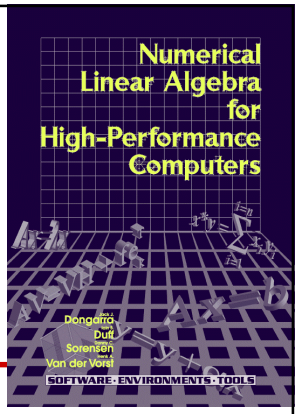
EuroPar 2000 Tutorial on High-Performance Numerical Linear Algebra: Fast and Robust Kernels for Scientific Computing



Jack Dongarra, University of Tennessee
Iain Duff, Rutherford-Appleton Lab & CERFACS
Danny Sorensen, Rice University
Henk van der Vorst, Utrecht University

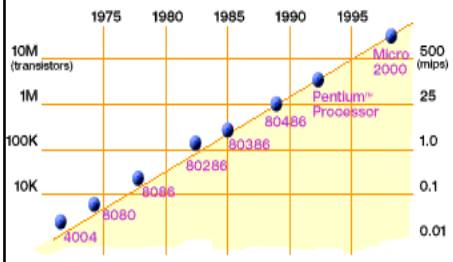
Euro-Par 2000
Technische Universität München
29.8.–1.9.2000

Outline

- Overview of HPC and Dense Methods
  - Jack Dongarra
  - 9:00 – 10:30
- Sparse Direct Methods for $Ax = b$
  - Iain Duff
  - 11:00 – 12:30
- Lunch 12:30 – 2:00
- Sparse Iterative Methods for $Ax = b$
  - Henk van der Vorst
  - 2:00 – 3:30
- Sparse Iterative Methods for $Ax = \lambda x$
  - Danny Sorensen
  - 4:00 – 5:30

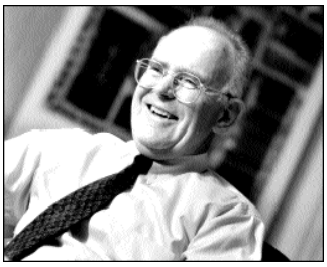
2

Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years
Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

3

High Performance Computers & Numerical Libraries

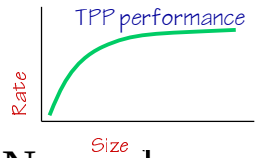
- 20 years ago
  - $1x10^6$ Floating Point Ops/sec (Mflop/s)
    - Scalar based
    - Loop unrolling
- 10 years ago
  - $1x10^9$ Floating Point Ops/sec (Gflop/s)
    - Vector & Shared memory computing, bandwidth aware
    - Block partitioned, latency tolerant
- Today
  - $1x10^{12}$ Floating Point Ops/sec (Tflop/s)
    - Highly parallel, distributed processing, message passing, network based
    - data decomposition, communication/computation
- 10 years away
  - $1x10^{15}$ 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

4

TOP500

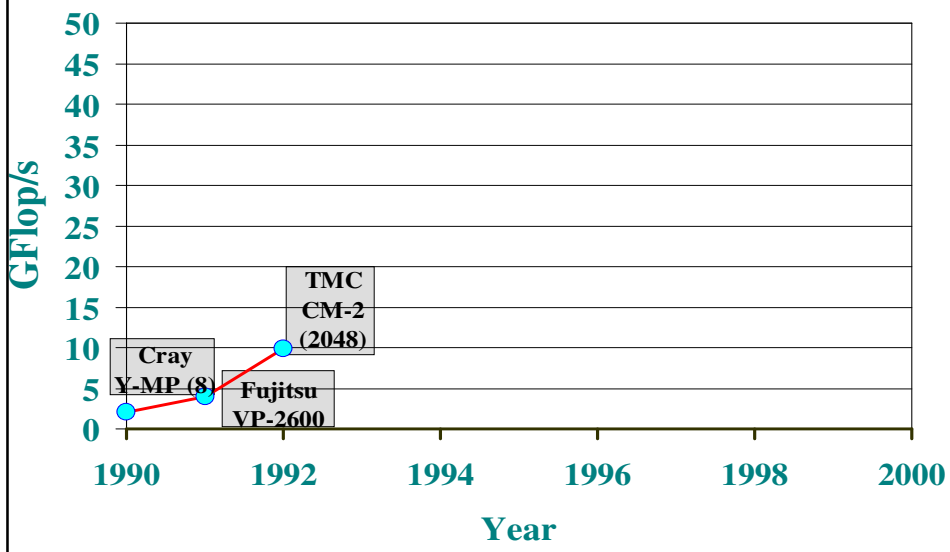
- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

  $Ax=b$, *dense problem*

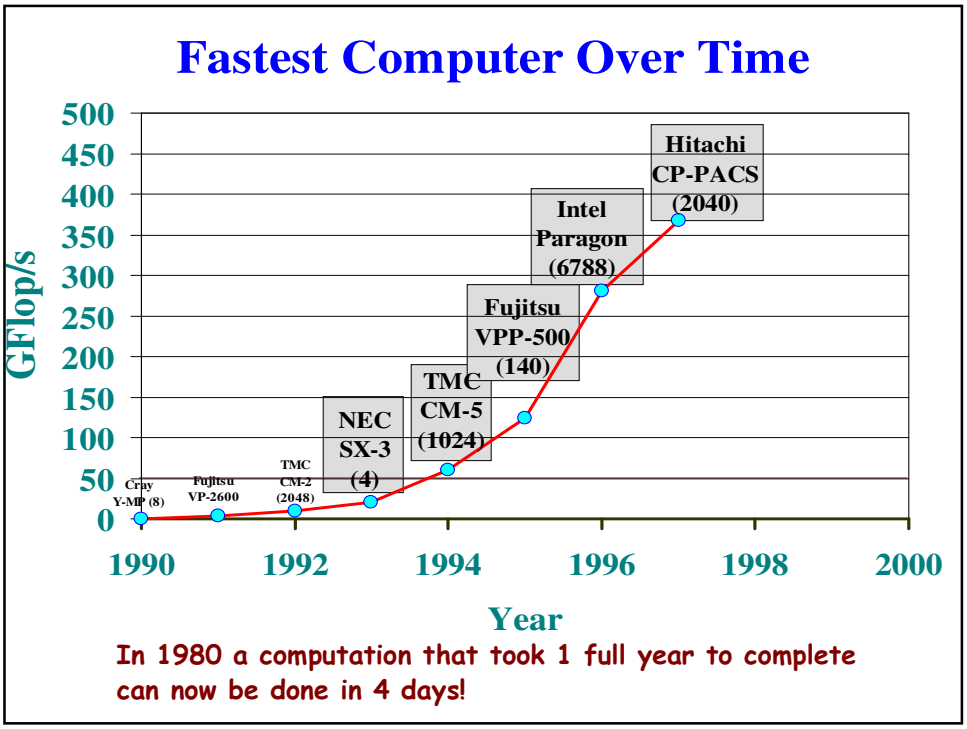


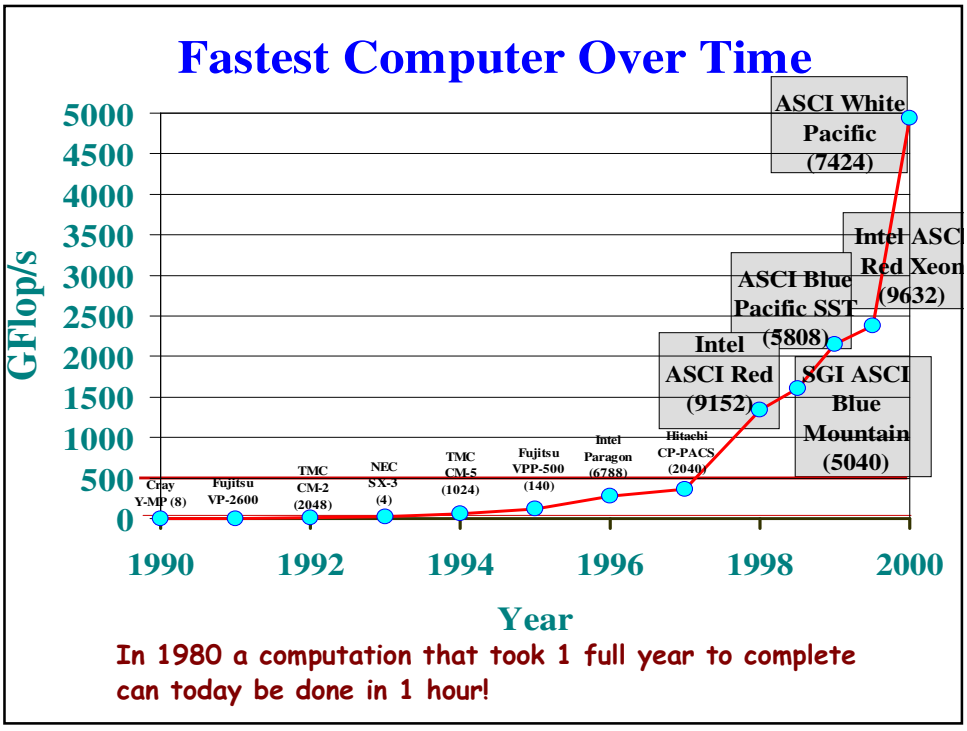
- Updated twice a year
  SC'xy in the States in November
  Meeting in Mannheim, Germany in June
- All data available from **www.top500.org**

Fastest Computer Over Time



In 1980 a computation that took 1 full year to complete can now be done in 1 month!

