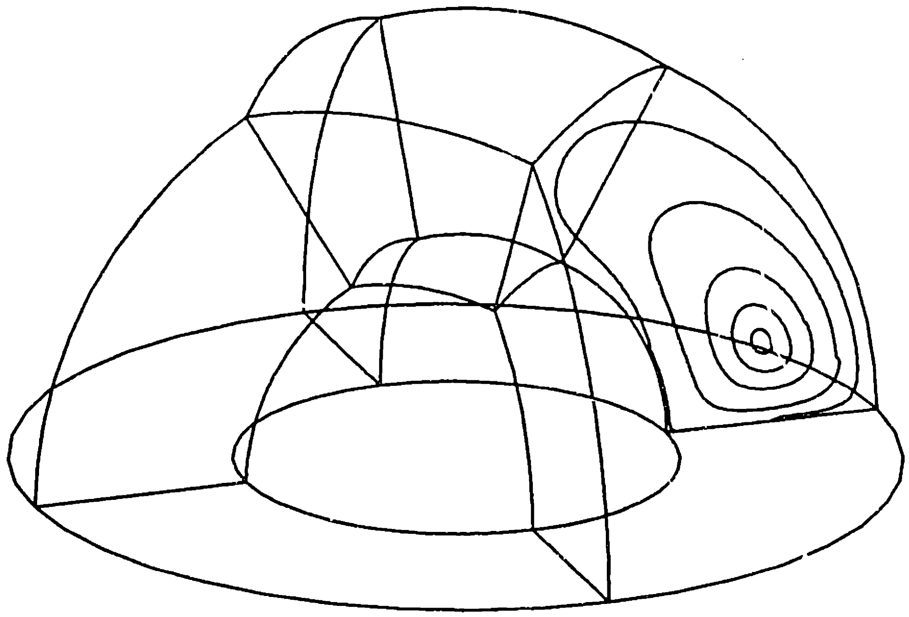


Figure 8.1: Mesh geometry for spiral groove thrust bearing. Line intersections indicate location of Gauss-Lobatto quadrature points for $N = 5$: (a) 32 element lower level comprising inlet grooves, (b) thin gap region consisting of 96 elements.

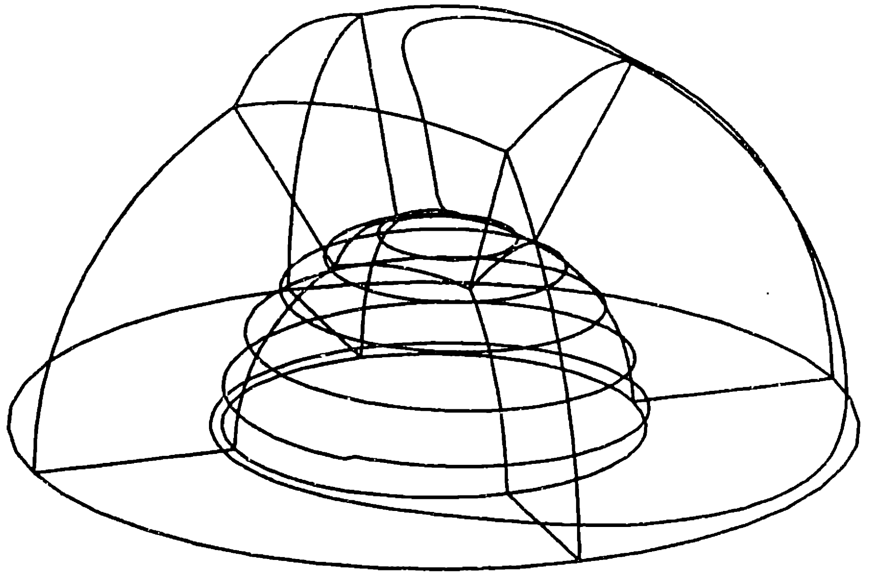
the observed performance degradation is due solely to the $\log M$ term in equation (5.12) or to variance resulting from non-optimal mapping. In either case, the performance on 64 nodes is quite good and indicates that this particular algorithm/architecture coupling can be leveraged further with the addition of more processors.

8.2 Rotating Flows

Our second example is a preliminary calculation which is part of an ongoing investigation into Couette flow between concentric spheres [51]. Although such flows are nominally two-dimensional (axisymmetric), it is well known that three-dimensional modes persist in the Taylor vortices near the equator [52] as well



(a)



(b)

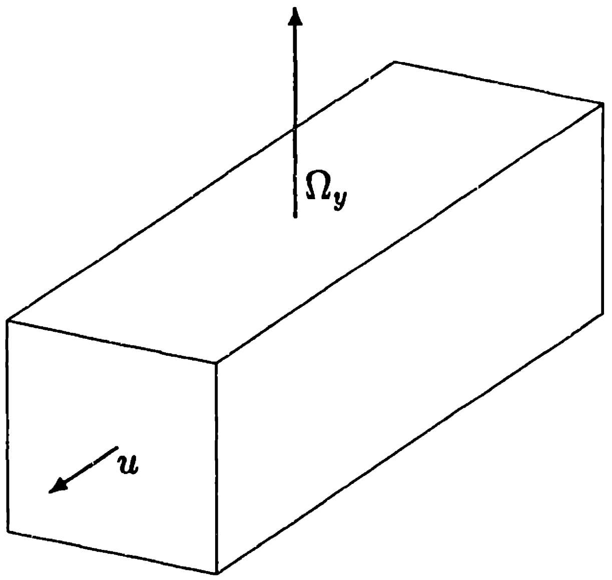
Figure 8.2: (a) Meridional projection of steady state streamlines for flow in a spherical gap at $Re = 25$, based on inner sphere rotation rate and diameter. (b) Three-dimensional streamline reveals a complex flow structure even at this low Reynolds number.

as in the vortex breakdown at the poles, and it is therefore of interest to pursue fully three-dimensional calculations. Our particular interest in this problem is to analyse the vortex breakdown at the poles which occurs at sufficiently high Reynolds in the large gap case. The present work was initiated by P. Bar-Yoseph.

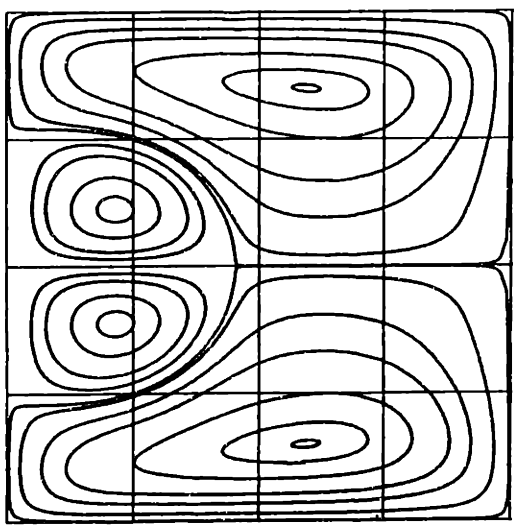
We consider an inner sphere of radius $R_i = 0.5$ and an outer sphere of unit radius, the inner sphere rotating with an angular velocity Ω. The Reynolds number is taken to be $Re = \Omega R_i^2/\nu = 25$. Invoking symmetry boundary conditions at the equator, $\frac{\partial u}{\partial z}, \frac{\partial v}{\partial z}, \frac{\partial p}{\partial z} = 0$, $w = 0$, the computational domain becomes a hemispherical shell consisting of five elements ($N = 10$) as shown in Figure 8.2a. As the inner sphere is set into motion, centrifugal forces drive fluid towards the outer sphere at the equator, and the resultant pressure gradient forces the flow along the inside of the outer shell up towards the poles; a large single vortex system results in each hemisphere. The full three-dimensional complexity of the steady state motion is illustrated in the streamline trace shown in Figure 8.2b.

We comment that this flow can in fact be computed much more efficiently using an axisymmetric formulation. However, such a simulation would fail to reveal three-dimensional modes. These initial results were computed on a four node Intel iPSC/2-VX hypercube and were severely constrained in spatial resolution due to lack of memory. Consequently, the flow is at a Reynolds number far below that required to generate vortex breakdown at the poles. Nonetheless, the calculation serves to illustrate the geometric flexibility of the three-dimensional isoparametric spectral element formulation.

In conjunction with the vortex breakdown project, we have examined the problem of flow in a rotating duct for varying aspect ratios and flow parameters, Ek $\equiv \nu/L^2\Omega$, and Ro$\equiv U/L\Omega$, the Ekman and Rossby numbers, respectively.



(a)


(b)

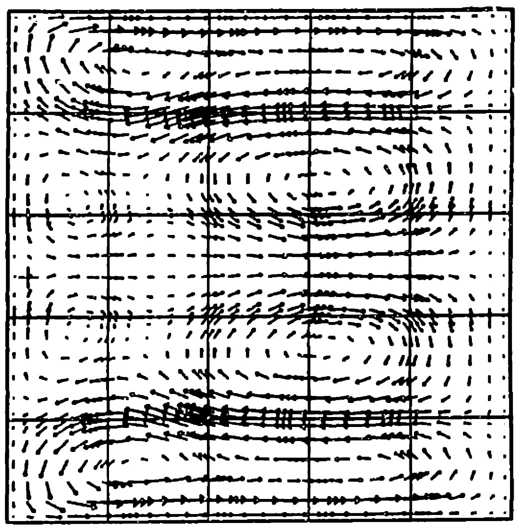
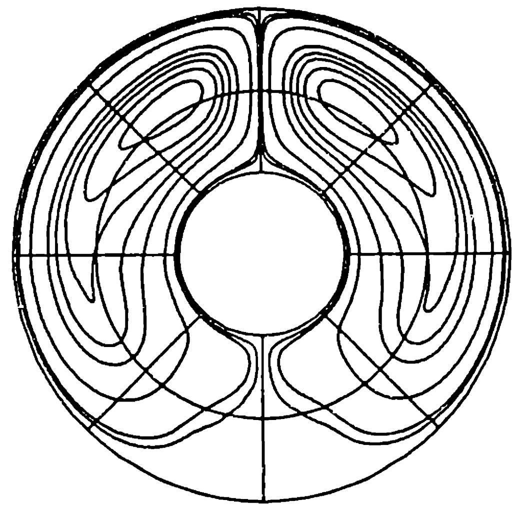

(c)

