The Impact Of Computer Architectures On Linear Algebra and Numerical Libraries

Jack Dongarra
Innovative Computing Laboratory
University of Tennessee
High Performance Computers

♦ ~ 20 years ago
  - $1 \times 10^6$ Floating Point Ops/sec (Mflop/s)
    - Scalar based

♦ ~ 10 years ago
  - $1 \times 10^9$ Floating Point Ops/sec (Gflop/s)
    - Vector & Shared memory computing, bandwidth aware
    - Block partitioned, latency tolerant

♦ ~ Today
  - $1 \times 10^{12}$ Floating Point Ops/sec (Tflop/s)
    - Highly parallel, distributed processing, message passing, network based
    - data decomposition, communication/computation

♦ ~ 10 years away
  - $1 \times 10^{15}$ Floating Point Ops/sec (Pflop/s)
    - Many more levels MH, combination/grids & HPC
    - More adaptive, LT and bandwidth aware, fault tolerant, extended precision, attention to SMP nodes
TOP500

- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP
  \[ Ax = b, \text{ dense problem} \]

- Updated twice a year
  SC‘xy in the States in November
  Meeting in Mannheim, Germany in June

- All data available from www.top500.org
In 1980 a computation that took 1 full year to complete can now be done in ~ 9 hours!
In 1980 a computation that took 1 full year to complete can now be done in ~ 13 minutes!
Fastest Computer Over Time

In 1980 a computation that took 1 full year to complete can today be done in ~90 seconds!
# Top 10 Machines (Nov 2000)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Company</th>
<th>Machine</th>
<th>Procs</th>
<th>Gflop/s</th>
<th>Place</th>
<th>Country</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM</td>
<td>ASCI White</td>
<td>8192</td>
<td>4938</td>
<td>Livermore National Laboratory</td>
<td>Livermore</td>
<td>2000</td>
</tr>
<tr>
<td>2</td>
<td>Intel</td>
<td>ASCI Red</td>
<td>9632</td>
<td>2380</td>
<td>Sandia National Labs</td>
<td>USA</td>
<td>1999</td>
</tr>
<tr>
<td>3</td>
<td>IBM</td>
<td>ASCI Blue-Pacific SST, IBM SP 604e</td>
<td>5808</td>
<td>2144</td>
<td>Lawrence Livermore National Laboratory</td>
<td>USA</td>
<td>1999</td>
</tr>
<tr>
<td>4</td>
<td>SGI</td>
<td>ASCI Blue Mountain</td>
<td>6144</td>
<td>1608</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>1998</td>
</tr>
<tr>
<td>5</td>
<td>IBM</td>
<td>SP Power3 375 MHz</td>
<td>1336</td>
<td>1417</td>
<td>Naval Oceanographic Office (NAVOCEANO)</td>
<td>USA</td>
<td>2000</td>
</tr>
<tr>
<td>6</td>
<td>IBM</td>
<td>SP Power3 375 MHz, 8 way</td>
<td>1104</td>
<td>1179</td>
<td>National Center for Environmental Protection</td>
<td>USA</td>
<td>2000</td>
</tr>
<tr>
<td>7</td>
<td>Hitachi</td>
<td>SR8000-F1/112</td>
<td>112</td>
<td>1035</td>
<td>Leibniz Rechenzentrum Muenchen</td>
<td>Germany</td>
<td>2000</td>
</tr>
<tr>
<td>8</td>
<td>IBM</td>
<td>SP Power3 375 MHz, 8 way</td>
<td>1152</td>
<td>929</td>
<td>UCSD/San Diego Supercomputer Center</td>
<td>USA</td>
<td>2000</td>
</tr>
<tr>
<td>9</td>
<td>Hitachi</td>
<td>SR8000-F1/100</td>
<td>100</td>
<td>917</td>
<td>High Energy Accelerator Research Organization /KEK</td>
<td>Japan</td>
<td>2000</td>
</tr>
<tr>
<td>10</td>
<td>Cray Inc.</td>
<td>T3E1200</td>
<td>1084</td>
<td>892</td>
<td>Government</td>
<td>USA</td>
<td>1998</td>
</tr>
</tbody>
</table>
Performance Development

[60G - 400 M][4.9 Tflop/s 55Gflop/s], Schwab #15, 1/2 each year, 209 > 100 Gf, faster than Moore’s law,
Performance Development

Entry 1 T 2005 and 1 P 2010
High-Performance Computing Directions: Beowulf-class PC Clusters