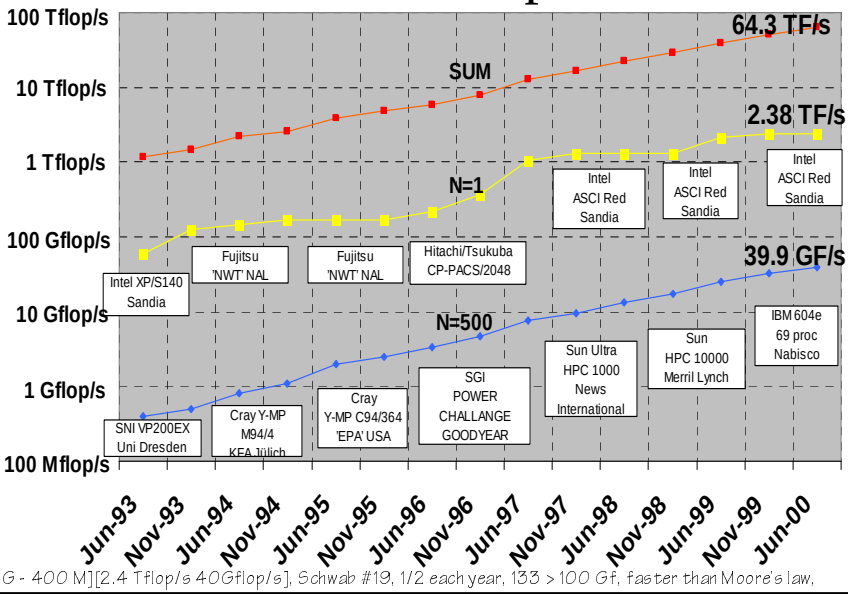

Fastest Computer Over Time
GFlop/s
500
450
400
350
300
250
200
150
100
50
0
1990
1992
1994
1996
1998
2000
Year
Cray Y-MP (8)
Fujitsu VP-2600
TMC CM-2 (2048)
NEC SX-3 (4)
TMC CM-5 (1024)
Fujitsu VPP-500 (140)
Intel Paragon (6788)
Hitachi CP-PACS (2040)
In 1980 a computation that took 1 full year to complete can now be done in 4 days!


Fastest Computer Over Time
GFlop/s
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1990
1992
1994
1996
1998
2000
Year
Cray Y-MP (8)
Fujitsu VP-2600
TMC CM-2 (2048)
NEC SX-3 (4)
TMC CM-5 (1024)
Fujitsu VPP-500 (140)
Intel Paragon (6788)
Hitachi CP-PACS (2040)
Intel ASCI Red (9152)
ASCI Blue Pacific SST (5808)
Intel ASCI Red Xeon (9632)
SGI ASCI Blue Mountain (5040)
ASCI White Pacific (7424)
In 1980 a computation that took 1 full year to complete can today be done in 1 hour!

Top 10 Machines (June 2000)

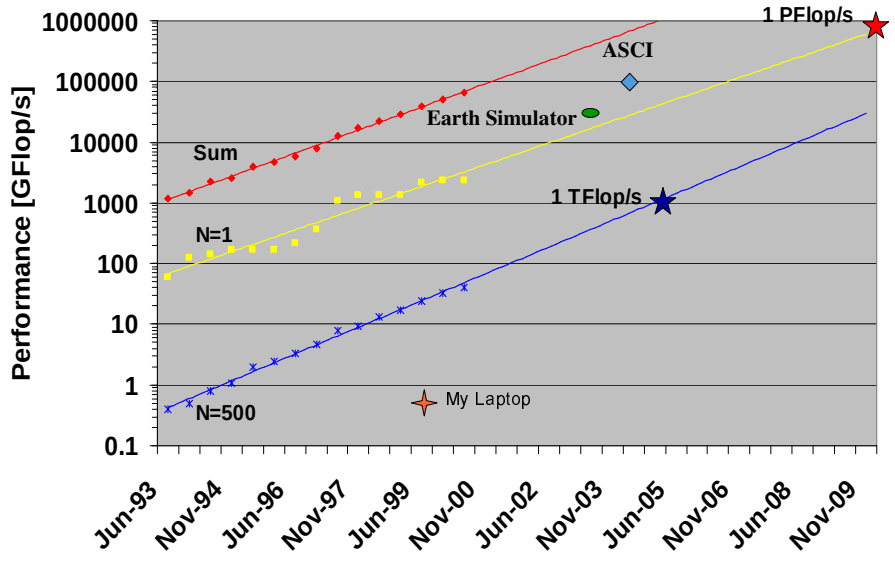
Rank	Company	Machine	Procs	Gflop/s	Place	Country	Year
1	Intel	ASCI Red	9632	2380	Sandia National Labs Albuquerque	USA	1999
2	IBM	ASCI Blue-Pacific SST, IBM SP 604e	5808	2144	Lawrence Livermore National Laboratory Livermore	USA	1999
3	SGI	ASCI Blue Mountain	6144	1608	Los Alamos National Laboratory Los Alamos	USA	1998
4	Hitachi	SR8000-F1/112	112	1035	Leibniz Rechenzentrum Muenchen	Germany	2000
5	Hitachi	SR8000-F1/100	100	917	High Energy Accelerator Research Organization /KEK Tsukuba	Japan	2000
6	Cray Inc.	T3E1200	1084	892	Government	USA	1998
7	Cray Inc.	T3E1200	1084	892	US Army HPC Research Center at NCS Minneapolis	USA	2000
8	Hitachi	SR8000/128	128	874	University of Tokyo Tokyo	Japan	1999
9	Cray Inc.	T3E900	1324	815	Government	USA	1997
10	IBM	SP Power3 375 MHz	1336	723	Naval Oceanographic Office (NAVOCEANO) Poughkeepsie	USA	2000

Performance Development



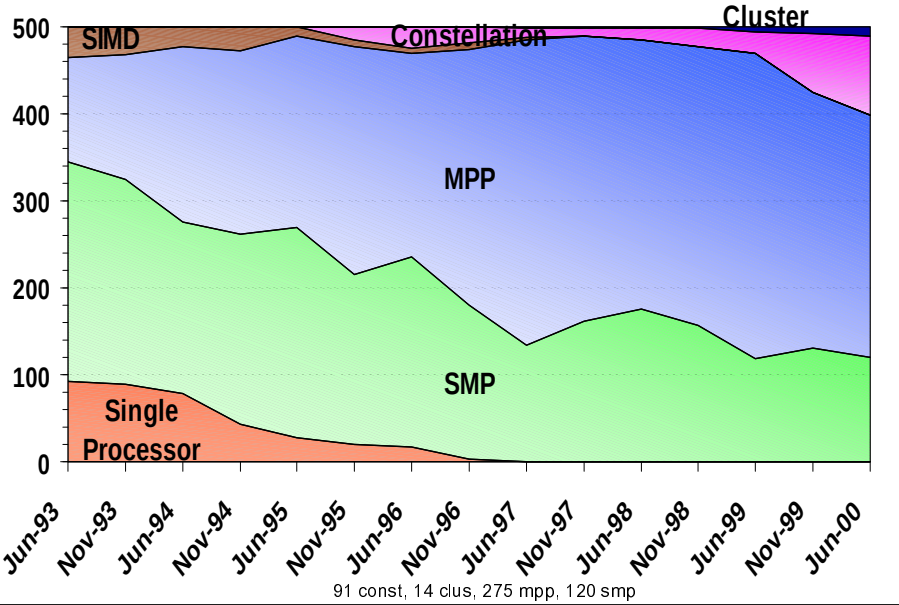
[60G - 400 M][2.4 Tflop/s 40Gflop/s], Schwab #19, 1/2 each year, 133 > 100 Gf, faster than Moore's law.