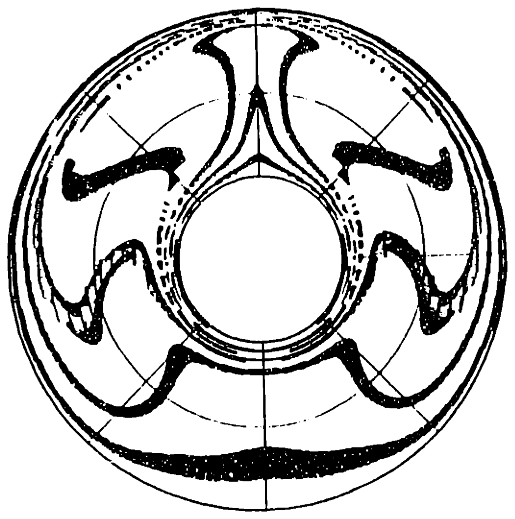
Figure 8.3: Fully developed flow in a rotating channel: (a) problem geometry, (b) steady state streamlines, and (c) in-plane velocity vectors at a time of 1.0 for Ek $= \nu/L^2\Omega = .01$ and Ro$= U/L\Omega = 3.0$. (Courtesy P. Bar-Yoseph)

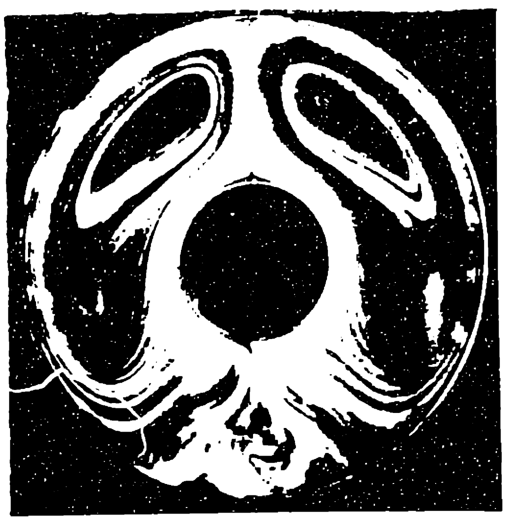
A typical geometry is shown in Figure 8.3a, in which a duct of unity aspect ratio is rotating about the vertical axis at a constant rotation rate Ω. The flow is assumed to be fully developed in the z-direction, that is, $\frac{\partial}{\partial z} = 0$ for all flow variables, save a constant pressure gradient. The steady state solution is reached by time integration from an initial condition corresponding to fully developed flow for a non-rotating duct. We present in Figures 8.3b and 8.3c the steady state streamlines and velocity vectors for the case of Ek=.01 and Ro=3. The velocity vectors and time history are in good agreement with numerical results presented in [53].

8.3 Natural Convection

We next consider natural convection between horizontal concentric cylinders for a Grashof number, Gr= $\frac{g\beta\Delta T D^3}{\nu^2}$ = 120,000. Here g is the acceleration of gravity; β, the coefficient of thermal expansion of the medium, which in this case is taken to be air; $\Delta T = 14.5°C$, the temperature difference between the inner and outer cylinders; D the outer cylinder diameter; and ν the kinematic viscosity. The inner cylinder diameter is $D/3$. The resultant Reynolds number based on cylinder diameter, maximum flow velocity, and kinematic viscosity is $Re \simeq 100 - 200$. The results of the spectral element solution ($K = 16, N = 10$) are shown in the form of steady state streamlines and temperature contours in Figures 8.4a and b, respectively. The qualitative agreement with the experimental results presented in Figures 8.4c and d (from [54]) is excellent.


(a)


(b)


(c)

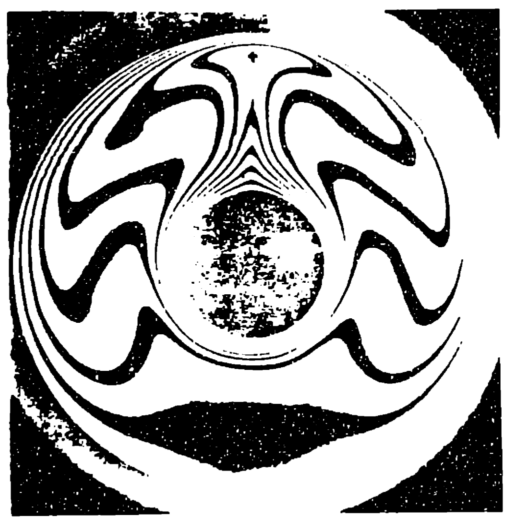

(d)

Figure 8.4: Natural convection for air at a Grashof number of Gr = 120,000. The temperature of the inner cylinder is 14.5°C above the outer cylinder. Numerical steady state streamlines (a) and temperature contours (b) compare well with experimental results (c),(d) from [54]. Discretization parameter is $h = (K = 16, N = 10)$.

8.4 Grooved Channel Flows

The study of flow in grooved channels is of interest because such geometries promote transition from steady to unsteady states in a controlled, supercritical manner at low Reynolds numbers, in marked contrast to the uncontrolled subcritical transition which occurs in standard plane Poiseiulle flow. Consequently, they serve as good test problems for comparison with classical linear stability theory and have many modal solutions which are of interest in their own right [55,56]. Because the solutions just above the critical transition Reynolds number are steady periodic in time and are fixed in space by the presence of inhomogeneities in the geometry, the grooved channel flows are more amenable to analysis and control than their chaotic plane Poiseiulle flow counterparts [57].

We present two representative calculations which have geometry identical to that recently studied by Amon and Patera [56]. The first is the two-dimensional problem depicted in Figure 8.5a. All lengths are non-dimensionalized with respect to the channel half-height, h. The domain length is $L = 5.0$, the groove width and height are $w = 3.0$ and $a = 1.68$, respectively. Periodicity is imposed at the channel entrance and exit and the flow is driven by a constant mean pressure gradient. Although Amon and Patera's calculations were for the case of constant mass flux, similar behavior is exhibited in the two problems. Results for Reynolds number $Re = \frac{3}{2}Uh/\nu = 350$ are presented. The steady periodic nature of the flow is evidenced by the time history of the x-component of velocity at the point $(x = 1.42, y = 1.58)$ in Figure 8.5b. Instantaneous streamlines in Figure 8.5c reveal Tollmein-Schlicting waves amplified by interaction of the mean flow with the driven cavity flow in the groove.

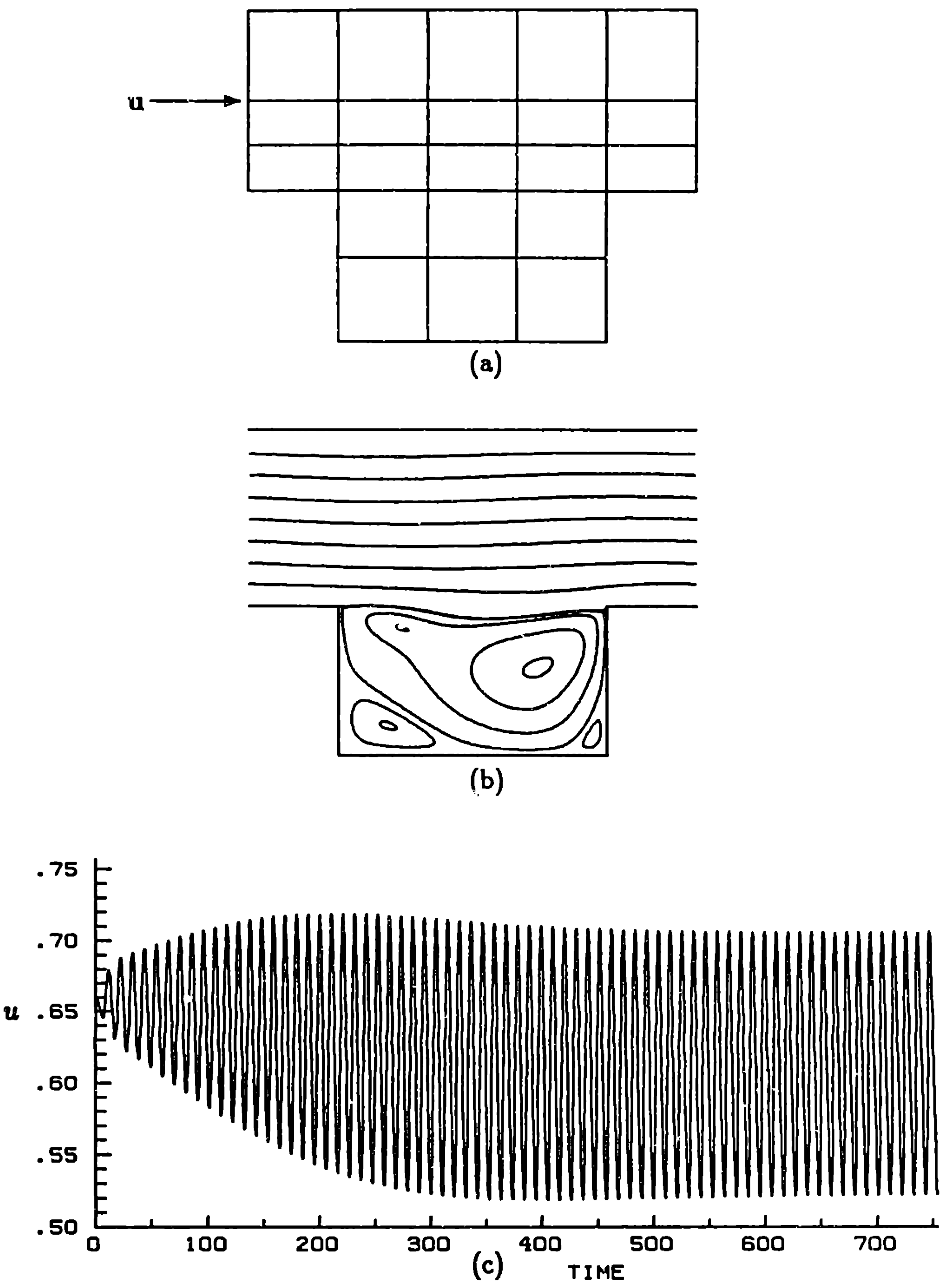


Figure 8.5: Groove channel geometry with spectral element discretization (a). Instantaneous streamlines clearly show wavy structure of the flow associated with excited Tollmein-Schlicting waves (b). Time history of streamwise velocity component near the channel center shows the existence of a steady periodic state (c).