**Definition:**
- **COTS PC Nodes**
  - Pentium, Alpha, PowerPC, SMP
- **COTS LAN/SAN Interconnect**
  - Ethernet, Myrinet, Giganet, ATM
- **Open Source Unix**
  - Linux, BSD
- **Message Passing Computing**
  - MPI, PVM
  - HPF

**Advantages:**
- Best price-performance
- Low entry-level cost
- Just-in-place configuration
- Vendor invulnerable
- Scalable
- Rapid technology tracking

*Enabled by* PC hardware, networks and operating system achieving capabilities of scientific workstations at a fraction of the cost and availability of industry standard message passing libraries. However, much more of a contact sport.
Where Does the Performance Go? or Why Should I Care About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)

- Processor-Memory Performance Gap: (grows 50% / year)
- DRAM 9%/yr. (2X/10 yrs)
- "Moore’s Law" μProc 60%/yr. (2X/1.5yr)

Optimizing Computation and Memory Use

♦ Computational optimizations

- Theoretical peak: (# fpus)*(flops/cycle) * Mhz
  - PIII: (1 fpu)*(1 flop/cycle)*(850 Mhz) = 850 MFLOP/s
  - Athlon: (2 fpu)*(1 flop/cycle)*(600 Mhz) = 1200 MFLOP/s
  - Power3: (2 fpu)*(2 flops/cycle)*(375 Mhz) = 1500 MFLOP/s

Operations like:

- $\alpha = x^T y$ : 2 operands (16 Bytes) needed for 2 flops; at 850 Mflop/s will requires 1700 MW/s bandwidth
- $y = \alpha x + y$ : 3 operands (24 Bytes) needed for 2 flops; at 850 Mflop/s will requires 2550 MW/s bandwidth

♦ Memory optimization

- Theoretical peak: (bus width) * (bus speed)
  - PIII: (32 bits)*(133 Mhz) = 532 MB/s = 66.5 MW/s
  - Athlon: (64 bits)*(133 Mhz) = 1064 MB/s = 133 MW/s
  - Power3: (128 bits)*(100 Mhz) = 1600 MB/s = 200 MW/s
Memory Hierarchy

- **By taking advantage of the principle of locality:**
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.

---

**Memory Hierarchy Diagram**

- **Processor**
  - Control
  - Datapath
    - Registers
  - On-Chip Cache

- **Level 2 and 3 Cache (SRAM)**

- **Main Memory (DRAM)**

- **Secondary Storage (Disk)**

- **Tertiary Storage (Disk/Tape)**

**Speed (ns):**
- 1s
- 10s
- 100s

**Size (bytes):**
- 100s
- Ks
- Ms
- Gs
- Ts

- 10,000,000s (10s sec)
- 10,000,000,000s (10s ms)
- 1,000,000,000s (1s ms)
- 10,000,000 s (.1s ms)
- 100,000 s (10s ms)
- 100,000,000 s (10s sec)
Self-Adapting Numerical Software (SANS)

♦ Today’s processors can achieve high-performance, but this requires extensive machine-specific hand tuning.

♦ Operations like the BLAS require many man-hours / platform
  - Software lags far behind hardware introduction
  - Only done if financial incentive is there

♦ Hardware, compilers, and software have a large design space w/many parameters
  - Blocking sizes, loop nesting permutations, loop unrolling depths, software pipelining strategies, register allocations, and instruction schedules.
  - Complicated interactions with the increasingly sophisticated micro-architectures of new microprocessors.

♦ Need for quick/dynamic deployment of optimized routines.

♦ ATLAS - Automatic Tuned Linear Algebra Software
Software Generation Strategy

- Level 1 cache multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization

- Takes about 30 minutes to run.

- "New" model of high performance programming where critical code is machine generated using parameter optimization.

- Code is iteratively generated & timed until optimal case is found. We try:
  - Differing NBs
  - Breaking false dependencies
  - M, N and K loop unrolling

- Designed for RISC arch
  - Super Scalar
  - Need reasonable C compiler

- Today ATLAS in use by Matlab, Mathematica, Octave, Maple, Debian, Scyld Beowulf, SuSE, …
ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.
Intel PIII 933 MHz
MKL 5.0 vs ATLAS 3.2.0 using Windows 2000

- **ATLAS** is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.
Matrix Vector Multiply DGEMV

![Graph showing MFLOPS for different architectures with various vendor and library configurations.]