Performance Development



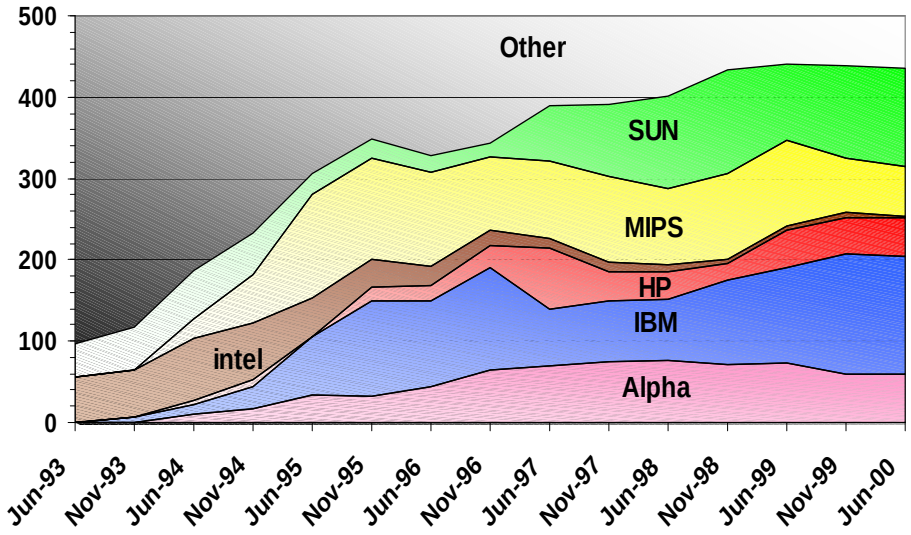
Entry 1 T 2005 and 1 P 2010

Architectures

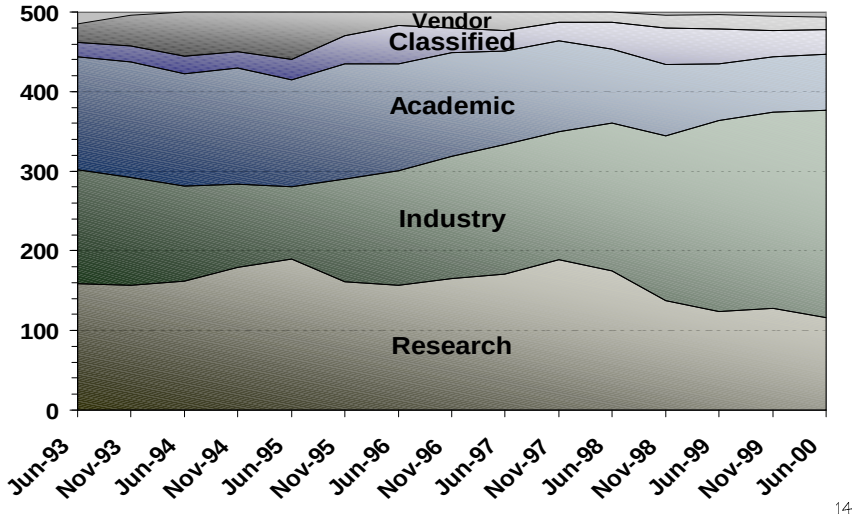


91 const, 14 clus, 275 mpp, 120 smp

Chip Technology



Customer Type



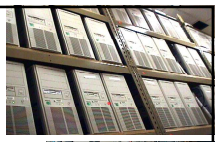
14

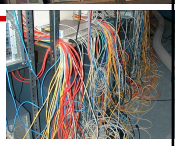
23 research, 50 industry, 14 academic

Clusters on the Top500

Rank	Manufacturer	Computer	Rmax	Installation Site	Country	Year	Area of Installation	# Proc	Rpeak
50	Sun	HPC 4500 Cluster	**272.1**	Sun Burlington	USA	1998	Vendor	720	483.84
62	Self-made	CPlant Cluster	**232.6**	Sandia National Laboratories Albuquerque	USA	1999	Research	580	580
187	Sun	HPC 10000 400 MHz Cluster	**68.77**	KT Freetel Seoul	Korea	1999	Industry Telecomm	110	88
207	Self-made	NT Supercluster	**62.59**	NCSA Urbana-Champaign	USA	1999	Research	256	140.8
237	Compaq	Alphleet Cluster	**61.3**	Institute of Physical and Chemical Res. (RIKEN) Wako	Japan	1999	Research	140	140
271	SGI	ORIGIN 2000 250 MHz - Eth-Cluster	**54.68**	The Sabre Group Ft Worth	USA	1999	Industry Transportation	448	224
272	SGI	ORIGIN 2000 250 MHz - Eth-Cluster	**54.68**	America On Line (AOL)	USA	1999	Industry WWW	320	160
273	SGI	ORIGIN 2000 250 MHz - Eth-Cluster	**54.68**	Industrial Light & Magic	USA	1999	Industry Image Proc./Rendering	224	11.2
364	Self-made	Avalon Cluster	**48.6**	Los Alamos National Laboratory /CNLS Los Alamos	USA	1998	Academic	140	149.4
366	SGI	ORIGIN 2000 300 MHz - Eth-Cluster	**48.33**	Industrial Light & Magic	USA	1999	Industry Image Proc./Rendering	128	76.8
367	SGI	ORIGIN 2000 300 MHz - Eth-Cluster	**48.33**	Sikorsky Stratford	USA	2000	Industry Aerospace	128	76.8
368	SGI	ORIGIN 2000 300 MHz - Eth-Cluster	**48.33**	Toshiba Tokyo	Japan	2000	Industry Electronics	128	76.8
381	Self-made	AC3 Velocity/MS NT4	**47.38**	Cornell Theory Center Ithaca	USA	1999	Academic	256	126
394	Sun	HPC 10000 400 MHz Cluster	**45.46**	Boeing IDS Group Orange County	USA	1999	Industry Aerospace	72	57.6

High-Performance Computing Directions: Beowulf-class PC Clusters





Definition:

- **COTS PC Nodes**
  - Pentium, Alpha, PowerPC, SMP
- **COTS LAN/SAN Interconnect**
  - Fast e-net, 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.

16

The Maturation of Highly Parallel Technology

- Affordable parallel systems with distributed memory are the architecture of choice for price performance.
- Reliability of systems is greatly improved.
- Third-party scientific and engineering applications are appearing.
- Business applications are appearing.
- Commercial customers, not just research labs, are acquiring systems.

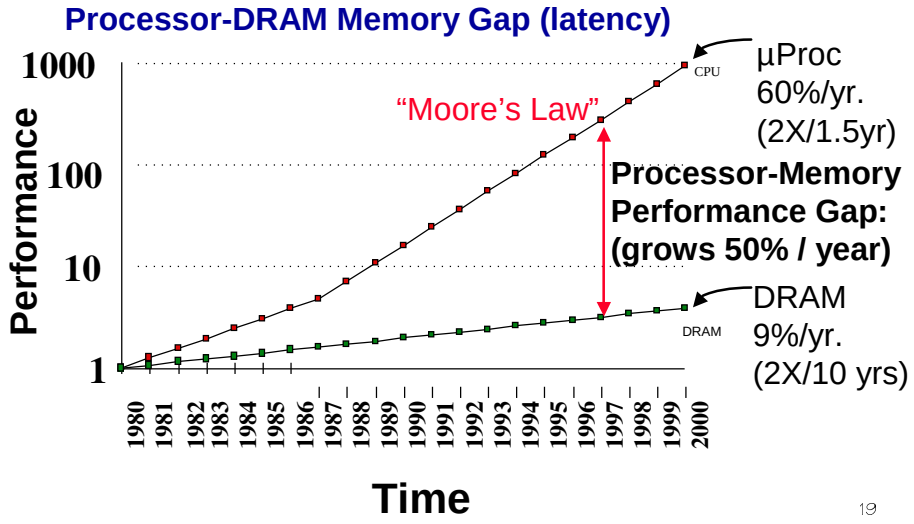
17

Challenges in Developing Distributed Memory Libraries

- How to integrate software?
  - Until recently no standards
  - Many parallel languages
  - Various parallel programming models
  - Assumptions about the parallel environment
    - » granularity
    - » topology
    - » overlapping of communication/computation
    - » development tools
- Where is the data
  - Who owns it?
  - Opt data distribution
- Who determines data layout
  - Determined by user?
  - Determined by library developer?
  - Allow dynamic data dist.
  - Load balancing

18

Where Does the Performance Go? or Why Should I Cares About the Memory Hierarchy?