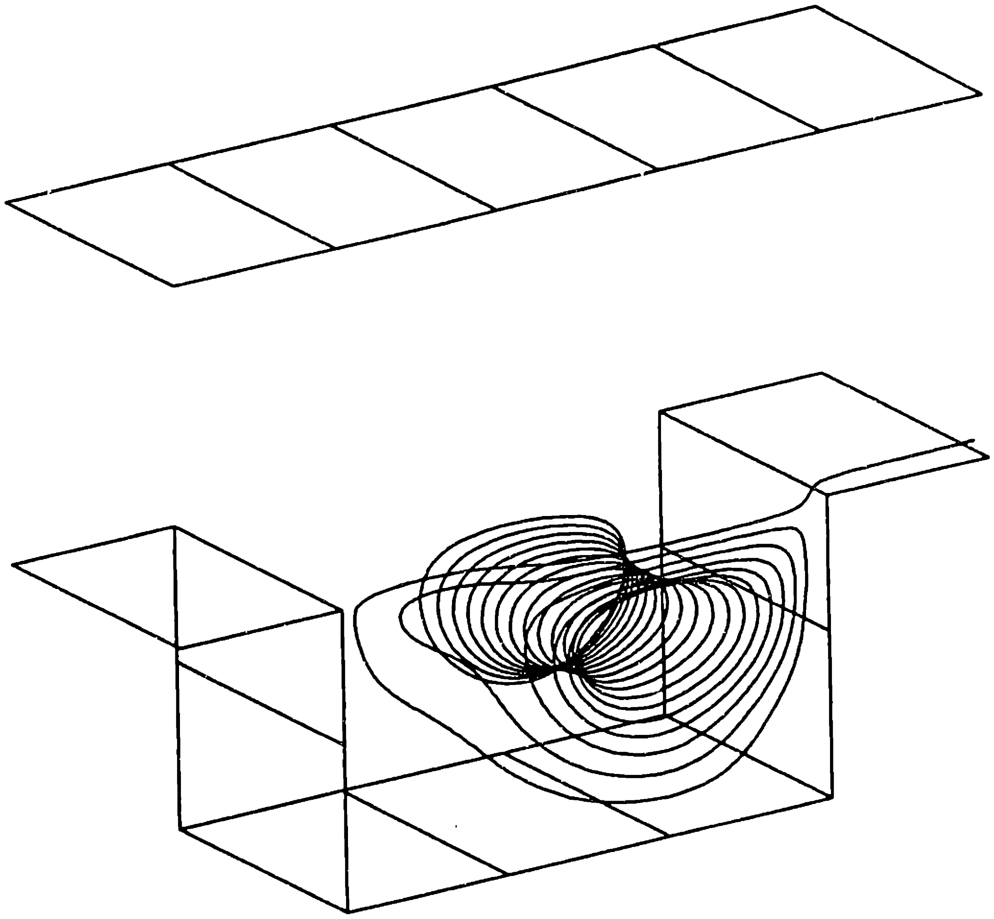


Figure 8.6: Streamline plot for grooved channel simulation reveals presence of three-dimensional (transverse) mode under steady periodic conditions.

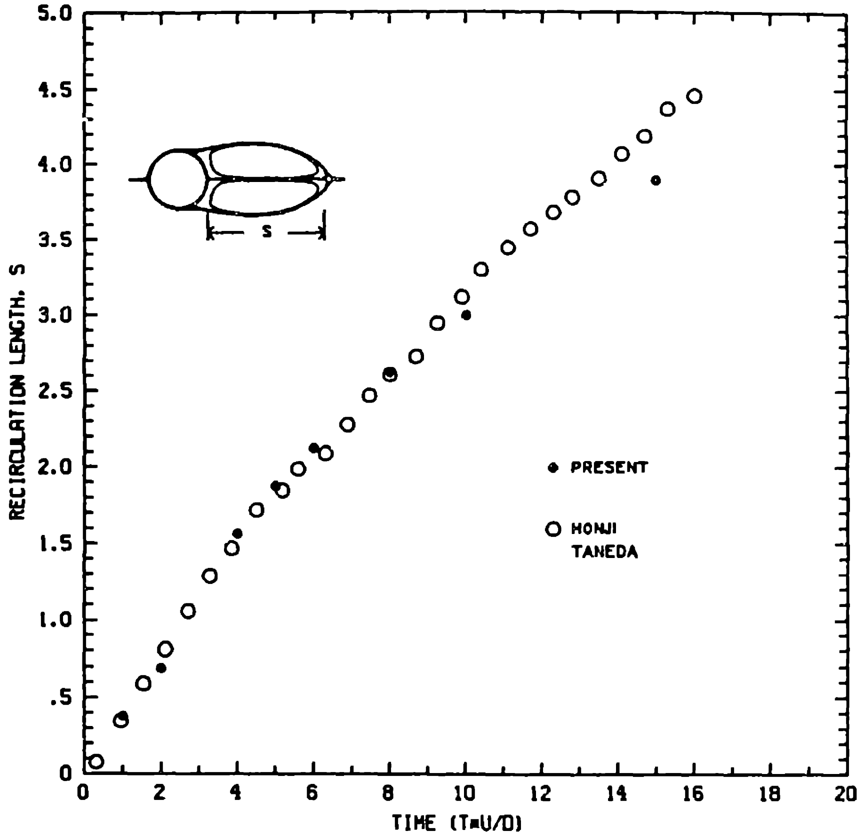
Three-dimensional simulations are being carried out to investigate the nature of the transition from two- to three-dimensional motion. Amon has investigated three-dimensional transition using linearized Fourier expansions in the cross-channel direction, and has found $Re_{crit,3D} = 340$ for the channel geometry of the previous calculation with constant volume flux. Using a cross-channel width $W = \frac{\pi}{2}$ with symmetry boundary conditions $(\frac{\partial \vec{u}}{\partial \hat{n}} = 0, \vec{u} \cdot \hat{n} = 0)$ at $z = 0$ and $z = \frac{\pi}{2}$, we have found sustained three-dimensional modes for full Navier-Stokes calculations at $Re = 320$ in the presence of a constant mean pressure gradient. The streamline pattern in Figure 8.6 clearly shows the presence of transverse velocity components. History traces similar to those of Figure 8.5c indicate the existence of a steady periodic solution.

8.5 External Flow Past a Cylinder

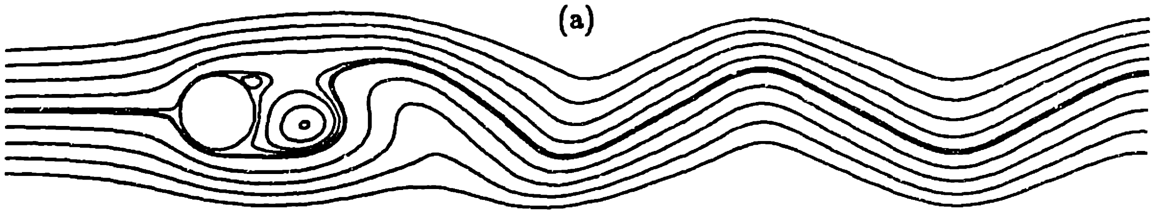
We turn now to a series of calculations for the classical problem of external flow past a cylinder. This problem serves as an excellent test problem because of the extensive literature available (see for example [58]) and the relatively well defined transition states which persist across a wide range of Reynolds numbers.

We first examine startup flow past a cylinder of diameter D at a Reynolds number of $R = U_\infty D/\nu = 100$. The cylinder is initially in quiescent fluid at $t = 0$, with an external uniform flow U_∞ imposed abruptly at $t = 0^+$. The problem is treated with the spectral element Navier-Stokes discretizations based on consistent approximation spaces for the velocity and the pressure, and the Uzawa conjugate gradient-based iterative solvers [1]. Figure 8.7a shows the good comparison between the numerical prediction of the recirculation zone length at early times with the experimental observations of [59]. At later times the familiar unsteady von Karman vortex street forms as shown in Figure 8.7b. The startup calculation was performed on an eight node iPSC/1-VX hypercube.

The next problem is again the cylinder startup problem, but now at a Reynolds number $R = 1000$. This calculation is based on a fractional step method with conjugate gradient solution of the elliptic subproblems [60]. Figure 8.8a shows the spectral element discretization of the computational domain



(a)



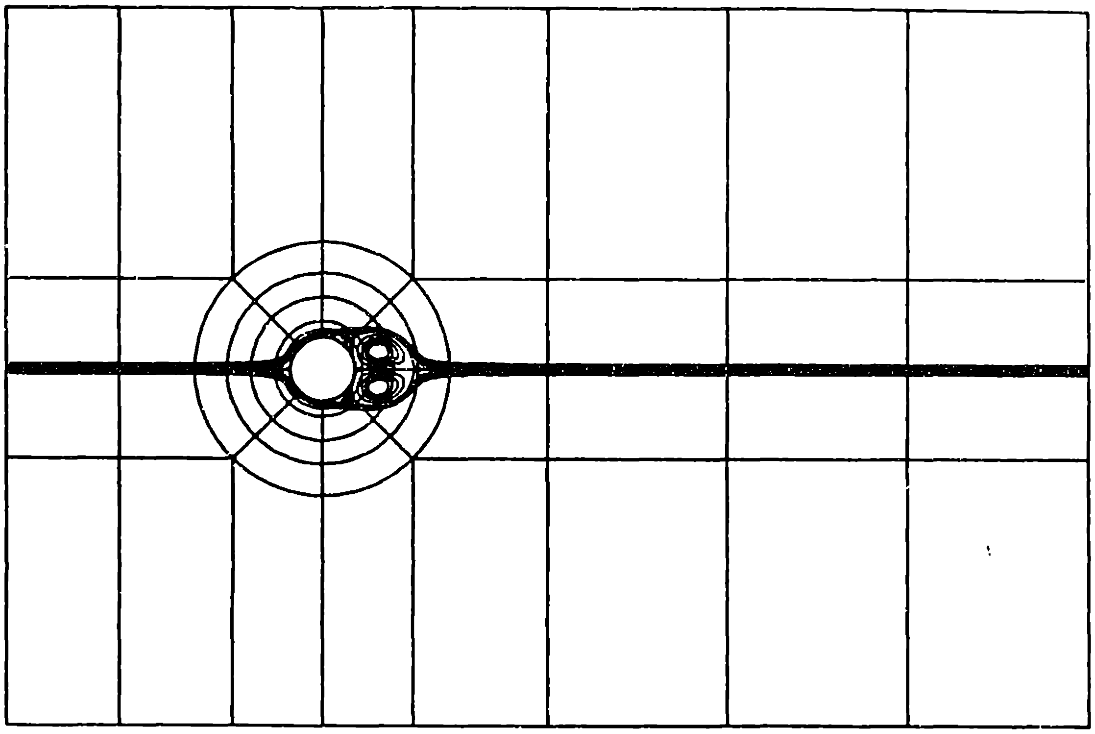
(b)

Figure 8.7: Startup flow past a cylinder at a Reynolds number $R = U_\infty D/\nu = 100$. (a) Comparison of early time nondimensional recirculation zone length with experimental measurements of [59]. (b) Instantaneous streamlines showing the von Karman vortex street at a nondimensional time of 110.