- Vendor NoTrans
- Vendor Trans
- ATLAS NoTrans
- Atlas Trans
- F77 NoTrans
- F77 NoTrans
ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.
Related Tuning Projects

♦ PHiPAC
  ➢ Portable High Performance ANSI C
    www.icsi.berkeley.edu/~bilmes/phipac initial automatic GEMM generation project

♦ FFTW Fastest Fourier Transform in the West
  ➢ www.fftw.org

♦ UHFFT
  ➢ tuning parallel FFT algorithms
  ➢ rodin.cs.uh.edu/~mirkovic/fft/parfft.htm

♦ SPIRAL
  ➢ Signal Processing Algorithms Implementation Research for Adaptable Libraries maps DSP algorithms to architectures

♦ Sparsity
  ➢ Sparse-matrix-vector and Sparse-matrix-matrix multiplication
    www.cs.berkeley.edu/~ejim/publication/ tunes code to sparsity structure of matrix more later in this tutorial
ATLAS Matrix Multiply
(64 & 32 bit floating point results)

32 bit floating point using SSE

- Intel P4 1.5 GHz SSE
- Intel P4 1.5 GHz
- AMD Athlon 1GHz
- Intel IA64 666MHz
Machine-Assisted Application Development and Adaptation

- **Communication libraries**
  - Optimize for the specifics of one's configuration.

- **Algorithm layout and implementation**
  - Look at the different ways to express implementation
Work in Progress:
ATLAS-like Approach Applied to Broadcast
(PII 8 Way Cluster with 100 Mb/s switched network)

<table>
<thead>
<tr>
<th>Message Size (bytes)</th>
<th>Optimal algorithm</th>
<th>Buffer Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>binomial</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>binomial</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>64</td>
<td>binomial</td>
<td>64</td>
</tr>
<tr>
<td>128</td>
<td>binomial</td>
<td>128</td>
</tr>
<tr>
<td>256</td>
<td>binomial</td>
<td>256</td>
</tr>
<tr>
<td>512</td>
<td>binomial</td>
<td>512</td>
</tr>
<tr>
<td>1K</td>
<td>sequential</td>
<td>1K</td>
</tr>
<tr>
<td>2K</td>
<td>binary</td>
<td>2K</td>
</tr>
<tr>
<td>4K</td>
<td>binary</td>
<td>2K</td>
</tr>
<tr>
<td>8K</td>
<td>binary</td>
<td>2K</td>
</tr>
<tr>
<td>16K</td>
<td>binary</td>
<td>4K</td>
</tr>
<tr>
<td>32K</td>
<td>binary</td>
<td>4K</td>
</tr>
<tr>
<td>64K</td>
<td>ring</td>
<td>4K</td>
</tr>
<tr>
<td>128K</td>
<td>ring</td>
<td>4K</td>
</tr>
<tr>
<td>256K</td>
<td>ring</td>
<td>4K</td>
</tr>
<tr>
<td>512K</td>
<td>ring</td>
<td>4K</td>
</tr>
<tr>
<td>1M</td>
<td>binary</td>
<td>4K</td>
</tr>
</tbody>
</table>
Reformulating/Rearranging/Reuse

♦ Example is the reduction to narrow band from for the SVD

\[
A_{new} = A - u^T - wv^T
\]

\[
y_{new} = A^T u
\]

\[
w_{new} = A_{new} v
\]

♦ Fetch each entry of A once
♦ Restructure and combined operations
♦ Results in a speedup of > 30%
CG Variants by Dynamic Selection at Run Time

- Variants combine inner products to reduce communication bottleneck at the expense of more scalar ops.
- Same number of iterations, no advantage on a sequential processor.
- With a large number of processor and a high-latency network, may be advantages.
- Improvements can range from 15% to 50% depending on size.