19

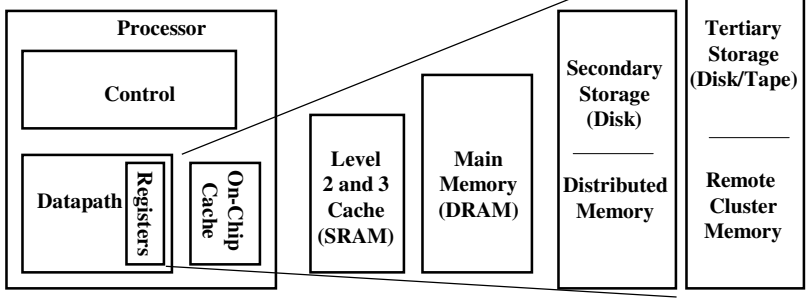
Optimizing Computation and Memory Use

- **Computational optimizations**
  - **Theoretical peak: (# fpus)*(flops/cycle) * Mhz**
    - PIII: (1 fpu)*(1 flop/cycle)*(650 Mhz) = 650 MFLOP/s
    - Athlon: (2 fpu)*(1flop/cycle)*(600 Mhz) = 1200 MFLOP/s
    - Power3: (2 fpu)*(2 flops/cycle)*(375 Mhz) = 1500 MFLOP/s
- **Memory optimization**
  - **Theoretical peak: (bus width) * (bus speed)**
    - PIII : (32 bits)*(133 Mhz) = 532 MB/s = 66.5 MW/s
    - Athlon: (64 bits)*(200 Mhz) = 1600 MB/s = 200 MW/s
    - Power3: (128 bits)*(100 Mhz) = 1600 MB/s = 200 MW/s
- **Memory about an order of magnitude slower**

20

Memory Hierarchy

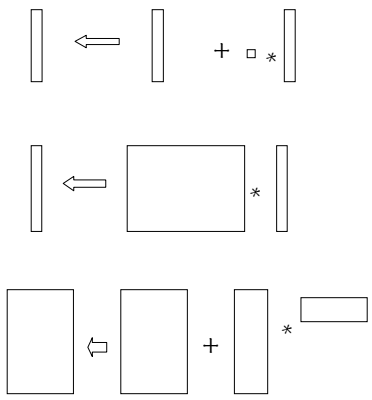
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	100,000 s (.1s ms)	10,000,000 s (10s ms)
				Gs	Ts

Level 1, 2 and 3 BLAS

- Level 1 BLAS Vector-Vector operations
- Level 2 BLAS Matrix-Vector operations
- Level 3 BLAS Matrix-Matrix operations

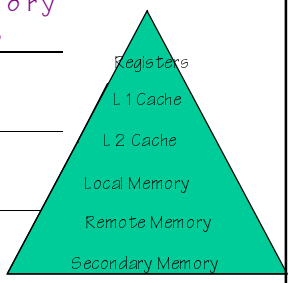


22

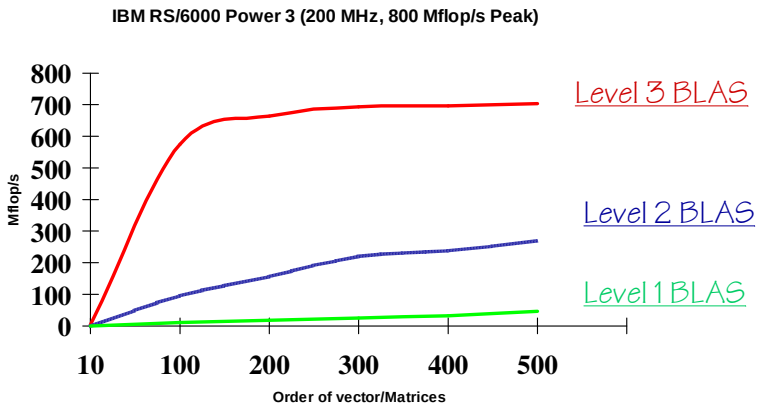
Why Higher Level BLAS?

- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this

BLAS	Memory Refs	Flops	Flops/ Memory Refs
Level 1 $y = y + \alpha x$	$3n$	$2n$	$2/3$
Level 2 $y = y + Ax$	n^2	$2n^2$	2
Level 3 $C = C + AB$	$4n^2$	$2n^3$	$n/2$



BLAS for Performance



- Development of blocked algorithms important for performance

24

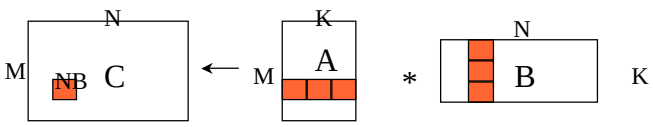
How To Get Performance From Commodity Processors?

- Today's processors can achieve high-performance, but this requires extensive machine-specific hand tuning.
- Routines 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 microarchitectures of new microprocessors.
- A few months ago no tuned BLAS for Pentium for Linux.
- Need for quick/dynamic deployment of optimized routines.
- ATLAS - Automatic Tuned Linear Algebra Software
  - PhiPac from Berkeley
  - FFTW from MIT

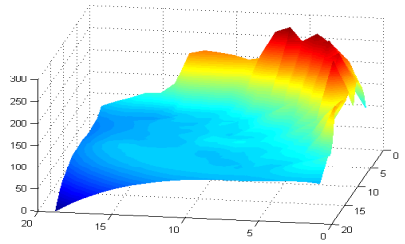
25

Adaptive Approach for Level 3

- Do a parameter study of the operation on the target machine, done once.
- Only generated code is on-chip multiply
- BLAS operation written in terms of generated on-chip multiply
- All tranpose cases coerced through data copy to 1 case of on-chip multiply
  - Only 1 case generated per platform



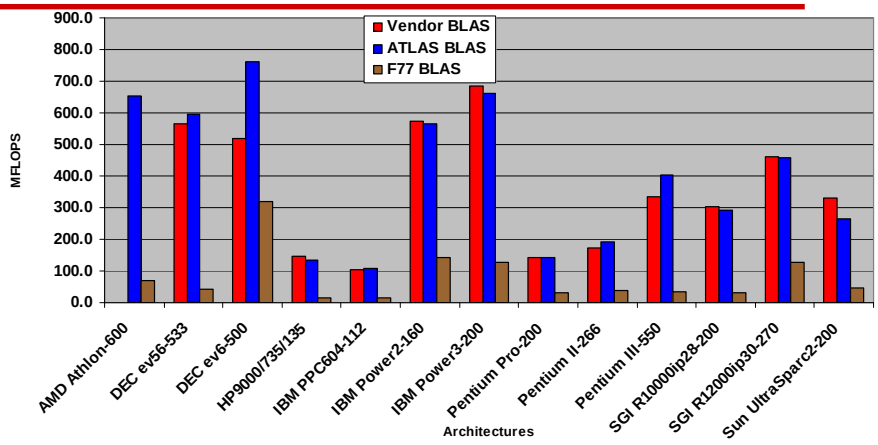
Code Generation Strategy



- On-chip multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization
- Takes a couple of hours to run.
- Code is iteratively generated & timed until optimal case is found. We try:
  - Differing NBs
  - Breaking false dependencies
  - M, N and K loop unrolling

27

ATLAS Across Various Architectures (DGEMM n=500)



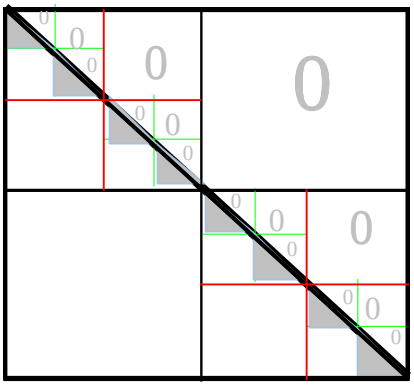
- ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.

28

Recursive Approach for Other Level 3 BLAS

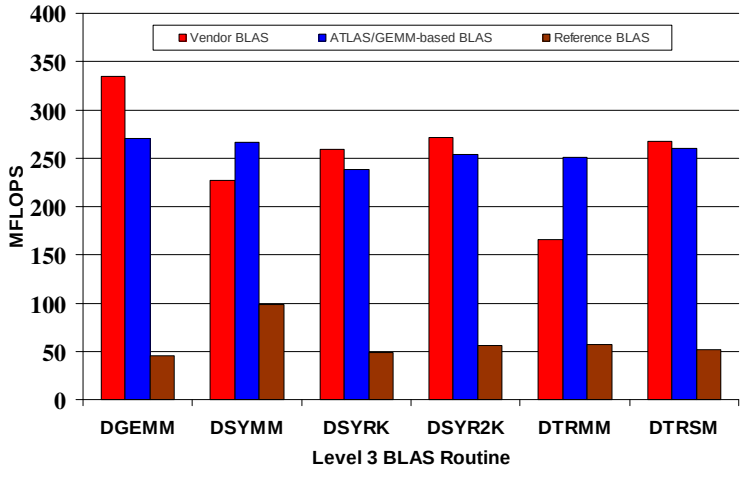
- Recur down to L1 cache block size
- Need kernel at bottom of recursion
  - Use gemm-based kernel for portability