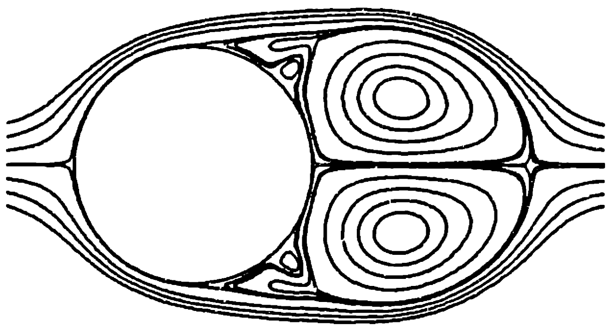
consisting of $K = 52$ elements, together with instantaneous streamlines at an early non-dimensional time of $U_\infty t/D = 3$. Figure 8.8b shows a more detailed close-up of the vortex structure which forms behind the cylinder; this result is in excellent agreement with the experimental results shown in Figure 8.8c [54].

One very interesting phenomena observed in cylinder and general bluff body flows is the rapid reduction in drag which occurs when the Reynolds number exceeds a critical value Re_c. The so called "drag crisis" results from a dramatic change in the character of the flow in which increased momentum transfer between the boundary layer and free stream allows the flow near the body to overcome the adverse pressure gradient, resulting in delayed separation and greater pressure recovery on the downstream surfa~e. Traditionally, the drag crisis has been associated with the onset of turbulence in the boundary layer at $Re_c \simeq 10^5$. However, numerical evidence has shown that it may be possible to induce the drag crisis through excitation of low Reynolds number two-dimensional Tollmien-Schlicting waves. Kawamura and Kuwahara have observed a drag crisis for a cylinder with a rough surface at $Re = 40,000$ [61], and Spalart has computed a drag reduction at $Re \simeq 10^5$ using vortex methods [62]. Both of these results were based upon two-dimensional calculations, indicating that turbulence need not be the only means of producing the momentum transfer required to overcome the adverse pressure gradients.

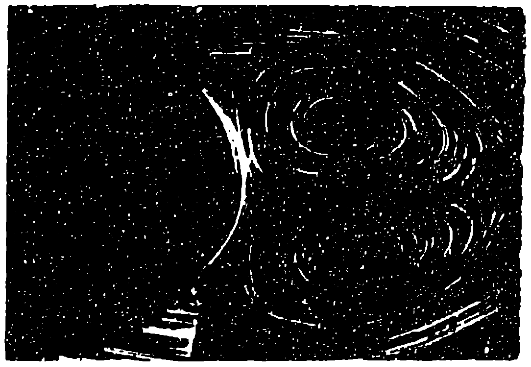
Armed with such evidence, we have embarked upon an investigation of the drag crisis for two-dimensional flows past a cylinder. In particular, we intend to combine the results of the grooved channel flow simulations of the previous section with the cylinder calculations and trigger Tolmein-Schlichting waves in the cylinder boundary layer by placing grooves around the perimeter. We have



(a)



(b)



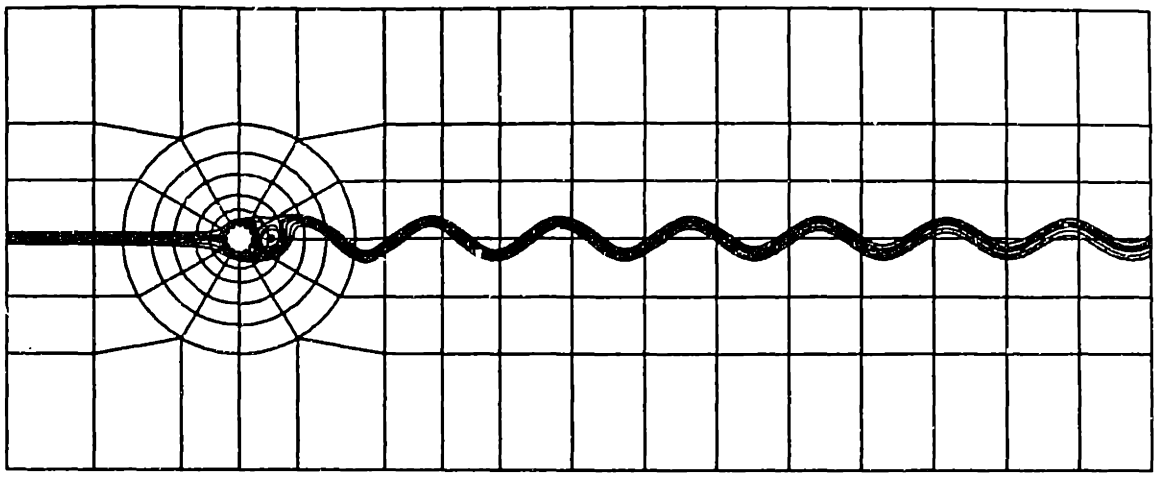
(c)

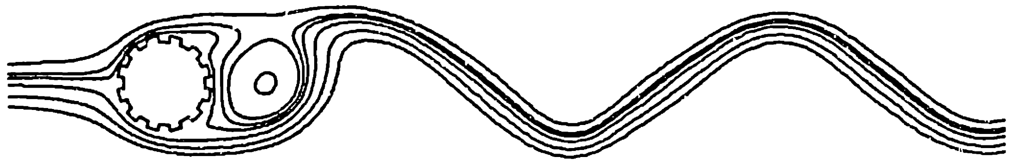
Figure 8.8: Startup flow past a cylinder at a Reynolds number $R = U_\infty D/\nu = 1000$. (a) Spectral element mesh ($K = 68$, $N = 10$) and instantaneous streamlines at a nondimensional time of $U_\infty t/D = 3$. (b) Enlarged view reveals small secondary vortices which are also observed experimentally (c), [54].

carried out one preliminary calculation at $Re = 200$ and show the results in the form of instantaneous streamlines in Figure 8.9. The full computational domain and unsteady von Karman vortex street is seen in Figure 8.9a. The size and spacing of the grooves is shown in Figure 8.9b. The range of resolved scales in this calculation is evident in 8.9c, which reveals the presence of numerous small eddies in the grooves. Note that the character of the groove flows changes from viscous, Stokes flow, to inertial, separated flow in moving from the forward stagnation point to the cylinder apex.

The grooved cylinder problem requires extensive exploration of the four-dimensional parameter space, $(Re, a/D, w/D, L/D)$, which are the the Reynolds number and the non-dimensional geometric parameters from the grooved channel flows. Because the flow is characterized by a broad range of spatial scales, and hence disparate time scales, each simulation requires many thousands of time steps to reach a steady periodic state. Several measures can be taken to reduce the computational effort, the most notable of these being the implementation of non-propagating grid refinement techniques which would allow for fine scale resolution near the cylinder/groove boundary layers without having unnecessary refinement in the far field. Recently developed non-conforming spectral element discretizations [35] allow for such flexibility and will be instrumental in continuance of this investigation.

It is clear that intelligent selection of the geometric parameters is required in order to optimize the search for the minimum achievable Re_c, and that such selection must be based upon extensive understanding of boundary layer interactions with grooves, and boundary layer development about cylinders. Such understanding results from studies similar to those described in the previous


(a)


(b)

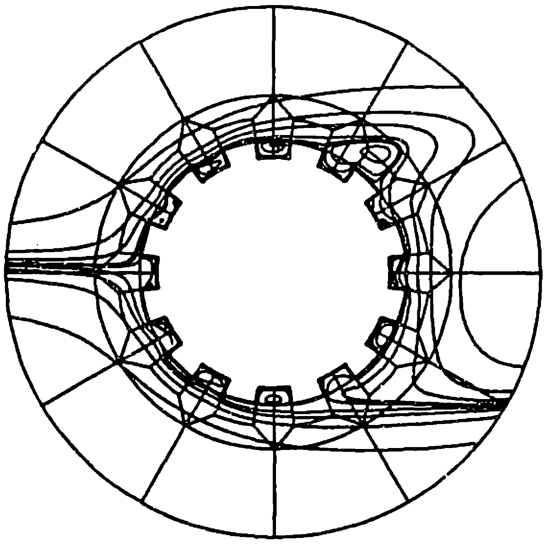

(c)

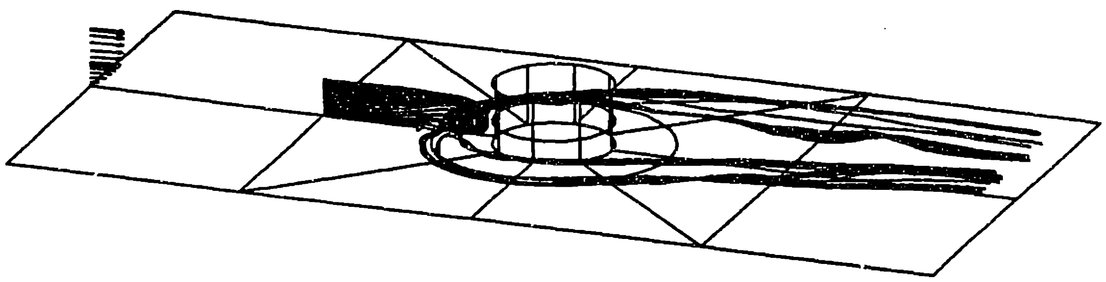
Figure 8.9: Flow past a grooved cylinder at $Re_D = 200$: (a) vortex street and spectral element discretization; (b) and (c) enlarged views reveal broad range of scales present in this flow. Viscous and inertial eddies are evident in the grooves.

section and at the beginning of this section. However, the validity of a conjectured drag reduction will ultimately be established only through numerous fully resolved simulations. Such extensive computations are made possible by having a local, interactive supercomputer environment, and made affordable by having the low cost computing power delivered by distributed memory parallel processing.