<table>
<thead>
<tr>
<th>Classical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Norm calculation:</td>
</tr>
<tr>
<td>[ error = \sqrt{r^T r} ]</td>
</tr>
<tr>
<td>Preconditioner application:</td>
</tr>
<tr>
<td>[ z \leftarrow M^{-1} r ]</td>
</tr>
<tr>
<td>Matrix-vector product:</td>
</tr>
<tr>
<td>Inner products 1:</td>
</tr>
<tr>
<td>[ \rho \leftarrow z^T r ]</td>
</tr>
<tr>
<td>[ \beta \leftarrow \rho / \rho_{old} ]</td>
</tr>
<tr>
<td>Search direction update:</td>
</tr>
<tr>
<td>[ p \leftarrow z + \beta p ]</td>
</tr>
<tr>
<td>Matrix-vector product:</td>
</tr>
<tr>
<td>[ ap \leftarrow A \times p ]</td>
</tr>
<tr>
<td>Preconditioner application:</td>
</tr>
<tr>
<td>Inner products 2:</td>
</tr>
<tr>
<td>[ \pi \leftarrow p^T ap ]</td>
</tr>
<tr>
<td>[ \alpha = \rho / \pi ]</td>
</tr>
<tr>
<td>Residual update:</td>
</tr>
<tr>
<td>[ r \leftarrow r - \alpha Ap ]</td>
</tr>
<tr>
<td>3 separate inner products</td>
</tr>
</tbody>
</table>
Variants combine inner products to reduce communication bottleneck at the expense of more scalar ops.

Same number of iterations, no advantage on a sequential processor.

With a large number of processor and a high-latency network, may be advantages.

Improvements can range from 15% to 50% depending on size.
Gaussian Elimination

Standard Way
subtract a multiple of a row

LINPACK
apply sequence to a column

LAPACK
apply sequence to nb
then apply nb to rest of matrix

\[
a_2 = L^{-1} a_2 \\
a_3 = a_3 - a_1 * a_2
\]
Gaussian Elimination via a Recursive Algorithm

F. Gustavson and S. Toledo

LU Algorithm:
1: Split matrix into two rectangles (m x n/2)
   if only 1 column, scale by reciprocal of pivot & return

2: Apply LU Algorithm to the left part

3: Apply transformations to right part
   (triangular solve \( A_{12} = L^{-1}A_{12} \) and
   matrix multiplication \( A_{22} = A_{22} - A_{21}A_{12} \))

4: Apply LU Algorithm to right part

Most of the work in the matrix multiply
Matrices of size n/2, n/4, n/8, ...
Recursive Factorizations

- Just as accurate as conventional method
- Same number of operations
- Automatic variable blocking
  - Level 1 and 3 BLAS only!
- Extreme clarity and simplicity of expression
- Highly efficient
- The recursive formulation is just a rearrangement of the point-wise LINPACK algorithm
- The standard error analysis applies (assuming the matrix operations are computed the “conventional” way).
Pentium III 550 MHz Dual Processor
LU Factorization

Dual-processor
Recursive LU
LAPACK

Uniprocessor
Recursive LU
LAPACK

Order
M flop/s
500 1000 1500 2000 2500 3000 3500 4000 4500 5000
Pentium III 933 MHz
SGEMM
Windows 2000
using SSE
Pentium III 933 MHz
DGEMM
Windows 2000

M flop/s

Order
DGEMM ATLAS & DGETRF Recursive

AMD Athlon 1GHz (~$1100 system)

Pentium III 933 MHz
S, D, C, and Z GEMM
Windows 2000
S & C use SSE
SuperLU - High Performance Sparse Solvers

- **SuperLU; X. Li and J. Demmel**
  - Solve sparse linear system $A \times x = b$ using Gaussian elimination.
  - Efficient and portable implementation on modern architectures:
    - **Sequential SuperLU**: PC and workstations
      - Achieved up to 40% peak Megaflop rate
    - **SuperLU_MT**: shared-memory parallel machines
      - Achieved up to 10 fold speedup
    - **SuperLU_DIST**: distributed-memory parallel machines
      - Achieved up to 100 fold speedup
  - Support real and complex matrices, fill-reducing orderings, equilibration, numerical pivoting, condition estimation, iterative refinement, and error bounds.

- **Enabled Scientific Discovery**
  - SuperLU solved complex unsymmetric systems of order up to 1.79 million, on the ASCI Blue Pacific Computer at LLNL.
Recursive Factorization Applied to Sparse Direct Methods

Victor Eijkhout, Piotr Luszczek & JD

1. Symbolic Factorization
2. Search for blocks that contain non-zeros
3. Conversion to sparse recursive storage
4. Search for embedded blocks
5. Numerical factorization

Layout of sparse recursive matrix
Dense recursive factorization

The algorithm:

function rlu(A)
begin
  rlu(A_{11}); recursive call
  A_{21} ← A_{21} \cdot U^{-1}(A_{11}); \text{xTRSM() on upper triangular submatrix}
  A_{12} ← L_1^{-1}(A_{11}) \cdot A_{12}; \text{xTRSM() on lower triangular submatrix}
  A_{22} ← A_{22} - A_{21} \cdot A_{12}; \text{xGEMM()}
  rlu(A_{22}); recursive call
end.