Recursive TRMM

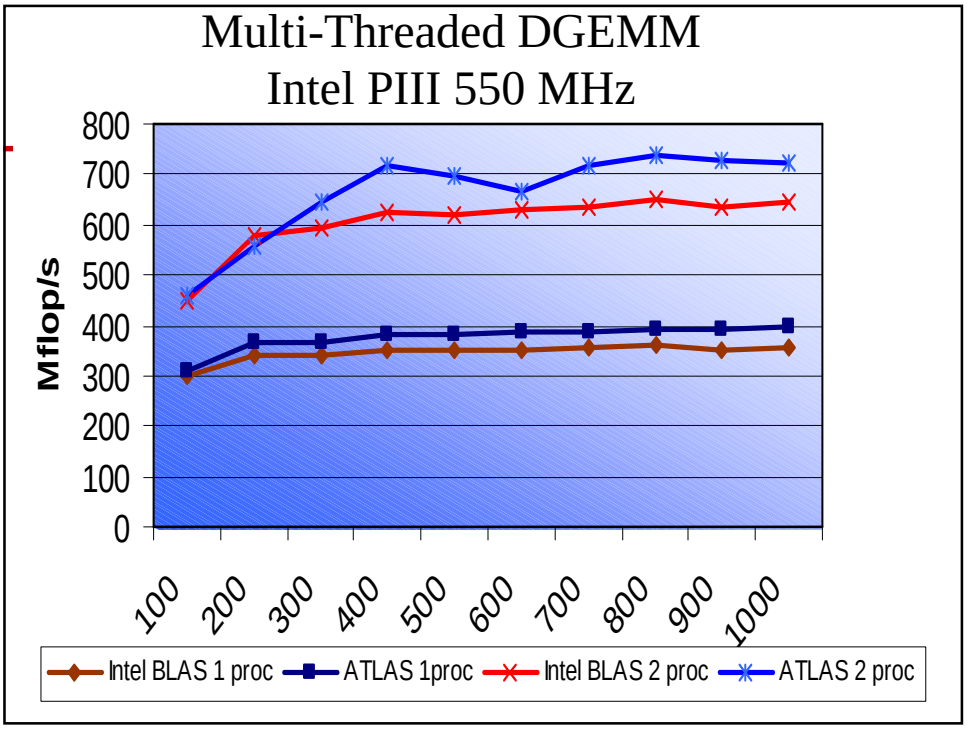


29

500x500 Recursive BLAS on UltraSparc 2200



30



Plans for ATLAS

- **Software Release, available today:**
  - Level 1, 2, and 3 BLAS implementations
  - See: www.netlib.org/atlas/
- **Next Version:**
  - Multi-treading
  - Java generator
- **Futures:**
  - Optimize message passing system
  - Runtime adaptation
    - » Sparsity analysis
    - » Iterative code improvement
  - Specialization for user applications
  - Adaptive libraries

32

Tools for Performance Evaluation

- Timing and performance evaluation has been an art
  - Resolution of the clock
  - Issues about cache effects
  - Different systems
- Situation about to change
  - Today's processors have internal counters

33

Performance Counters

- Almost all high performance processors include hardware performance counters.
- Some are easy to access, others not available to users.
- On most platforms the APIs, if they exist, are not appropriate for a common user, functional or well documented.
- Existing performance counter APIs
  - Cray T3E
  - SGI MIPS R10000
  - IBM Power series
  - DEC Alpha pfm pseudo-device interface
  - Windows 95, NT and Linux

34

Performance Data That May Be Available

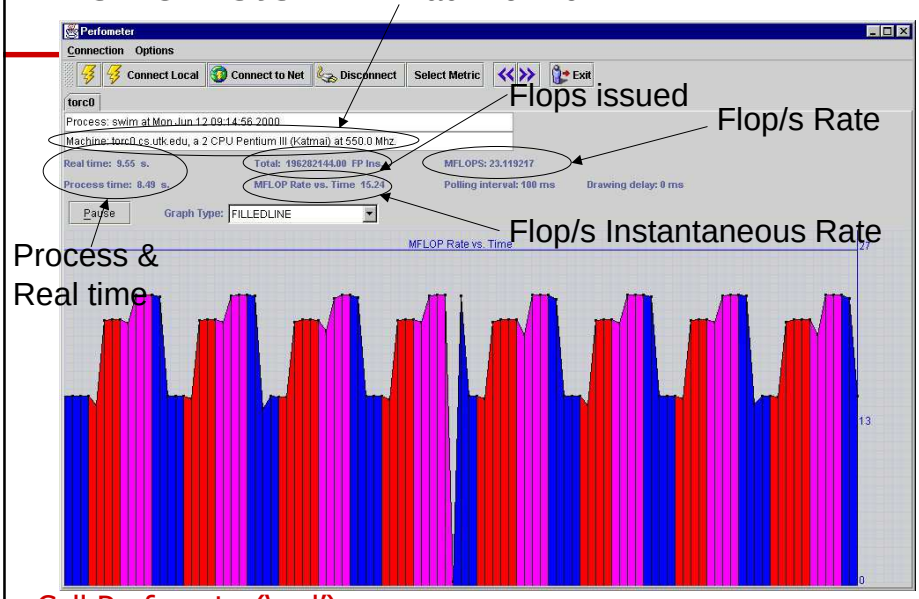
- Cycle count
- Floating point instruction count
- Integer instruction count
- Instruction count
- Load/store count
- Branch taken / not taken count
- Branch mispredictions
- Pipeline stalls due to memory subsystem
- Pipeline stalls due to resource conflicts
- I/D cache misses for different levels
- Cache invalidations
- TLB misses
- TLB invalidations

35

PAPI's Graphical Tools Perfometer Usage

- Application is instrumented with PAPI
  - call perfometer()
- Will be layered over the best existing vendor-specific APIs for these platforms
- Sections of code that are of interest are designated with specific colors
  - Using a call to set_perfometer('color')
- Application is started, at the call to perfometer a task is spawned to collect and send the information to a Java applet containing the graphical view.

Perfometer



Call Perfometer('red')

Go To Demo

38

EISPACK and LINPACK

- **EISPACK**
  - Design for the algebraic eigenvalue problem, $Ax = \lambda x$ and $Ax = \lambda Bx$.
  - work of J. Wilkinson and colleagues in the 70's.
  - Fortran 77 software based on translation of ALGOL.
- **LINPACK**
  - Design for the solving systems of equations, $Ax = b$.
  - Fortran 77 software using the Level 1 BLAS.

39

History of Block Partitioned Algorithms

- Early algorithms involved use of small main memory using tapes as secondary storage.
- Recent work centers on use of vector registers, level 1 and 2 cache, main memory, and "out of core" memory.

40

Blocked Partitioned Algorithms

- LU Factorization
- Cholesky factorization
- Symmetric indefinite factorization
- Matrix inversion
- QR, QL, RQ, LQ factorizations
- Form Q or Q^TC
- Orthogonal reduction to:
  - (upper) Hessenberg form
  - symmetric tridiagonal form
  - bidiagonal form
- Block QR iteration for nonsymmetric eigenvalue problems

41

LAPACK

- Linear Algebra library in Fortran 77
  - Solution of systems of equations
  - Solution of eigenvalue problems
- Combine algorithms from LINPACK and EISPACK into a single package
- Efficient on a wide range of computers
  - RISC, Vector, SMPs
- User interface similar to LINPACK
  - Single, Double, Complex, Double Complex
- Built on the Level 1, 2, and 3 BLAS

42

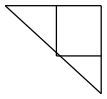
LAPACK

- Most of the parallelism in the BLAS.
- Advantages of using the BLAS for parallelism:
  - Clarity
  - Modularity
  - Performance
  - Portability

43

Derivation of Blocked Algorithms Cholesky Factorization $A = U^T U$

$$\begin{pmatrix} A_{11} & a_j & A_{13} \\ a_j^T & a_{jj} & \alpha_j^T \\ A_{13}^T & \alpha_j & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ u_j^T & u_{jj} & 0 \\ U_{13}^T & \mu_j & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & u_j & U_{13} \\ 0 & u_{jj} & \mu_j^T \\ 0 & 0 & U_{33} \end{pmatrix}$$



Equating coefficient of the j^{th} column, we obtain

$$a_j = U_{11}^T u_j$$
$$a_{jj} = u_j^T u_j + u_{jj}^2$$

Hence, if U_{11} has already been computed, we can compute u_j and u_{jj} from the equations:

$$U_{11}^T u_j = a_j$$
$$u_{jj}^2 = a_{jj} - u_j^T u_j$$

44

LINPACK Implementation

- Here is the body of the LINPACK routine SPOFA which implements the method:

```
      DO 30 J = 1, N
         INFO = J
         S = 0.0E0
         JM1 = J - 1
         IF( JM1.LT.1 ) GO TO 20
         DO 10 K = 1, JM1
            T = A( K, J ) - SDOT( K-1, A( 1, K ), 1,A( 1, J ), 1 )
            T = T / A( K, K )
            A( K, J ) = T
            S = S + T*T
   10    CONTINUE
   20    CONTINUE
         S = A( J, J ) - S
C     ...EXIT
         IF( S.LE.0.0E0 ) GO TO 40
         A( J, J ) = SQRT( S )
   30 CONTINUE
```

45

LAPACK Implementation

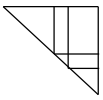
```
      DO 10 J = 1, N
      CALL STRSV( 'Upper', 'Transpose', 'Non-Unit', J-1, A, LDA, A( 1, J ), 1 )
      S = A( J, J ) - SDOT( J-1, A( 1, J ), 1, A( 1, J ), 1 )
      IF( S.LE.ZERO ) GO TO 20
      A( J, J ) = SQRT( S )
10 CONTINUE
```

- This change by itself is sufficient to significantly improve the performance on a number of machines.
- From 72 to 251 Mflop/s for a matrix of order 500 on one processor of a CRAY Y-MP.
- However on 378 Mflop/s on 8 Procs. Of a CRAY Y-MP.
- Suggest further work needed.