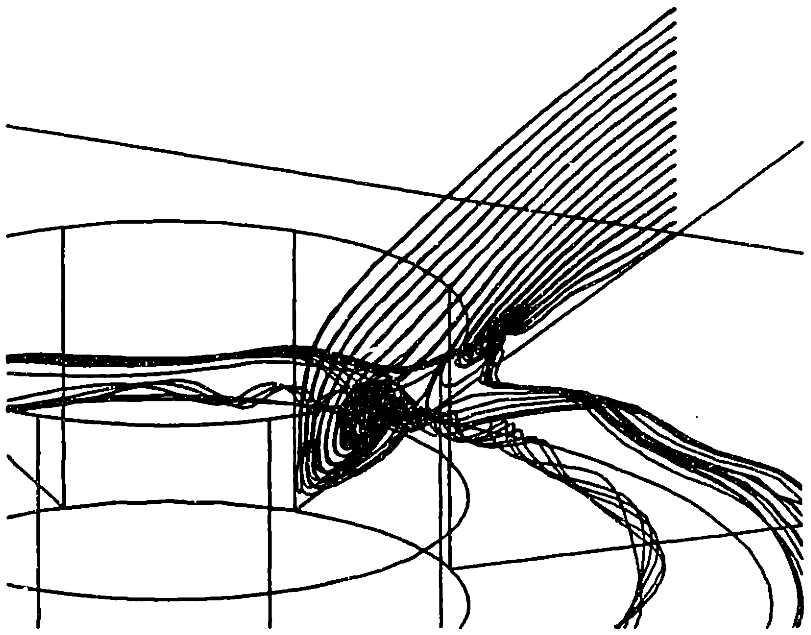
8.6 Three-Dimensional Horseshoe Vortices

We turn now to a series of three-dimensional calculations, in fact, to a series of simulations of secondary horseshoe vortices which form at the base of end-mounted cylinder on a flat plate. This problem has a great deal of aesthetic appeal (see in particular [63,64]), as well as numerous applications in submarine and aircraft design. The presence of horseshoe vortices is quite evident in wind-sculpted snow drifts which form at the base of trees and telephone poles. The horseshoe vortex results from the pressure differential developed when a boundary layer flow impinges upon an end-mounted cylinder. The flow in the free stream imposes a greater dynamic pressure upon the cylinder than the low speed fluid at the cylinder base. The low speed flow is unable to overcome the adverse pressure gradient and flow reversal results, thereby producing a steady persisting vortex in front of the cylinder. Conservation of angular momentum (Kelvin's theorem) causes the vortex to wrap around the cylinder, giving it the shape from which it derives its name. An extensive exposition on the laminar horseshoe vortex is given in [65].

The first example is three-dimensional flow past a cylinder between two



(a)



(b)

Figure 8.10: Three-dimensional flow past a cylinder mounted on a flat plate at a Reynolds number $R = 1000$ based on maximum inflow velocity, U, and cylinder diameter, D. (a) Mesh, inflow velocity profile, and three-dimensional horseshoe vortex structures shown at an early time $Ut/D = 4$. (b) Detailed flow structure close to the cylinder reveals the presence of two horseshoe vortices at time $Ut/D = 9$.

plates. In the flow direction the velocity profile $u = U[1 - 16(1/2 - z/H)^4]$, is imposed at the inflow plane, while outflow boundary conditions are imposed at the exit plane. Periodic boundary conditions are imposed in the y-direction, no-slip wall boundary conditions are imposed on the plate ($z = 0$), and symmetry boundary conditions are assumed at the half plane ($z = H/2$). The Reynolds number based on the symmetry plane inflow velocity (U) and cylinder diameter, D, is $R = 1000$; the geometric ratio D/H is unity for this calculation. Figure 8.10a shows the three-dimensional horseshoe vortex structure that forms around the cylinder at a non-dimensional time $Ut/D = 9$, while Figure 8.10b shows the more detailed flow structure close to the cylinder. The discretization parameter is $h = (N = 8, K = 40)$ and the execution time is roughly 40 hours on a four node iPSC/2-VX hypercube.

The second simulation is flow around a low aspect-ratio cylinder having a height $H = 1/2$ and diameter $D = 1$. In this case, the computational domain is extended to $z = 1.75$ and fluid is allowed to pass over the top of the cylinder. Baker [65] presents extensive flow visualization results for this configuration with $Re = UD/\nu = 4370$ and $D/\delta^* = 21.3$, where δ^* is the boundary layer displacement thickness at the position of the cylinder when the cylinder is not present. For the present calculations, $Re = 3000$ and $D/\delta^* = 21$ at the inlet plane. The boundary conditions are the same as in the previous example with the exception that the inlet profile is now $u = U[1 - (1 - z/D)^{21}]$. Numerical visualization results are presented in Figures 8.11 The chaotic flow seen above the cylinder indicates that additional refinement is required to fully resolve the flow in that region. Qualitative agreement with Sutton's experiment [65] is seen in the lower part of Figure 8.11. The vortex pattern corresponds to the six vortex system described by Baker, as his experimental results predict for these

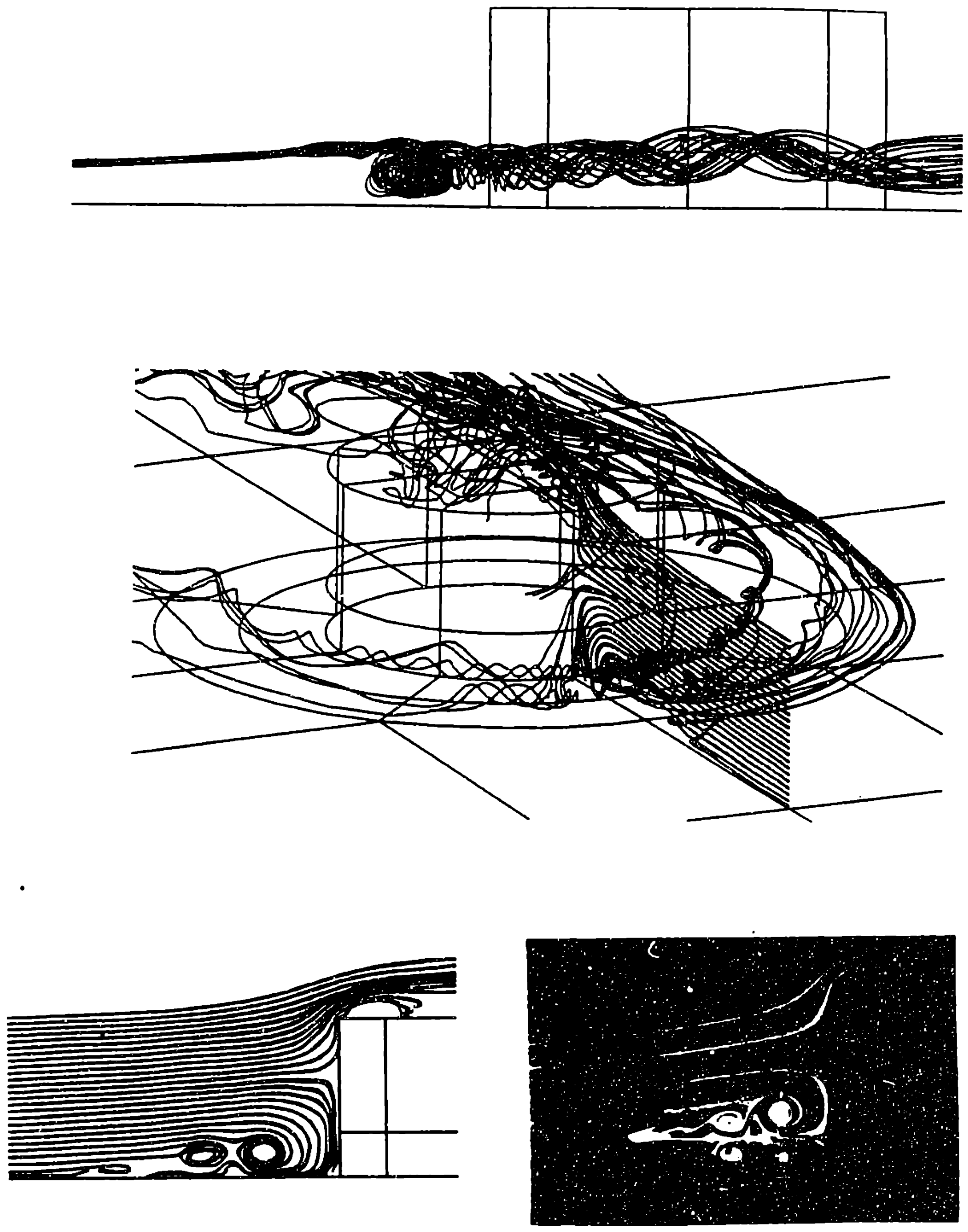


Figure 8.11: Similar to Figure 8.10 except that now the flow at $Re = 3000$ is allowed to pass over the cylinder (center). The displacement boundary layer thickness of the impinging flow is $\delta^* = D/21$. Tight vortex core is seen at the cylinder base in the top figure. Multiple vortex formation is evident in the lower figures. Six vortex pattern at this Reynolds number is similar to the results of Sutton [65].

run conditions, although the weakest S_0 vortex is not observed in the present case.

We turn now to the investigation of a more applied problem. We consider internal flow in a circular pipe in which a square prism is placed transverse to the axis of the pipe, as shown in Figures 8.12a and b (a portion of the pipe wall has been removed to permit observation of the relative orientation of the prism). The pipe diameter is taken to be $D = 1$ and the prism "diameter" $d = 1/4$. The inlet velocity profile is parabolic with maximum velocity on the axis of $U = 1$. The application calls for measurement of volume flux through the pipe by establishing a relationship between the frequency of vortex shedding from the prism and the total flow rate. The problem is strongly influenced by three-dimensional effects because the boundary layer thickness of the impinging flow is of the same order as the length of the prism. Consequently, the vortex shedding is not similar to that computed for two-dimensional problems.

The discretization consists of 96 elements for which $N = 8$. The complexity of the resultant flow pattern in the steady laminar regime is seen in Figures 8.13a through c, which show the three-dimensional streamline pattern. Such results are typical for low speed shear flow behind cylinders. The axial pressure gradient along the prism in the recirculation zone just aft of the prism causes transport of low speed fluid which "feeds" the vortex at the symmetry plane. The net result is that fluid particles which initally form a sheet transverse to the pipe and parallel to the prism end up forming a central core at the pipe exit. The resulting vorticity pattern is evident in the velocity vector plot shown in Figure 8.13d, which is taken at a cross section two pipe diameters downstream of the prism. Based upon initial trial runs, no vortex shedding is evidenced for