Replace xTRSM and xGEMM with sparse implementations that are themselves recursive.
Sparse Recursive Factorization Algorithm

- **Solutions - continued**
  - Fast sparse xGEMM() is two-level algorithm
    - Recursive operation on sparse data structures
    - Dense xGEMM() call when recursion reaches single block
- Uses Reverse Cuthill-McKee ordering causing fill-in around the band
- No partial pivoting
  - Use iterative improvement or
  - Pivot only within blocks
Recursive storage conversion steps

Matrix divided into 2x2 blocks

Matrix with explicit 0’s and fill-in

Recursive algorithm division lines

- original nonzero value
0 - zero value introduced due to blocking
x - zero value introduced due to fill-in
<table>
<thead>
<tr>
<th>Matrix name</th>
<th>order</th>
<th>nonzeros</th>
<th>SuperLU time [s]</th>
<th>FERR</th>
<th>LUSR time [s]</th>
<th>FERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>af23560</td>
<td>23560</td>
<td>460598</td>
<td>44.19</td>
<td>5.8e-14</td>
<td>31.34</td>
<td>1.8e-14</td>
</tr>
<tr>
<td>ex11</td>
<td>16614</td>
<td>109694</td>
<td>109.67</td>
<td>2.5e-05</td>
<td>55.3</td>
<td>1.3e-06</td>
</tr>
<tr>
<td>goodwin</td>
<td>7320</td>
<td>324772</td>
<td>6.49</td>
<td>1.2e-08</td>
<td>6.74</td>
<td>4.6e-06</td>
</tr>
<tr>
<td>jpw_991</td>
<td>991</td>
<td>6027</td>
<td>0.19</td>
<td>2.9e-15</td>
<td>0.25</td>
<td>2.6e-15</td>
</tr>
<tr>
<td>mcf1</td>
<td>765</td>
<td>24382</td>
<td>0.07</td>
<td>1.2e-13</td>
<td>0.22</td>
<td>9.1e-13</td>
</tr>
<tr>
<td>memplus</td>
<td>17758</td>
<td>126150</td>
<td>0.29</td>
<td>2.1e-12</td>
<td>12.67</td>
<td>6.6e-13</td>
</tr>
<tr>
<td>olafu</td>
<td>16146</td>
<td>1015156</td>
<td>26.16</td>
<td>1.1e-06</td>
<td>22.1</td>
<td>3.7e-09</td>
</tr>
<tr>
<td>orsreg_1</td>
<td>2205</td>
<td>14133</td>
<td>0.46</td>
<td>1.3e-13</td>
<td>0.45</td>
<td>2.1e-13</td>
</tr>
<tr>
<td>psmigr_1</td>
<td>3140</td>
<td>543162</td>
<td>110.79</td>
<td>7.9e-11</td>
<td>88.61</td>
<td>1.2e-05</td>
</tr>
<tr>
<td>raefsky3</td>
<td>21200</td>
<td>1488768</td>
<td>62.07</td>
<td>1.4e-09</td>
<td>69.67</td>
<td>4.4e-13</td>
</tr>
<tr>
<td>raefsky4</td>
<td>19777</td>
<td>1316789</td>
<td>82.45</td>
<td>2.3e-06</td>
<td>104.29</td>
<td>3.5e-06</td>
</tr>
<tr>
<td>saylr4</td>
<td>3564</td>
<td>22316</td>
<td>0.85</td>
<td>3.2e-11</td>
<td>0.95</td>
<td>1.2e-11</td>
</tr>
<tr>
<td>sherman3</td>
<td>5005</td>
<td>20033</td>
<td>0.61</td>
<td>6.0e-13</td>
<td>0.67</td>
<td>4.8e-13</td>
</tr>
<tr>
<td>sherman5</td>
<td>3312</td>
<td>20793</td>
<td>0.28</td>
<td>1.4e-13</td>
<td>0.32</td>
<td>6.2e-15</td>
</tr>
<tr>
<td>wang3</td>
<td>26064</td>
<td>177168</td>
<td>84.14</td>
<td>2.4e-14</td>
<td>79.18</td>
<td>1.6e-14</td>
</tr>
</tbody>
</table>
Breakdown of Time Across Phases
For the Recursive Sparse Factorization

Different Test Matrices

- Numerical fact.
- Ebedded blocking
- Recursive conversion
- Block conversion
- Symbolical fact.
Use thousands of Internet-connected PCs to help in the search for extraterrestrial intelligence.