46

Derivation of Blocked Algorithms

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{12}^T & A_{22} & A_{12} \\ A_{13}^T & A_{12}^T & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ U_{12}^T & U_{22}^T & 0 \\ U_{13}^T & U_{23}^T & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23}^T \\ 0 & 0 & U_{33} \end{pmatrix}$$



Equating coefficient of second block of columns, we obtain

$$A_{12} = U_{11}^T U_{12}$$

$$A_{22} = U_{12}^T U_{12} + U_{22}^T U_{22}$$

Hence, if U_{11} has already been computed, we can compute U_{12} as the solution of the following equations by a call to the Level 3 BLAS routine STRSM:

$$U_{11}^T U_{12} = A_{12}$$

$$U_{22}^T U_{22} = A_{22} - U_{12}^T U_{12}$$

47

LAPACK Blocked Algorithms

```
      DO 10 J = 1, N, NB
         CALL STRSM( 'Left', 'Upper', 'Transpose','Non-Unit', J-1, JB, ONE, A, LDA,
     $              A( 1, J ), LDA )
         CALL SSYRK( 'Upper', 'Transpose', JB, J-1,-ONE, A( 1, J ), LDA, ONE,
     $              A( J, J ), LDA )
         CALL SPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
         IF( INFO.NE.0 ) GO TO 20
   10 CONTINUE
```

- On Y-MP, L3 BLAS squeezes a little more out of 1 proc, but makes a large improvement when using 8 procs.

Cray Y-MP Cholesky, n=500	1 Proc	8 procs
j-varient LINPACK	72	72
j-varient Level 2 BLAS	251	378
j-varient Level 3 BLAS	287	1225
k-varient Level 3 BLAS	290	1414

48

LAPACK Contents

- Combines algorithms from LINPACK and EISPACK into a single package. User interface similar to LINPACK.
- Built on the Level 1, 2 and 3 BLAS, for high performance (manufacturers optimize BLAS)
- LAPACK does not provide routines for structured problems or general sparse matrices (i.e sparse storage formats such as compressed-row, -column, -diagonal, skyline ...).

49

LAPACK Ongoing Work

- Add functionality
  - updating/downdating, divide and conquer least squares,bidiagonal bisection, bidiagonal inverse iteration, band SVD, Jacobi methods, ...
- Move to new generation of high performance machines
  - IBM SPs, CRAY T3E, SGI Origin, clusters of workstations
- New challenges
  - New languages: FORTRAN 90, HP FORTRAN, ...
  - (CMMD, MPL, NX ...)
    - » many flavors of message passing, need standard (PVM, MPI): BLACS
- Highly varying ratio $\frac{\text{Computational speed}}{\text{Communication speed}}$
- Many ways to layout data,
- Fastest parallel algorithm sometimes less stable numerically.

50

ScaLAPACK

- Library of software dealing with dense & banded routines
- Distributed Memory - Message Passing
- MIMD Computers and Networks of Workstations
- Clusters of SMPs

51

Programming Style

- SPMD Fortran 77 with object based design
- Built on various modules
  - PBLAS Interprocessor communication
  - BLACS
    - » PVM, MPI, IBM SP, CRI T3, Intel, TMC
    - » Provides right level of notation.
  - BLAS
- LAPACK software expertise/quality
  - Software approach
  - Numerical methods

52

Overall Structure of Software

- Object based - Array descriptor
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
  - Currently dense, banded, & out-of-core
- Using the concept of context

53

PBLAS

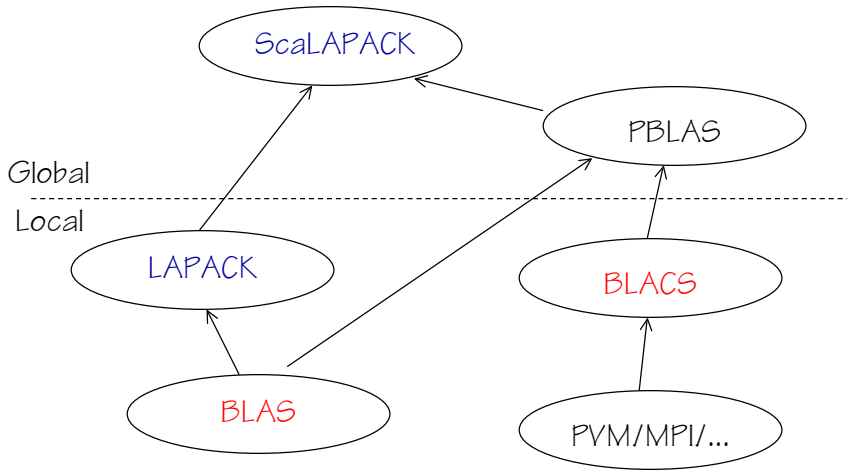
- Similar to the BLAS in functionality and naming.
- Built on the BLAS and BLACS
- Provide global view of matrix

```
CALL DGEXXX   ( M, N, A( IA, JA ), LDA,... )
```

```
CALL PDGEXXX( M, N, A, IA, JA, DESCA,... )
```

54

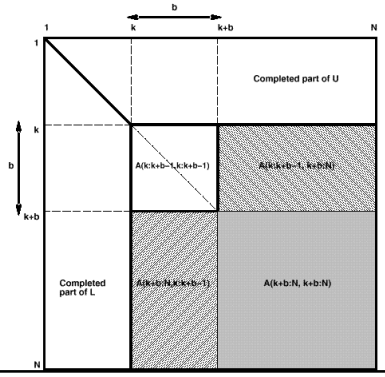
ScaLAPACK Structure



55

Choosing a Data Distribution

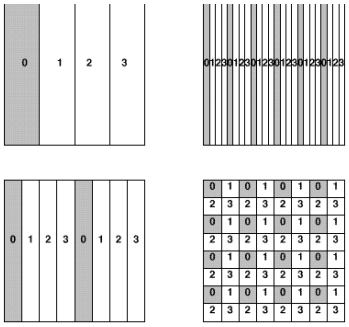
- **Main issues are:**
  - **Load balancing**
  - **Use of the Level 3 BLAS**



56

Possible Data Layouts

- 1D block and cyclic column distributions



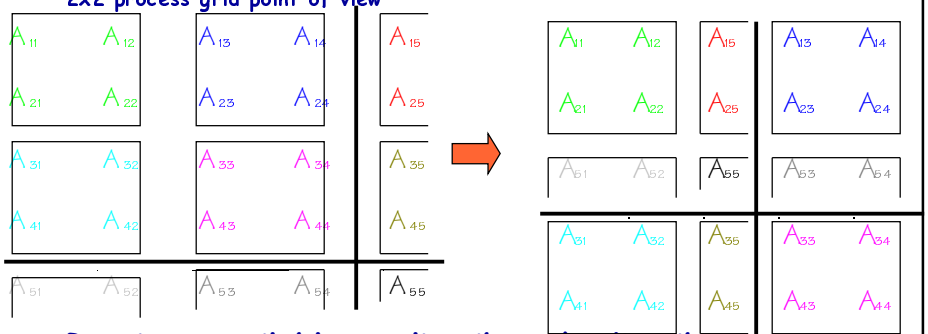
- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

57

Distribution and Storage

- Matrix is block-partitioned & maps blocks
- Distributed 2-D block-cyclic scheme

5x5 matrix partitioned in 2x2 blocks
2x2 process grid point of view



- Routines available to distribute/redistribute data.

58

Parallelism in ScaLAPACK

- **Level 3 BLAS block operations**
  - All the reduction routines
- **Pipelining**
  - QR Algorithm, Triangular Solvers, classic factorizations
- **Redundant computations**
  - Condition estimators
- **Static work assignment**
  - Bisection
- **Task parallelism**
  - Sign function eigenvalue computations
- **Divide and Conquer**
  - Tridiagonal and band solvers, symmetric eigenvalue problem and Sign function
- **Cyclic reduction**
  - Reduced system in the band solver
- **Data parallelism**
  - Sign function

59

ScaLAPACK - What's Included

Problem type Ax = b	SDrv	EDrv	Factor	Solve	Inv	Cond Est	Iter Refir
Triangular				X	X	X	X
SPD	X	X	X	X	X	X	X
SPD Banded	X		X	X			
SPD Tridiagonal	X		X	X			
General	X	X	X	X	X	X	X
General Banded	X		X	X			
General Tridiagonal	X		X	X			
Least squares	X		X	X			
GQR			X				
GRQ			X				

Ax = λx or Ax = λBx	SDrv	Edrv	Reduct	Solution
Symmetric (2 types)	X	X	X	X
General (2 types)			X	X
Generalized BSPD		X	X	X
SVD			X	X

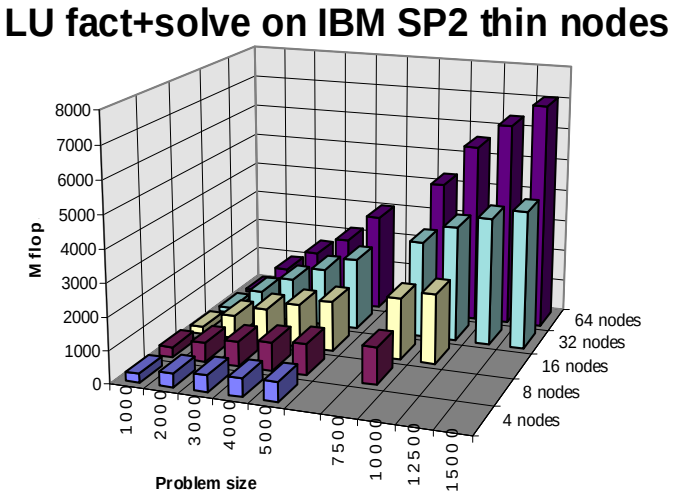
- **Timing and Testing routines for almost all**
- **This is a large component of the package**
- **Prebuilt libraries available for SP, PGON, HPPA, DEC, Sun, RS6K**

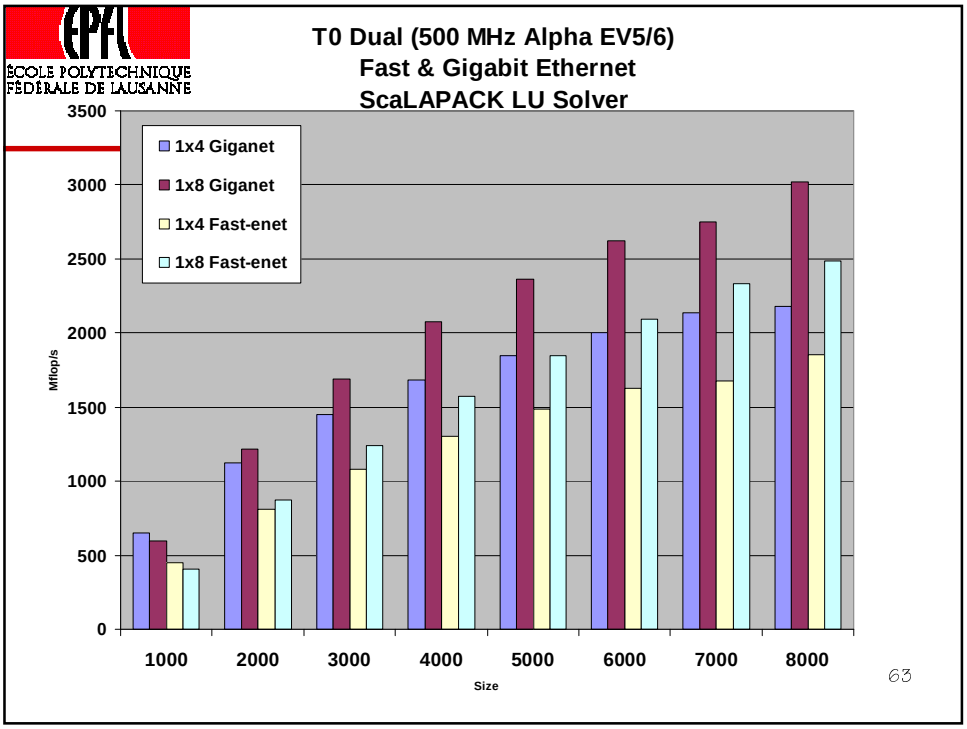
Heterogeneous Computing

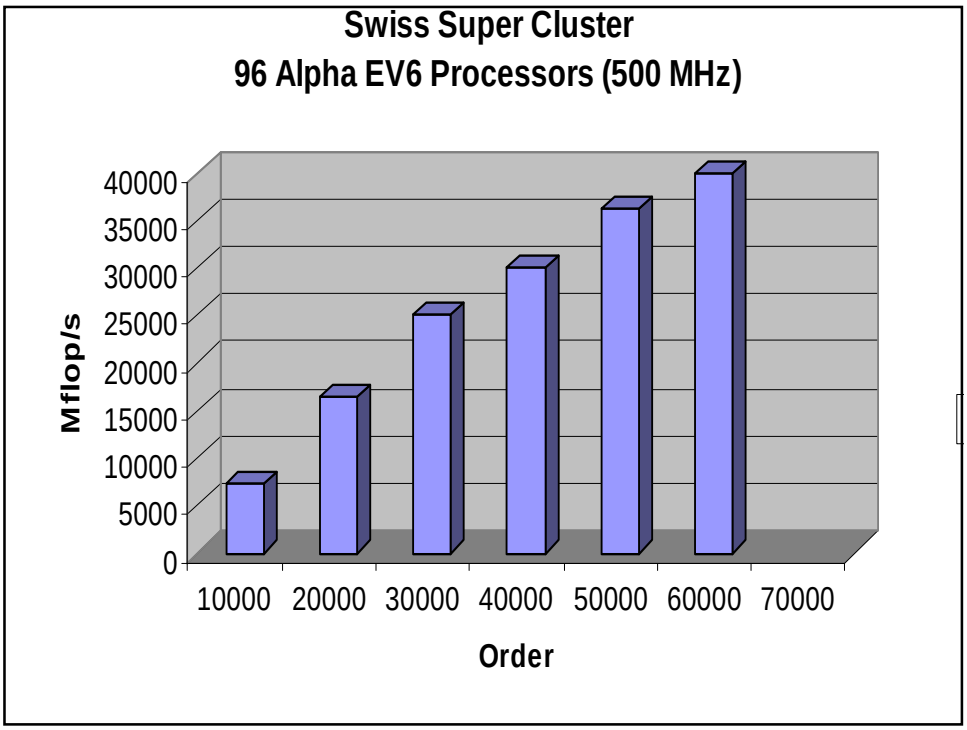
- Software intended to be used in this context
- Communication of ft. pt. numbers between processors
- Machine precision and other machine specific parameters
- Iterative convergence across clusters of processors
- Defensive programming required

61

Performance




ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
T0 Dual (500 MHz Alpha EV5/6)
Fast & Gigabit Ethernet
ScaLAPACK LU Solver
1x4 Giganet
1x8 Giganet
1x4 Fast-enet
1x8 Fast-enet
Mflop/s
3500
3000
2500
2000
1500
1000
500
0
1000
2000
3000
4000
5000
6000
7000
8000
Size
63

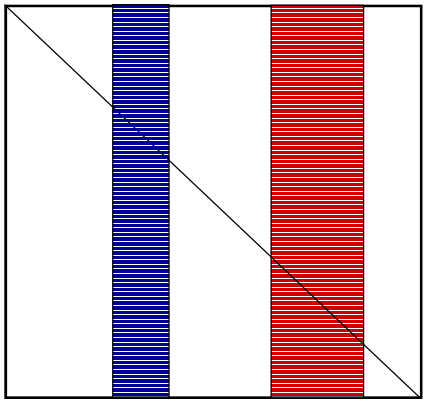

Swiss Super Cluster
96 Alpha EV6 Processors (500 MHz)
Mflop/s
40000
35000
30000
25000
20000
15000
10000
5000
0
10000
20000
30000
40000
50000
60000
70000
Order

Out of Core Approach

- High-level I/O Interface
- ScaLAPACK uses a `Right-looking' variant for LU, QR and Cholesky factorizations.
- A `Left-looking' variant is used for Out-of-core factorization to reduce I/O traffic.
- Requires two in-core column panels.
- Imposes another level in the memory hierarchy.

65

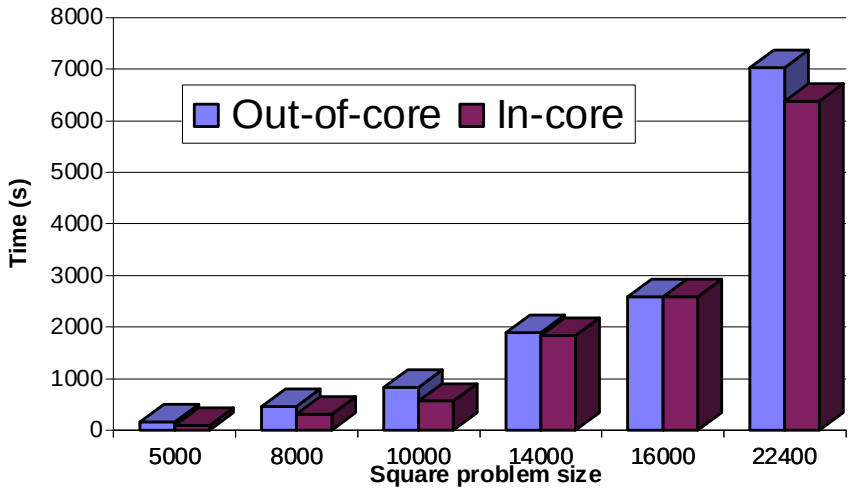
Out of Core Algorithm



- Hybrid approach
- Algorithm is ``Left-Looking'' in nature, but uses ``Right-Looking'' (ScaLAPACK) on Panel B
- Latency Tolerant
- Model for deep memory hierarchy algorithms

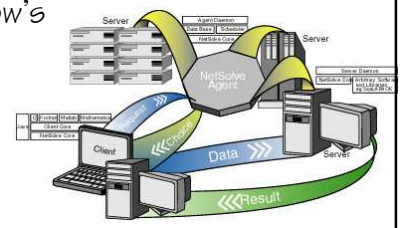
66

QR Factorization on 64 processors Intel Paragon



Motivation for NetSolve

Design an easy-to-use tool to provide efficient and uniform access to a variety of scientific packages on UNIX and Window's platforms

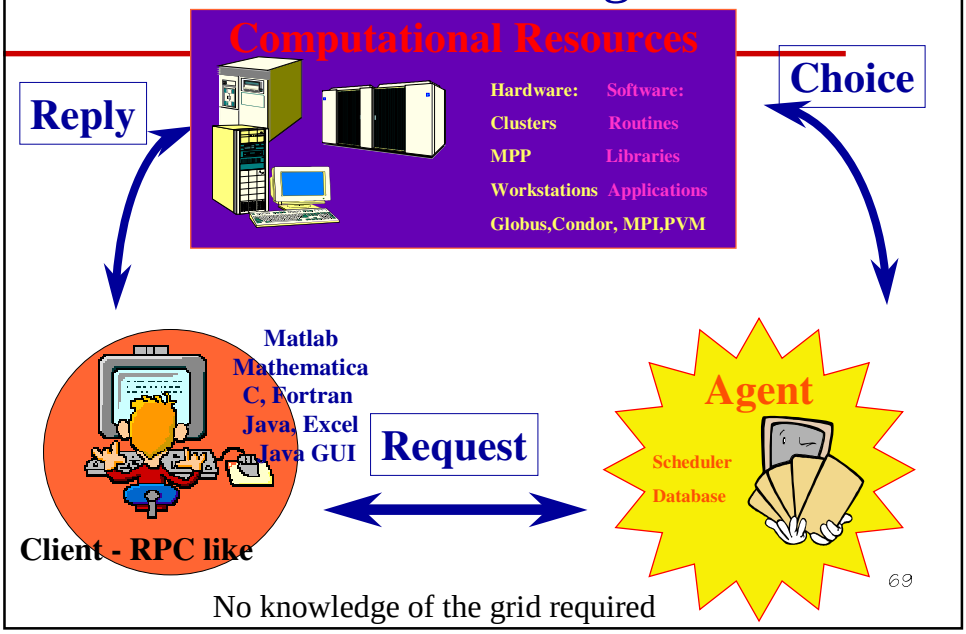


Basics

- Client-Server Design
- Non-hierarchical system
- Load Balancing and Fault Tolerance
- Heterogeneous Environment Supported
- Multiple and simple client interfaces
- Built on standard components

68

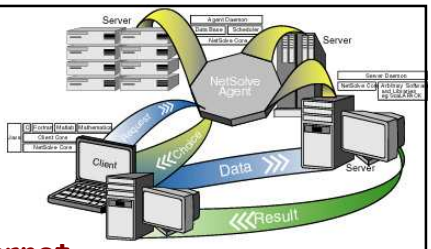
NetSolve - The Big Picture



No knowledge of the grid required

69

NetSolve



- Three basic scenarios:
  - Client, servers and agents anywhere on Internet (3(10)-150(80-ws/mpp)- Mcell)
  - Client, servers and agents on an Intranet
  - Client, server and agent on the same machine
- "Blue Collar" Grid Based Computing
  - User can set things up, no "su" required
  - Does not require deep knowledge of network programming
- Focus on Matlab users
  - OO language, objects are matrices (pse, eg os)
  - One of the most popular desktop systems for numerical computing, 400K Users

70

NetSolve - The Client

- No knowledge of networking involved
- Hide complexity of numerical software
- Computation **location transparency**
- Provides access to **Virtual Libraries** **:**
  - **Component grid-based framework**
  - **Central management of library resources**
  - **User not concerned with most up-to-date version**
  - **Automatic tie to Netlib repository in project**
- Provides synchronous or asynchronous calls
  (User level parallelism)

71

NetSolve - FORTRAN Interface

Easy to 'switch' to NetSolve

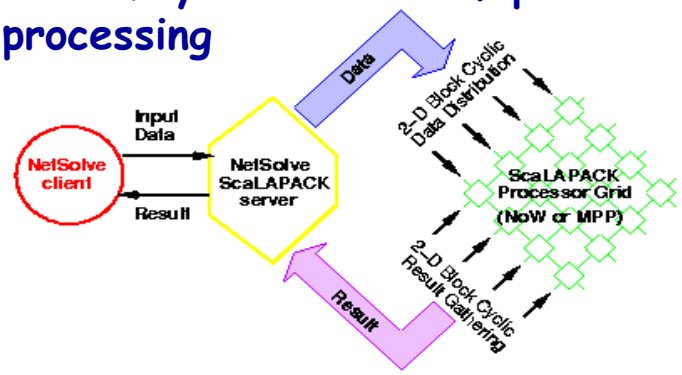
```
parameter( MAX = 100)
double precision A(MAX,MAX), B(MAX)
integer IPIV(MAX), N, INFO, LWORK
integer NSINFO

call DGESV(N,1,A,MAX,IPIV,B,MAX,INFO)


call NETSL('DGESV()',NSINFO,
           N,1,A,MAX,IPIV,B,MAX,INFO)
```

Hiding the Parallel Processing

- User maybe unaware of parallel processing



- NetSolve takes care of the starting the message passing system, data distribution, and returning the results.

73

NetSolve - MATLAB Interface

Synchronous Call

```
>> define sparse matrix A
>> define rhs
>> [x, its] = netsolve('itmeth','petsc', A, rhs, 1.e-6, 50);
...
>> [x, its] = petsc(A, rhs);   % for PETSc
>> [x, its] = aztec(A, rhs);   % for AZTEC
>> [x] = superlu(A, rhs);      % for SuperLU
>> [x] = ma28(A, rhs);         % for MA28
```

Asynchronous Calls also available

NetSolve - The Server Side

Computational Server :

- Various **Software** resources installed on various **Hardware** Resources
- **Configurable and Extensible :**
  - Framework to easily add arbitrary software ...
  - Many numerical libraries being integrated by the NetSolve team
  - Many software being integrated by users

Agent :

- Gateway to the computational servers
- Performs **Load Balancing** among the resources

http://www.cs.utk.edu/netsolve

NetSolve - Load Balancing

NetSolve agent :

predicts the execution times and sorts the servers

Prediction for a server based on :

- **Its distance over the network**
  - **Latency and Bandwidth**
  - **Statistical Averaging**
- **Its performance (LINPACK benchmark)**
- **Its workload**
- **The problem size and the algorithm complexity**

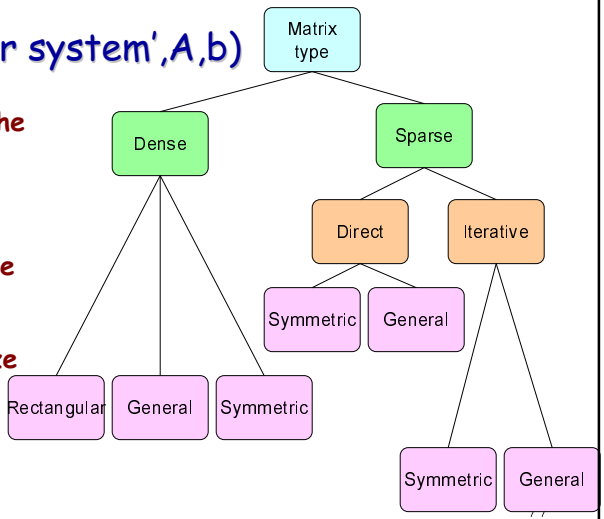
Cached data → **Quick estimate**

workload out of date ?

http://www.cs.utk.edu/netsolve

NetSolve to Determine the Best Algorithm/Software (A bit in the future)

- x = netsolve('linear system',A,b)
  - NetSolve examines the user's data together with the resources available (in the grid sense) and makes decisions on best time to solution dynamically.
  - Decision based on size of problem, hardware/software, network connection etc.
  - Method and Placement.



References

- http://www.netlib.org
- http://www.netlib.org/lapack
- http://www.netlib.org/scalapack
- http://www.netlib.org/lapack/lawns
- http://www.netlib.org/atlas
- http://www.netlib.org/papi/
- http://www.netlib.org/netsolve/
- http://www.netlib.org/lapack90
- http://www.nhse.org

- lapack@cs.utk.edu
- scalapack@cs.utk.edu

78