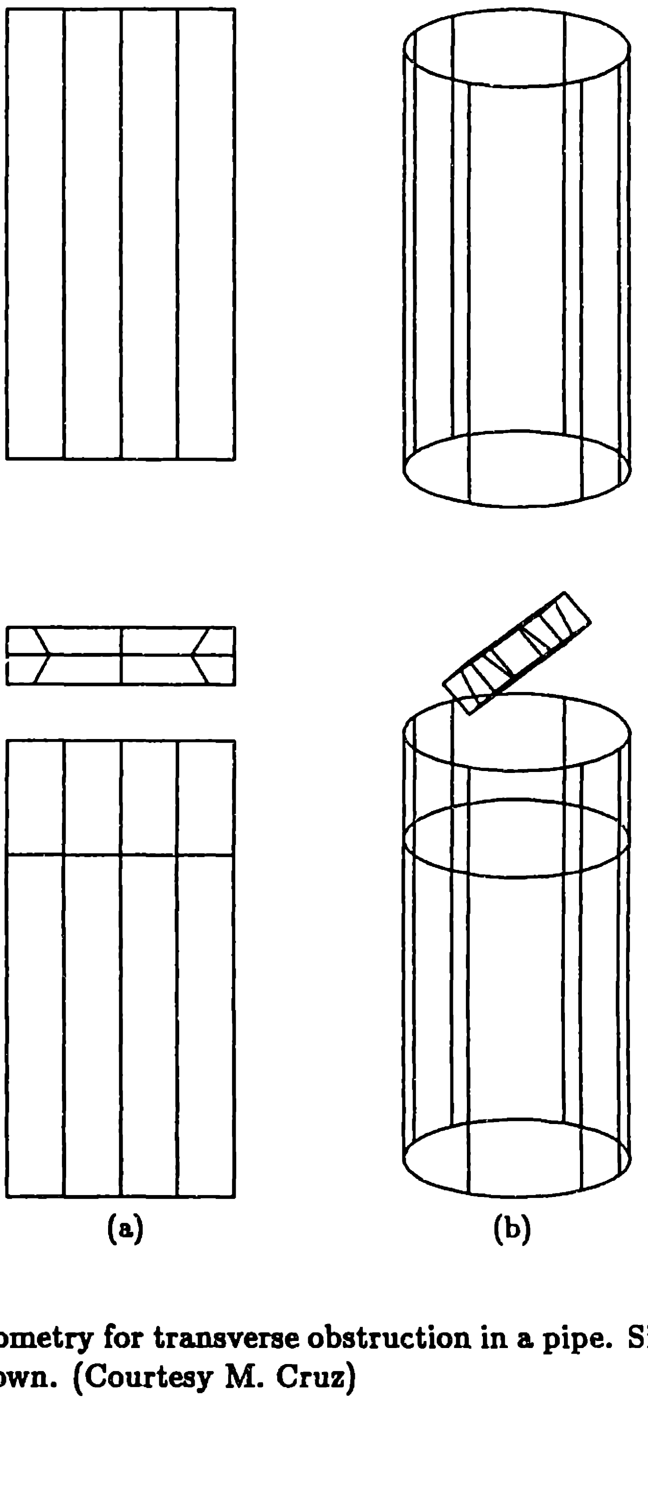


Figure 8.12: Geometry for transverse obstruction in a pipe. Side walls near the prism are not shown. (Courtesy M. Cruz)

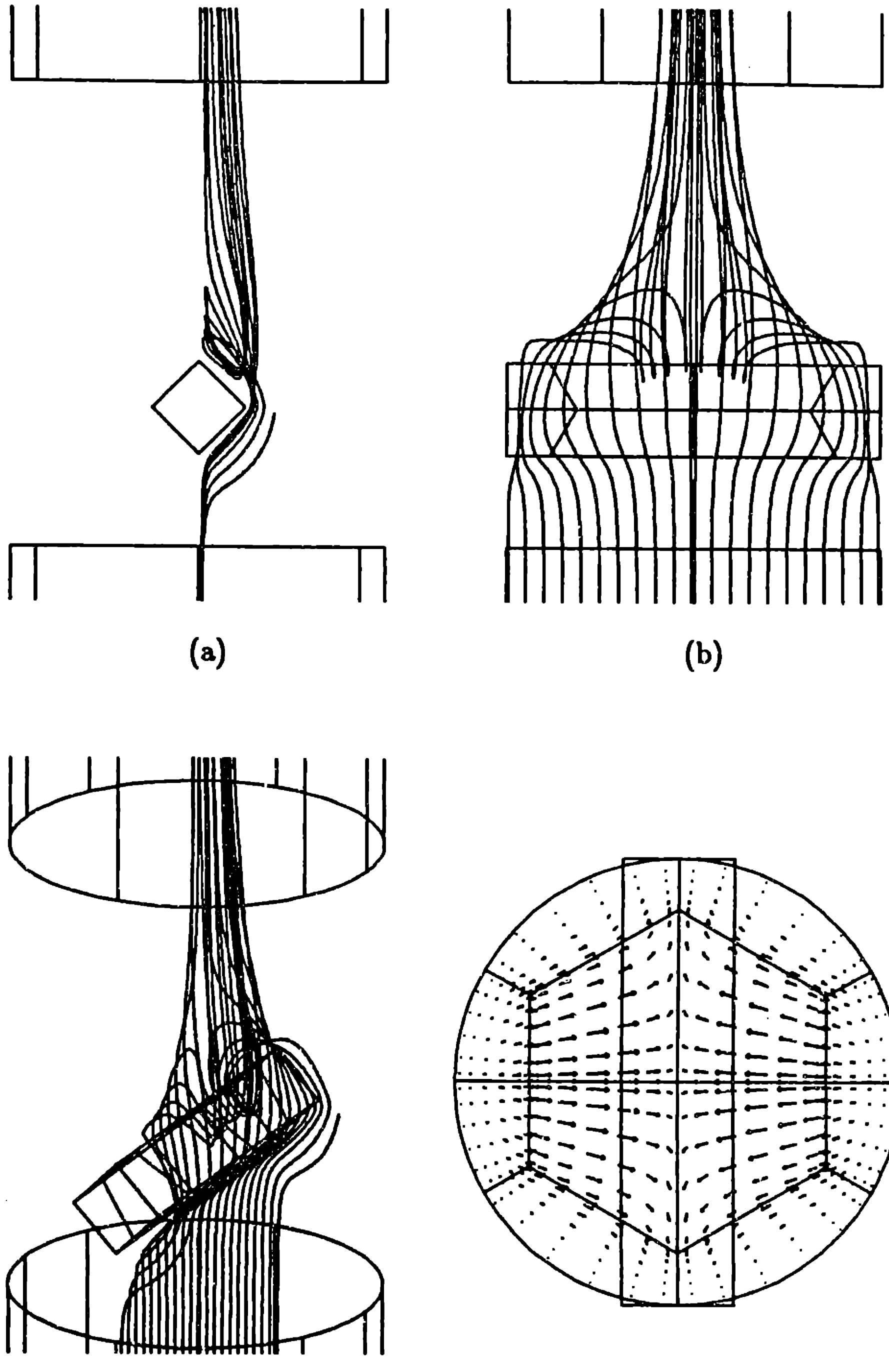
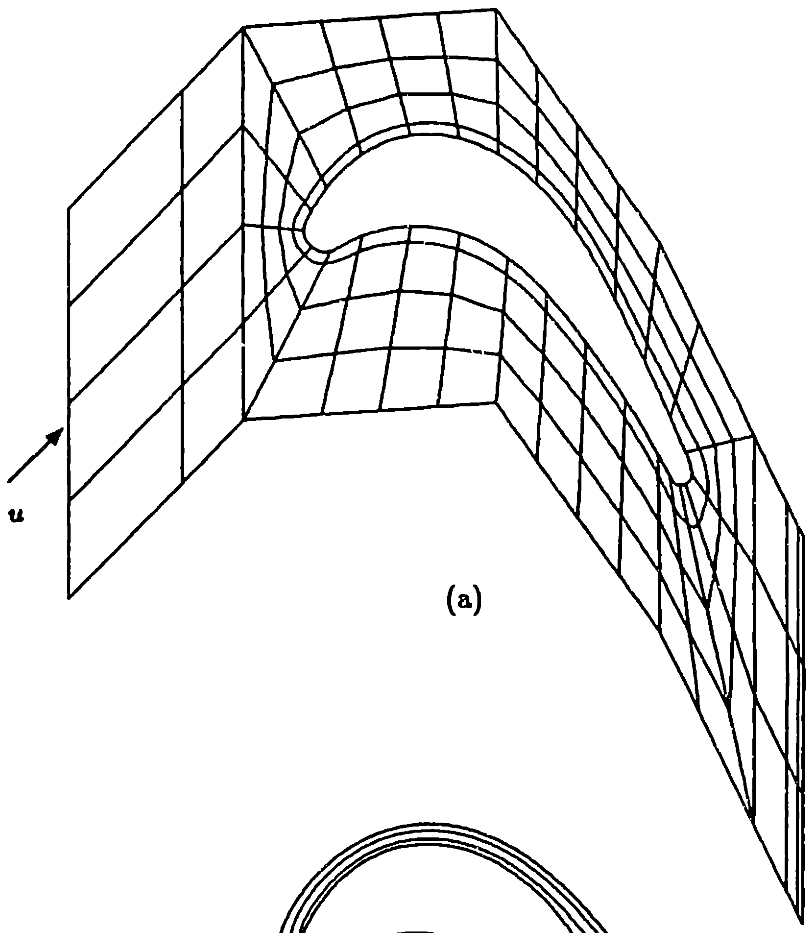


Figure 8.13: Streamline patterns and velocity vectors reveal three-dimensional nature of this fully developed pipe flow passing a transverse prism. The velocity vectors (d) two diameters downstream of the prism indicate the presence of weak counter rotating vortices similar to the horseshoe vortices of the previous examples.

Reynolds numbers of $Re = Ud/\nu \leq 200$. Higher resolution simulations will be requried to establish vortex shedding.

Our final simulation is in fact a two-dimensional subproblem for a three-dimensional horseshoe vortex calculation currently being investigated by Renaud [66]. It is presented here to illustrate the geometric flexibility of the isoparametric spectral element formulation. The problem is that of flow past a unity aspect ratio turbine blade at a Reynolds number based on chord length, L_c, of $Re = 1000$. Of interest is to study the behavior of the horseshoe vortices under loaded conditions. Two dimensional solutions to the turbine blade flow are used as initial conditions for the full three-dimensional simulation. The two-dimensional discretization consists of $K = 125$ elements, $N = 8$. The three-dimensional discretization will have four levels of elements and require the maximum capacity of the thirty-two node iPSC/2-VX hypercube memory. The geometry is seen in Figure 8.14a. Periodic boundary conditions are used above and below the blade, the inlet velocity is specified to be $U = 1$, and Neumann conditions are imposed at the domain exit. Flow separation is evident in Figure 8.14b at a convective time of $tU/L_c = 6$.



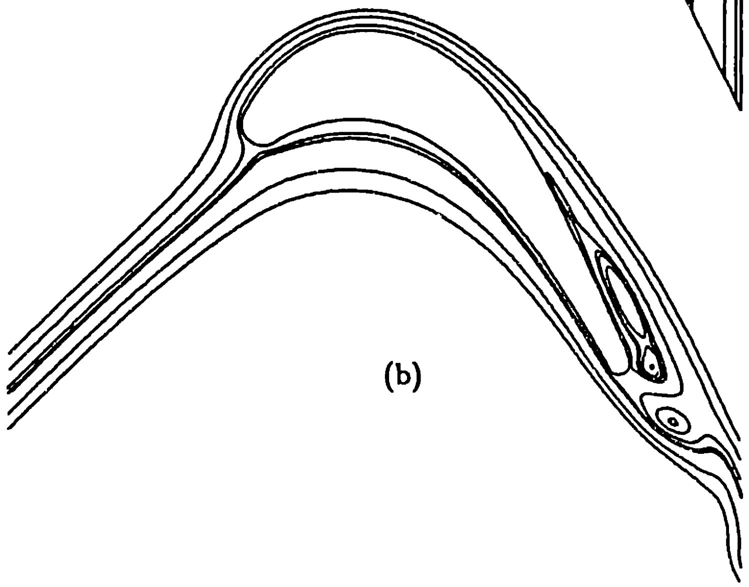


Figure 8.14: Mesh ($K = 125, N = 8$) for turbine blade analysis at $Re = 1000$ based on chord length L_c (a). Flow separation is evident at a time $tu/L_c = 6$ (b). (Mesh courtesy of E. Renaud).

Appendix A
Speed and Cost Data for Several Modern Computer Systems

	Price (K$)	Rated peak MFLOPS	e (MFLOPS/$)
iPSC/1-d4	91	0.3	0.33×10^{-5}
iPSC/1-VX/d4	286*	160	55.9×10^{-5}
iPSC/2(4M)/d4	203*	1.8	0.87×10^{-5}
iPSC/2-VX/d4	363*	160	44.1×10^{-5}
CRAY X/MP-12	5000	190	3.80×10^{-5}
CRAY-2/4-256	15500	1000	6.45×10^{-5}
μVAX-II	10	0.1	1.0×10^{-5}
FPS-164	500	5.0	1.0×10^{-5}

*Indicates quoted manufacturer's price.

Appendix B
Timing Results for 80,000 Degree-of-Freedom Stokes Problem

	T_{solve} (seconds)	Parallel Efficiency (%)	MFLOPS$'$	e' (MFLOPS$'$/$)
iPSC/1-d4	19100	99	0.3	0.33×10^{-5}
iPSC/1-VX/d4	360	25	16	5.6×10^{-5}
iPSC/2(4M)-d4	5760	99	1.0	0.47×10^{-5}
iPSC/2-VX/d4	130	75	44	12.1×10^{-5}
CRAY X/MP-48 (1)	87	-	66	1.32×10^{-5}
CRAY-2/4-256 (1)	104*	-	55	1.42×10^{-5}
CRAY-2/4-256 (4)	32*	80	176	1.14×10^{-5}
μVAX-II	57200	-	0.1	1.0×10^{-5}

* Reported timing adjusted for iteration count.

Appendix C
Isoparametric Discretizations in $\mathbf{R}^3$

In this section we describe a scheme for efficient evaluation of the discrete Laplacian for general deformed elements in $\mathbf{R}^3$. A lucid exposition of general isoparametric mappings can be found in [67]. As the discretization is based upon an integrated weighted residual or variational form, we only need to consider the derivation for a single element; the integral over the entire domain is the sum of the integrals on each element. We assume at the outset that $\vec{x} = (x, y, z) = \vec{x}(r, s, t)$ is a known function of the local coordinates (r, s, t). For a single element, the variational form requires evaluation of the following integral:

$$I = \int_{\Omega^k} \nabla\phi \cdot \nabla u \, d\Omega \quad , \tag{C.1}$$

where ϕ is an arbitrary test function. When using iterative solvers, u will be a known function, e.g. the ith iterate, so we evaluate (C.1) accordingly. The extension to the case where u is unknown is straightforward. The expression (C.1) in local coordinates is:

$$I = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \nabla\phi \cdot \nabla u \, J \, dr \, ds \, dt \quad , \tag{C.2}$$

where,

$$\nabla u \equiv \left[\frac{\partial r}{\partial x}\frac{\partial u}{\partial r} + \frac{\partial s}{\partial x}\frac{\partial u}{\partial s} + \frac{\partial t}{\partial x}\frac{\partial u}{\partial t}\right]\hat{\imath} + \left[\frac{\partial r}{\partial y}\frac{\partial u}{\partial r} + \frac{\partial s}{\partial y}\frac{\partial u}{\partial s} + \frac{\partial t}{\partial y}\frac{\partial u}{\partial t}\right]\hat{\jmath} + \left[\frac{\partial r}{\partial z}\frac{\partial u}{\partial r} + \frac{\partial s}{\partial z}\frac{\partial u}{\partial s} + \frac{\partial t}{\partial z}\frac{\partial u}{\partial t}\right]\hat{k} \quad , \tag{C.3}$$

and

$$J \equiv \begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{vmatrix} \quad . \tag{C.4}$$

J is the local Jacobian.

We first note that the metrics, $\frac{\partial r}{\partial x}$, etc., are of the form $r(x, y, z)$, rather than $x(r, s, t)$. Using the following identity:

$$\begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial z} \\ \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \\ \frac{\partial t}{\partial x} & \frac{\partial t}{\partial y} & \frac{\partial t}{\partial z} \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{C.5}$$

we can readily express the required metrics as explicit functions of (r, s, t). Recall that derivatives with respect to r, s, and t can be efficiently evaluated using matrix-matrix product subroutines. Note that inversion of the above system will result in an inverse Jacobian in front of each metric in (C.3).

Taking the inner product $\nabla\phi \cdot \nabla u$ with the expansion given by (C.3), we arrive at a compact expression for the integrand in (C.2):

$$I = \begin{bmatrix} \frac{\partial \phi}{\partial r} \\ \frac{\partial \phi}{\partial s} \\ \frac{\partial \phi}{\partial t} \end{bmatrix}^T \begin{bmatrix} \tilde{\mathcal{G}}_{11} & \tilde{\mathcal{G}}_{12} & \tilde{\mathcal{G}}_{13} \\ \tilde{\mathcal{G}}_{21} & \tilde{\mathcal{G}}_{22} & \tilde{\mathcal{G}}_{23} \\ \tilde{\mathcal{G}}_{31} & \tilde{\mathcal{G}}_{32} & \tilde{\mathcal{G}}_{33} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial s} \\ \frac{\partial u}{\partial t} \end{bmatrix} , \qquad (C.6)$$

where

$$\tilde{\mathcal{G}}_{ij} \equiv J \sum_{k=1}^{3} \left(\frac{\partial r_i}{\partial x_k} \frac{\partial r_j}{\partial x_k} \right) \quad . \qquad (C.7)$$

Here r_i and x_k are taken to be $(r_1, r_2, r_3) = (r, s, t)$ and $(x_1, x_2, x_3) = (x, y, z)$, respectively. The geometric factors $\tilde{\mathcal{G}}_{ij}$ are symmetric, i.e. $\tilde{\mathcal{G}}_{ij} = \tilde{\mathcal{G}}_{ji}$, and, for the case of affine transformations, form a diagonal matrix, i.e., $\tilde{\mathcal{G}}_{ij} = c_i \delta_{ij}$, where c_i is a constant determined by the $x-$, $y-$, or $z-$ stretching of the element.

We now proceed to express the integral (C.1) in terms of our discrete functional representations. The integrand I is just a scalar function of (r, s, t), and as such can be integrated according to the Gauss-Lobatto quadrature rule from Chapter 3:

$$\int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} I \, dr \, ds \, dt = \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=0}^{N} I_{ijk} \, \rho_{ijk} \ , \qquad (C.8)$$

where ρ_{ijk} is the quadrature weight associated with the nodal point $\mathbf{r}_{ijk}$. Since collocation is employed to evaluate the integrals of products of functions, the following generic forms are equivalent:

$$\begin{aligned} \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} f \, g \, dr \, ds \, dt &= \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=0}^{N} f_{ijk} \, g_{ijk} \, \rho_{ijk} \qquad (C.9) \\ &= \mathbf{f} \cdot (\mathbf{g} \circ \rho) \\ &= \mathbf{g} \cdot (\mathbf{f} \circ \rho) \\ &= \rho \cdot (\mathbf{f} \circ \mathbf{g}) \ , \end{aligned}$$

where $\mathbf{f} = f_{ijk}$ is taken to be discrete representation of f on a given element, and $\mathbf{f} \circ \mathbf{g}$ implies pointwise collocation of the vectors $\mathbf{f}$ and $\mathbf{g}$. In other words, we can attach the quadrature weight ρ_{ijk} as a multiplier of any of the functions in our original integrand in (C.2). In particular, we can take it to be a multiplier of the Jacobian, J, and factor it directly into the geometric factors:

$$\mathcal{G}_{ij} \equiv \rho \circ \tilde{\mathcal{G}}_{ij} \quad . \tag{C.10}$$

In so doing, we imply that condensation will now occur somewhere else in the evaluation of $\int \mathcal{I}$. This in fact is exactly the what is required and we are now in a position to express (C.1) in its discrete equivalent as $\phi^T \mathbf{A}\mathbf{u}$.

Expanding the transposed vector in (C.6) and inserting the derivative operators D_{r_i} for $\frac{\partial u}{\partial r_i}$, yields the final expression for discrete evaluation of the desired integral:

$$\phi^T \mathbf{A}\mathbf{u} = \phi^T \cdot \begin{bmatrix} D_r^T & D_s^T & D_t^T \end{bmatrix} \begin{bmatrix} \mathcal{G}_{11} & \mathcal{G}_{12} & \mathcal{G}_{13} \\ \mathcal{G}_{21} & \mathcal{G}_{22} & \mathcal{G}_{23} \\ \mathcal{G}_{31} & \mathcal{G}_{32} & \mathcal{G}_{33} \end{bmatrix} \begin{bmatrix} D_r \\ D_s \\ D_t \end{bmatrix} \mathbf{u} \tag{C.11}$$

The $\mathcal{G}_{ij}$'s are computed and stored once at the beginning of the calculation (this is not the case for moving boundary problems [36]).

When evaluating matrix-vector products of the form $\mathbf{A}\mathbf{u}$, there is of course no condensation on ϕ. $\mathbf{A}\mathbf{u}$ products are therefore evaluated as expressed in (C.11), working from right to left. We begin with three matrix-matrix products, $(D_{ip} u_{pjk})$, $(D_{jp} u_{ipk})$, $(D_{kp} u_{ijp})$, (op. cnt. - $3 \times 2N^4$); followed by nine collocation and six summation operations for the geometric factors (op. cnt. - $15N^3$); three additional matrix-matrix products for the $D_{r_i}^T$'s (op. cnt. - $3 \times 2N^4$); and finally, two summation operations (op. cnt. - $2N^3$) to yield the residual vector, **Au**.

Bibliography

[1] E.M. Rønquist, *Optimal Spectral Element Methods for the Unsteady Three-dimensional Incompressible Navier-Stokes Equations*, Ph.D. Thesis, Massachusetts Institute of Technology, 1988.

[2] T.F. Chan, Y. Saad, and M.H. Schultz, in *Hypercube Multiprocessors 1986* (M.T. Heath, ed), SIAM, Philadelphia, 1986.

[3] O.A. McBryan, and E.F. van de Velde, in *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing* (C.W. Gear and R.G. Voigt, eds.), SIAM, Philadelphia, 1987.

[4] T.F. Chan, and D.C.Resasco, in *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing* (C.W. Gear and R.G. Voigt, eds.), SIAM, Philadelphia, 1987.

[5] R. Glowinski and M.F. Wheeler in *Proceedings of the First International Conference on Domain Decomposition Methods for Partial Differential Equations, Paris* (R. Glowinski, G. Golub, G. Meurant, and J. Periaux, eds.), pp. 144-172, SIAM, Philadelphia, 1987.

[6] O.B. Widlund, in *Proceedings of the First International Conference on Domain Decomposition Methods for Partial Differential Equations, Paris* (R. Glowinski, G. Golub, G. Meurant, and J. Periaux, eds.),pp. 113-127, SIAM, Philadelphia, 1987.

[7] D.E. Keyes and W.D. Gropp, in *Proceedings of the Second International Conference on Domain Decomposition Methods for Partial Differential Equations, Los Angeles* SIAM, Philadelphia, 1988

[8] L.Adams and R.G. Voigt, in *Large Scale Scientific Computation* (S. Parter, ed.), pp 301-321, Academic Press, Orlando, Florida, 1984.

[9] W.D. Gropp and D.E. Keyes, SIAM J. of Sci. and Stat. Comput. 9 (1988) 312-326.

[10] Y. Saad and M.H. Schultz, *Topological Properties of Hypercubes*, Research Report YALEU/DCS/RR-389, Yale University, New Haven, 1985.

[11] H.X.Lin, and H.J. Sips, in *Proc. 1986 Int. Conf. on Parallel Processing*, 503-510, 1986.

[12] A.T. Patera, A spectral element method for fluid dynamics; Laminar flow in a channel expansion, *J. Comput. Phys.*, **54**, 1984, p.468.

[13] D. Gottlieb and S. Orszag, *Numerical Analysis of Spectral Methods* SIAM-CBMS, Philadelphia, 1977.

[14] P. Moin and J. Kim, On the Numerical Solution of Time-Dependent Viscous Inompressible Fluid Flows Involving Solid Boundaries, *J. Comput. Phys.*, **35**,1980, p.381.

[15] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, 1987.

[16] S.A. Orszag, Spectral Methods for Problems in Complex Geometries, *J. Comput. Phys.*, **37**, 1980, p. 70.

[17] Y. Maday and A.T. Patera, Spectral element methods for the Navier-Stokes equations, in *State of the Art Surveys in Computational Mechanics* (Edited by A.K. Noor), ASME, New York, to appear.

[18] A.H. Stroud, and D. Secrest, *Gaussian Quadrature Formulas*, Prentice Hall, 1966.

[19] Brezzi, F., On the existence, uniqueness and approximation of saddle-point problems arising from Lagrange multipliers, *Rairo Anal. Numer.*, **8**, R2, 1974, p. 129.

[20] Girault, V. and Raviart, P.A., *Finite Element Approximation of the Navier-Stokes Equations*, Springer, 1986.

[21] Maday, Y., Patera. A.T., and Rønquist, E.M., A well-posed optimal spectral element approximation for the Stokes problem, *SIAM J. Numer. Anal.*, to appear.

[22] Maday, Y., Patera. A.T., and Rønquist, E.M., Optimal Legendre spectral element methods for the multi-dimensional Stokes problem, in preparation.

[23] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

[24] E.M. Rønquist, *private communication.*

[25] Temam, R., *Navier-Stokes Equations. Theory and Numerical Analysis*, North-Holland, Amsterdam, 1984.

[26] Orszag, S.A., Israeli, M. and Deville, M.O., Boundary conditions for incompressible flows, *J. Sci. Comput.*, **1**, 1986, p. 75.

[27] Korczak, K.Z. and Patera, A.T., An isoparametric spectral element method for solution of the Navier-Stokes equations in complex geometry, *J. Comput. Phys.*, **62**, 1986, p. 361.

[28] L.I.G. Kovasznay, Laminar Flow behind a Two-Dimensional Grid, *Proceedings of the Cambridge Philosophical Society*, p.44, 1948.

[29] Y. Saad and M.H. Schultz, *GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems,* SIAM J. Sci. Stat. Comp., **7**(1986), p. 856.

[30] Golub, G.H. and Van Loan, C.F., *Matrix Computations*, John Hopkins University Press, Baltimore, Maryland, 1983.

[31] E.M. Rønquist and A.T. Patera, in *Proc. of the Seventh GAMM Conf. on Num. Methods in Fluid Mechanics*, Vieweg, to appear, 1988.

[32] Y. Maday, D.I. Meiron, E.M. Rønquist, and A.T. Patera, *Iterative Saddle Problem Decomposition Methods for the Steady and Unsteady Stokes Equations*, 1987.

[33] Y. Maday and R. Munoz, J. of Sci. Comp., (1988), to appear.

[34] E.M. Rønquist and A.T. Patera, J. of Sci. Comp., 4 (1987).

[35] C.A. Mavriplis, *Nonconforming Discretizations and a Posteriori Error Estimators for Adaptive Spectral Element Techniques*, Ph.D. Thesis, Massachusetts Institute of Technology, 1989.

[36] L.W. Ho, *A Legendre Spectral Element Method for Simulation of Incompressible Unsteady Viscous Free-Surface Flows*, Ph.D. Thesis, Massachusetts Institute of Technology, 1989.

[37] J.S. Przemieniecki, AIAA J. 1 (1963) 138.

[38] J.L. Gustafson, G.R. Montry, and R.E. Benner, Development of Parallel Methods for a 1024-Processor Hypercube, *SIAM J. on Sci. and Stat. Comput.*, **9**, 4, July 1988, pp. 609-638.

[39] G. Anagnostou, Ph.D. Thesis, Massachusetts Institute of Technology, in progress.

[40] R.E. Bank and C.C. Douglas, *Sharp Estimates for Multigrid Rates of Convergence with General Smoothing and Acceleration*, SINUM 22,4 (1985), 617-633.

[41] A.K. Noor, Ed., *Parallel Computations and Their Impact on Mechanics*, ASME, N.Y., 1987.

[42] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker (1988). *Solving Problems on Concurrent Processors. Volume 1 : General Techniques and Regular Problems.* Prentice-Hall, Englewood Cliffs, New Jersey.

[43] Fox, G.C., and Otto, S.W., *Concurrent Computation and the Theory of Complex Systems* in Hypercube Multiprocessors, (M.T. Heath, ed.), SIAM, Philadelphia, 1986.

[44] B. Nour-Omid, A. Raefsky, and G. Lyzenga, *Solving Finite Element Equations on Concurrent Computers* in *Parallel Computations and Their Impact on Mechanics*,(A.K. Noor, Ed.), ASME, N.Y., 1987.

[45] D. Zeritis, *Optimal Mapping Techniques and the Nested Dissection Algorithm* B.S. Thesis, Massachusetts Institute of Technology, 1988.

[46] Lucas, R.F., *Solving Planar Systems of Equations on Distributed-Memory Multiprocessors*, Stanford University Ph.D. Thesis, Dept. of Electrical Engineering, 1987.

[47] M.T. Heath, The Hypercube: A Tutorial Overview, in *Hypercube Multiprocessors 1986*, (M.T. Heath, ed.), SIAM, Philadelphia, 1986.

[48] G. Anagnostou, P.F. Fischer, D. Dewey, and A.T. Patera (1988). Finite/Spectral Element Navier-Stokes Methods on Vector Hypercubes and Geometry-Defining Processor Reconfigurable Lattices. In *Proc. Taiwan - USA Conf. on Computational Fluid Dynamics*, to appear.

[49] P.F. Fischer, and A.T. Patera (1988). Parallel Spectral Element Solution of the Stokes Problem. *J. Comput. Phys.*, to appear.

[50] W.A. Gross, *Fluid Film Lubrication*, John Wiley & Sons, Inc., New York (1980).

[51] P. Bar-Yoseph, S. Seelig, A. Sloan, and K.G. Roesner, Vortex breakdown in spherical gap, Phys. Fluids, 30(6), 1987, p. 1581.

[52] P.S. Marcus and L.S. Tuckerman, Simulation of Flow Between Concentric Rotating Spheres, Part 2, *J. Fluid Mech.*, **185**, 1987, p. 1.

[53] H.S. Kheshgi and L.E. Scriven, Viscous flow through a rotating square channel, Phys. Fluids, **28**(10), 1985, p. 2968.

[54] M. Van Dyke, *An Album of Fluid Motion*, The Parabolic Press, Stanford, California, 1982.

[55] N.K. Ghaddar, M. Magen, B.B. Mikic, and A.T. Patera, Numerical investigation of inompressible flow in grooved channels. Part 2. Resonance and oscillatory heat-transfer enhancement, *J. Fluid Mech.*, **168**, 1986, p. 541.

[56] C.H. Amon and A.T. Patera, Numerical calculation of stable three-dimensional tertiary states in grooved-channel flow, Phys. Fluids, submitted 1989.

[57] P.R. Bandyopadhyay, Resonant flow in small cavities submerged in a boundary layer, *Proc. R. Soc. Lond. A* **420**, 1988, p. 219.

[58] A. Zukauskas and J. Ziugzda, *Heat Transfer of a Cylinder in Crossflow*, Hemisphere, Washington D.C., (1986).

[59] H. Honji and A. Taneda, Unsteady Flow Past a Circular Cylinder, *J. Phys. Soc. Japan*, **27**, 6, 1969, p. 1668.

[60] G.E. Karniadakis, E.T. Bullister, and A.T. Patera, A Spectral Element Method for Solution of the Two- and Three-Dimensional Time-Dependent Incompressible Navier-Stokes Equations, in: *Finite Element Mehtods for Nonlinear Problems, Europe-US Symposium, Trondheim, Norway 1985*, Bergan, Bathe, Wunderlich, eds., Springer, Berlin Heidelberg, 1986.

[61] T. Kawamura and K. Kuwahara, Computation of High Reynolds Number Flow around a Circular Cylinder with Surface Roughness, AIAA-84-0430, 1984.

[62] P.R. Spalart, Vortex Methods for Separated Flows, NASA Tech. Memo. 100068., June 1988.

[63] E.S. Taylor and A.H. Shapiro, *Secondary Flow*, produced by the Nat. Comm. for Fluid Mech. Films and Educational Services Inc.

[64] Nat. Comm. for Fluid Mech. Films, *Illustrated Experiments in Fluid Mechanics*, MIT Press, Cambridge Mass. and London, England, 1972.

[65] C.J. Baker, The laminar horseshoe vortex, *J. Fluid Mech.*, 95, 1979, p. 347.

[66] E. Renaud, Ph.D. Thesis, Massachusetts Institute of Technology, in progress.

[67] R. Courant, *Differential and Integral Calculus, Vol.* II, Wiley-Interscience, 1968.