Uses data collected with the Arecibo Radio Telescope, in Puerto Rico.

When their computer is idle or being wasted this software will download a 300 kilobyte chunk of data for analysis.

The results of this analysis are sent back to the SETI team, combined with thousands of other participants.

Largest distributed computation project in existence

- ~ 400,000 machines
- Averaging 26 Tflop/s

Today many companies trying this for profit.
Distributed and Parallel Systems

Distributed systems [heterogeneous]
- SETI@home
- Entropia
- Grid based Computing
- Beowulf cluster
- Network of WS special interconnect
- Parallel Dist mem

Massively parallel systems [homogeneous]
- ASCI Tiflops

- Gather (unused) resources
- Steal cycles
- System SW manages resources
- System SW adds value
- 10% - 20% overhead is OK
- Resources drive applications
- Time to completion is not critical
- Time-shared

- Bounded set of resources
- Apps grow to consume all cycles
- Application manages resources
- System SW gets in the way
- 5% overhead is maximum
- Apps drive purchase of equipment
- Real-time constraints
- Space-shared
The Grid

♦ To treat CPU cycles and software like commodities.

♦ on steroids.

♦ Enable the coordinated use of geographically distributed resources - in the absence of central control and existing trust relationships.

♦ Computing power is produced much like utilities such as power and water are produced for consumers.

♦ Users will have access to “power” on demand
NetSolve
Network Enabled Server

- NetSolve is an example of a grid based hardware/software server.
- Easy-of-use paramount
- Based on a RPC model but with ...
  - resource discovery, dynamic problem solving capabilities, load balancing, fault tolerance asynchronicity, security, ...
- Other examples are NEOS from Argonne and NINF Japan.
- Use a resource, not tie together geographically distributed resources for a single application.
NetSolve: The Big Picture

Client

Request

Schedule

AGENT(s)

Database

Matlab
Mathematica
C, Fortran
Java, Excel

S2

Op(C, A, B)

S2

S1

A

S3

C

S4

Answer (C)

No knowledge of the grid required, RPC like.
Basic Usage
Scenarios

♦ Grid based numerical library routines
  ➢ User doesn’t have to have software library on their machine, LAPACK, SuperLU, ScaLAPACK, PETSc, AZTEC, ARPACK

♦ Task farming applications
  ➢ “Pleasantly parallel” execution
  ➢ eg Parameter studies

♦ Remote application execution
  ➢ Complete applications with user specifying input parameters and receiving output

♦ “Blue Collar” Grid Based Computing
  ➢ Does not require deep knowledge of network programming
  ➢ Level of expressiveness right for many users
  ➢ User can set things up, no “su” required
  ➢ In use today, up to 200 servers in 9 countries
Futures for Linear Algebra Numerical Algorithms and Software

- Numerical software will be adaptive, exploratory, and intelligent
- Determinism in numerical computing will be gone.
  - After all, it's not reasonable to ask for exactness in numerical computations.
  - Audibility of the computation, reproducibility at a cost
- Importance of floating point arithmetic will be undiminished.
  - 16, 32, 64, 128 bits and beyond.
- Reproducibility, fault tolerance, and auditability
- Adaptivity is a key so applications can function appropriately
Contributors to These Ideas

♦ Top500
  - Erich Strohmaier, LBL
  - Hans Meuer, Mannheim U

♦ Linear Algebra
  - Victor Eijkhout, UTK
  - Piotr Luszczek, UTK
  - Antoine Petitet, UTK
  - Clint Whaley, UTK

♦ NetSolve
  - Dorian Arnold, UTK
  - Susan Blackford, UTK
  - Henri Casanova, UCSD
  - Michelle Miller, UTK
  - Sathish Vadhiyar, UTK

For additional information see...

www.netlib.org/netsolve/
www.netlib.org/atlas/
www.cs.utk.edu/~dongarra/

Many opportunities within the group at Tennessee
The Materials are pre-release code, which may not be fully functional and which Intel may substantially modify in producing any final version. Intel can provide no assurance that it will ever produce or make generally available a final version. You agree to maintain as confidential all information relating to your use of the Materials and not to disclose to any third party any benchmarks, performance results, or other information relating to the Materials comprising the pre-release.

See: