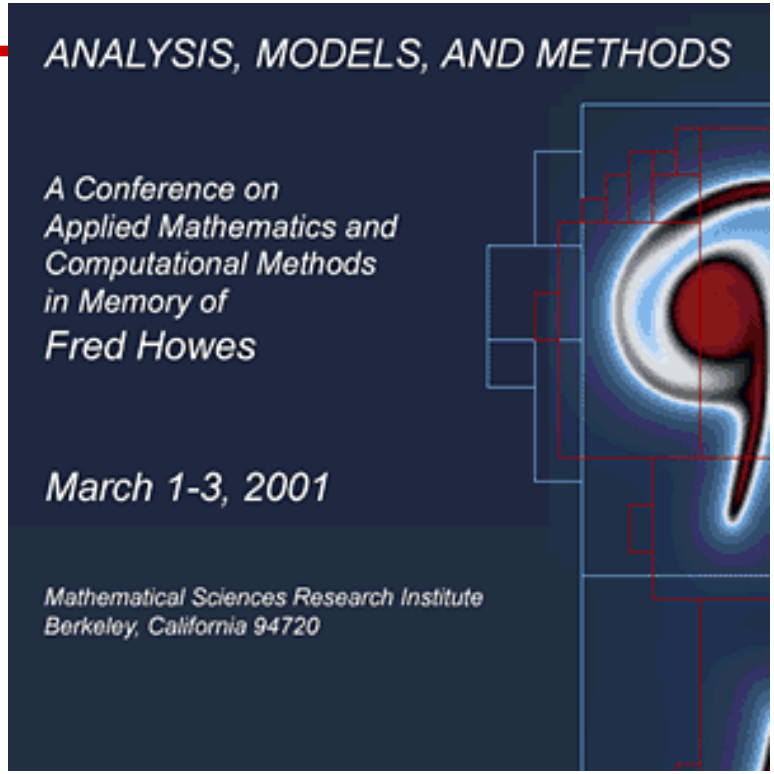
The Impact Of Computer Architectures On Linear Algebra and Numerical Libraries

Jack Dongarra
Innovative Computing Laboratory
University of Tennessee



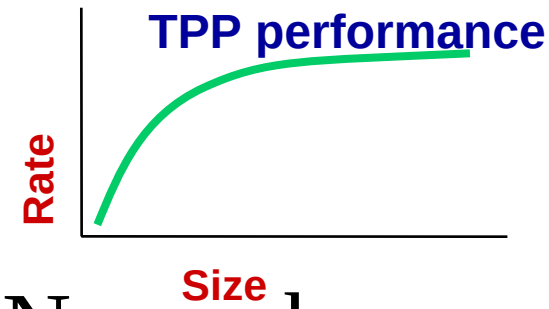
High Performance Computers

- ~ 20 years ago
  - $1x10^6$ Floating Point Ops/sec (Mflop/s)
    - Scalar based
- ~ 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

TOP500

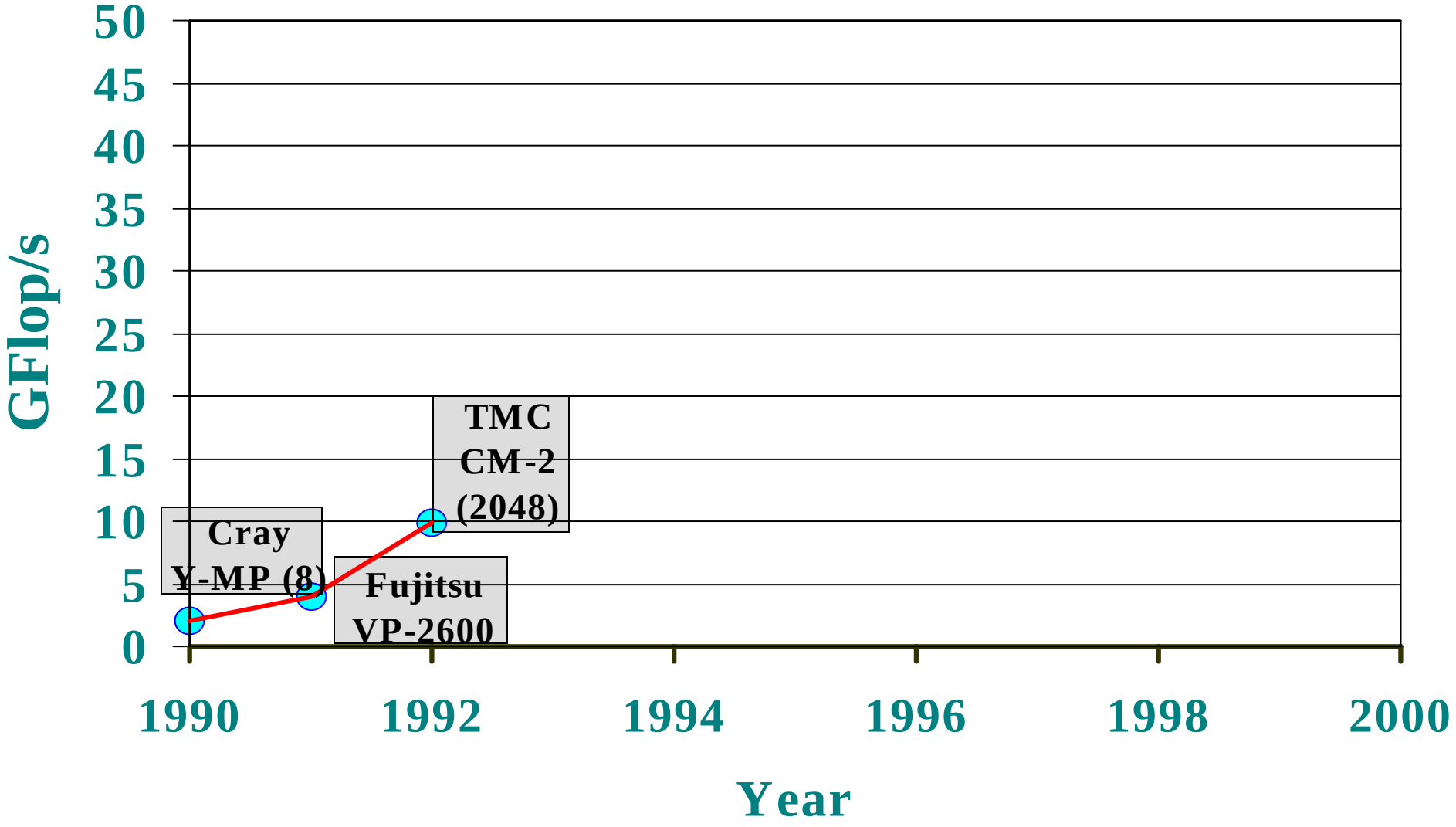
- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

  $Ax=b$, *dense problem*



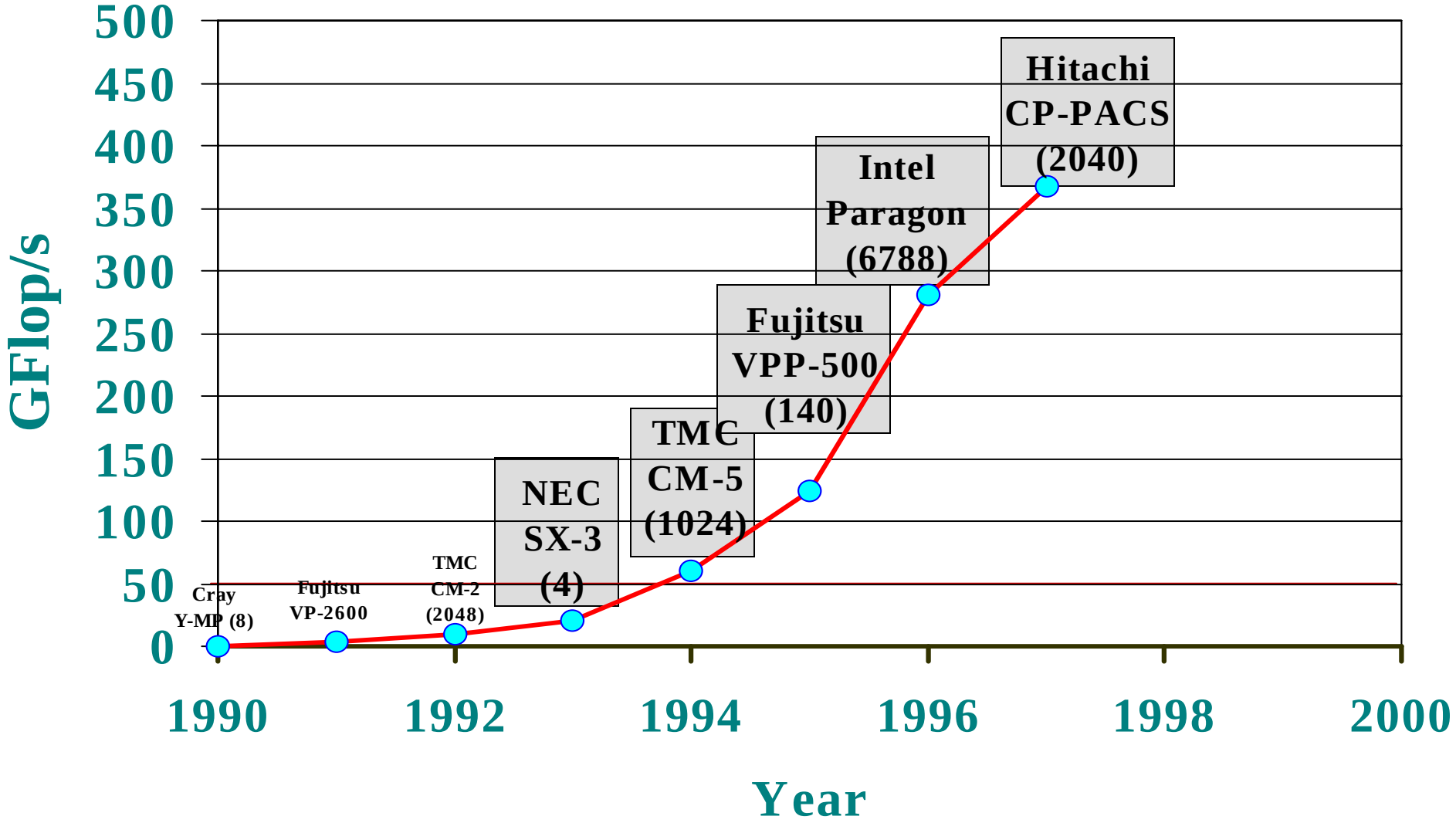
- Updated twice a year
  SC‘xy in the States in November
  Meeting in Mannheim, Germany in June
- All data available from **www.top500.org**

Fastest Computer Over Time



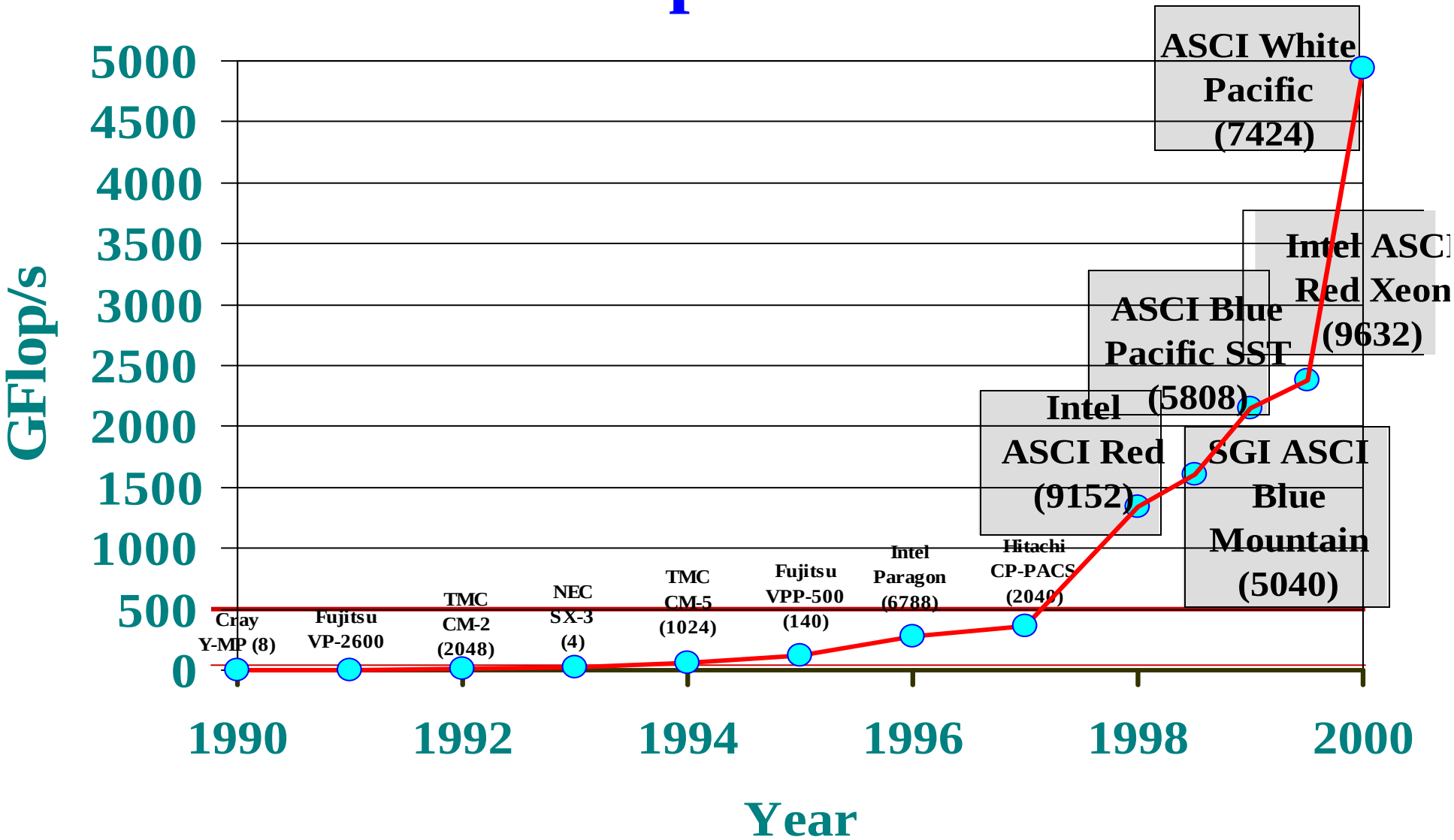
In 1980 a computation that took 1 full year to complete can now be done in ~ 9 hours!

Fastest Computer Over Time



In 1980 a computation that took 1 full year to complete can now be done in ~ 13 minutes!

Fastest Computer Over Time

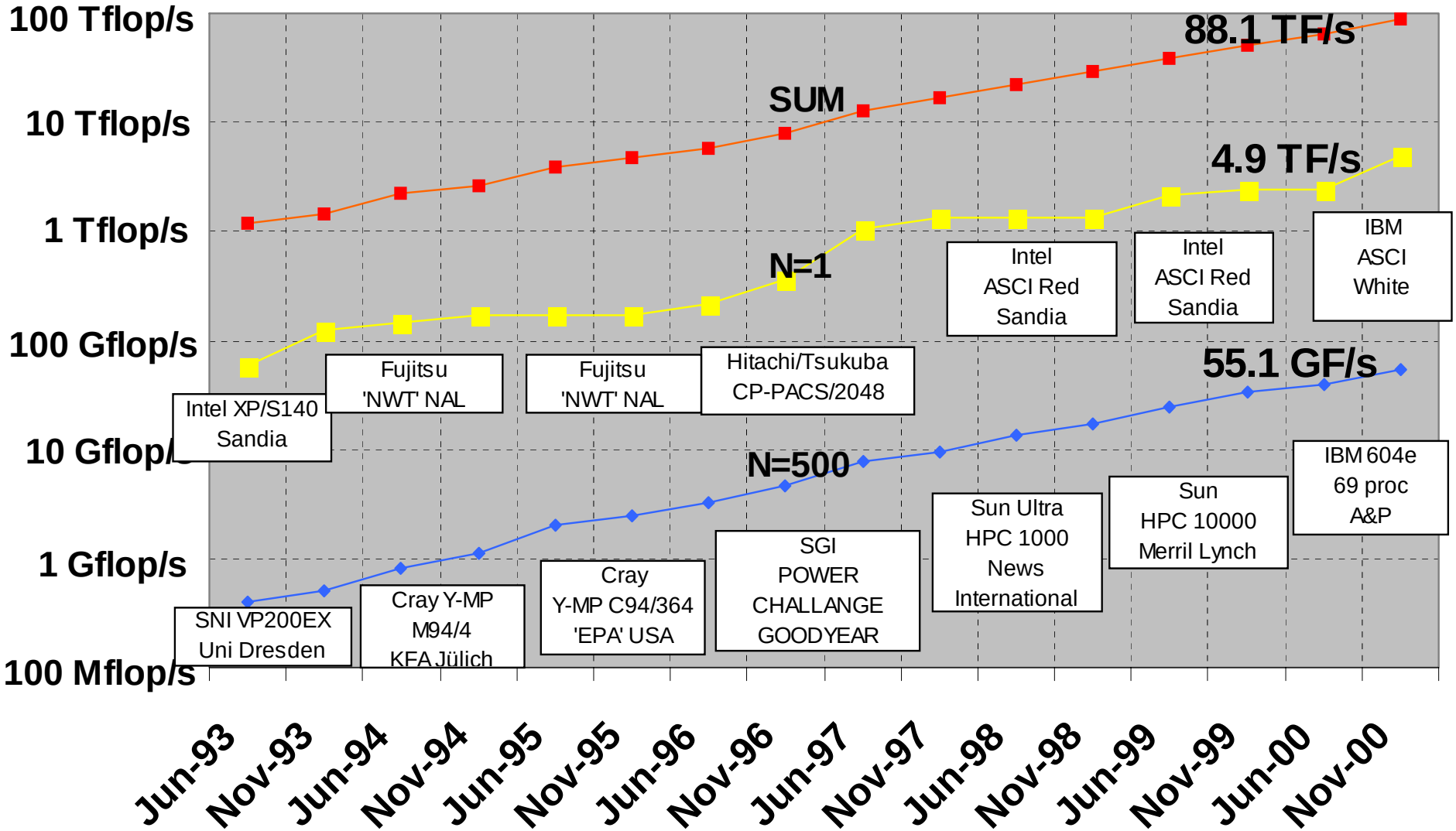


In 1980 a computation that took 1 full year to complete can today be done in ~90 second!

Top 10 Machines (Nov 2000)

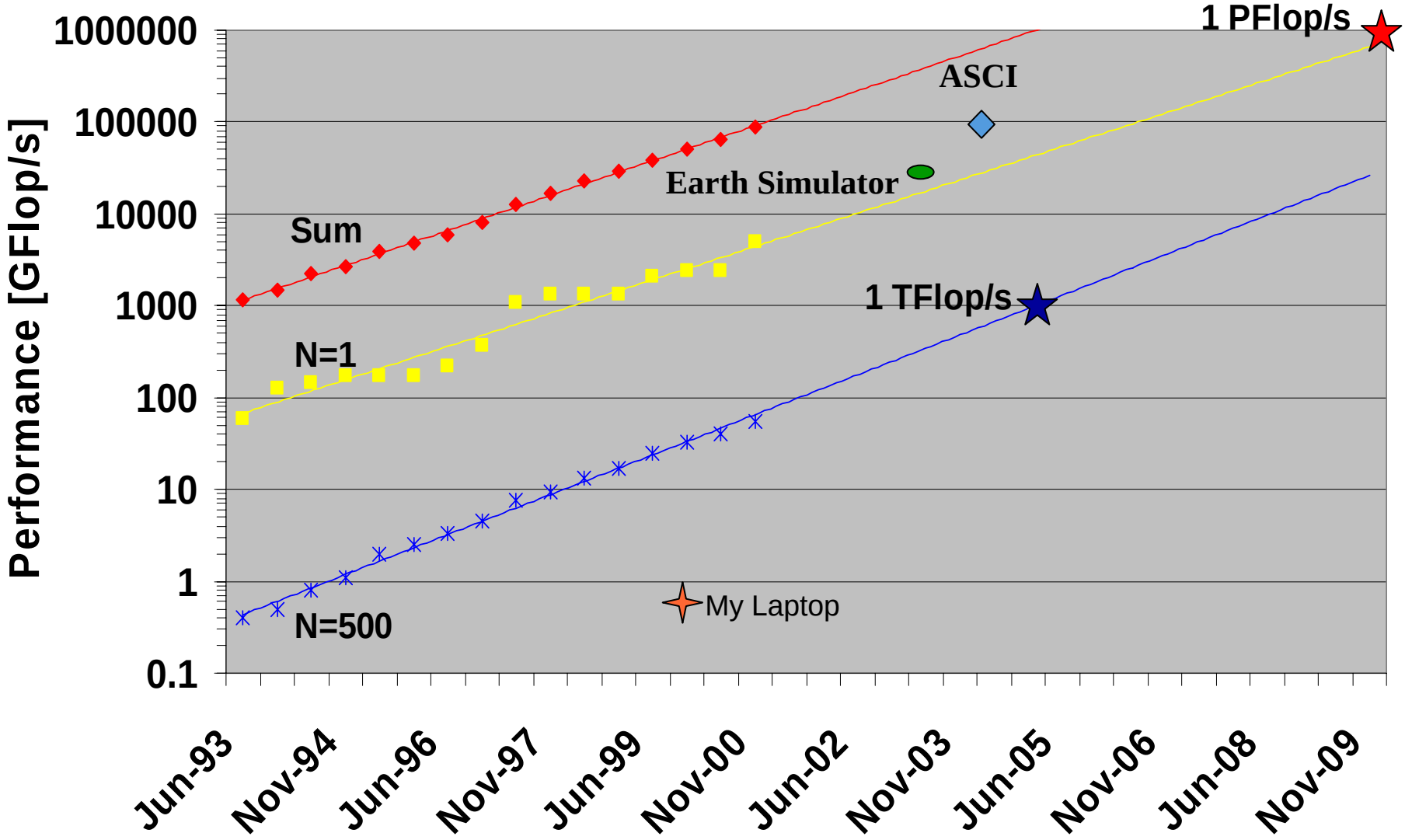
Rank	Company	Machine	Procs	Gflop/s	Place	Country	Year
1	IBM	ASCI White	8192	4938	Livermore National Laboratory Livermore		2000
2	Intel	ASCI Red	9632	2380	Sandia National Labs Albuquerque	USA	1999
3	IBM	ASCI Blue-Pacific SST, IBM SP 604e	5808	2144	Lawrence Livermore National Laboratory Livermore	USA	1999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

Performance Development



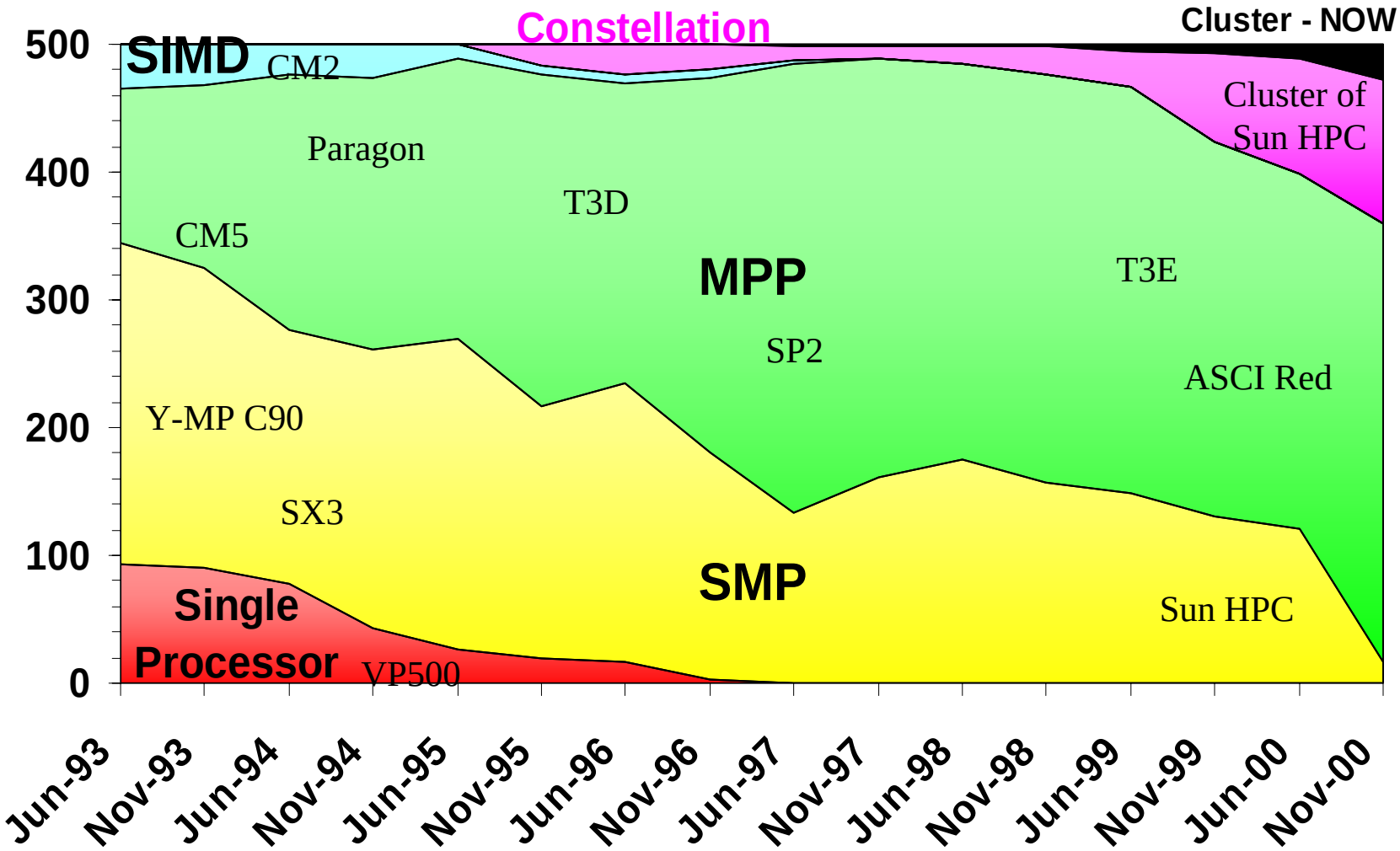
[60G - 400 M][4.9 Tflop/s 55Gflop/s], Schwab #15, 1/2 each year, 209 > 100 Gf, faster than Moore's law,

Performance Development



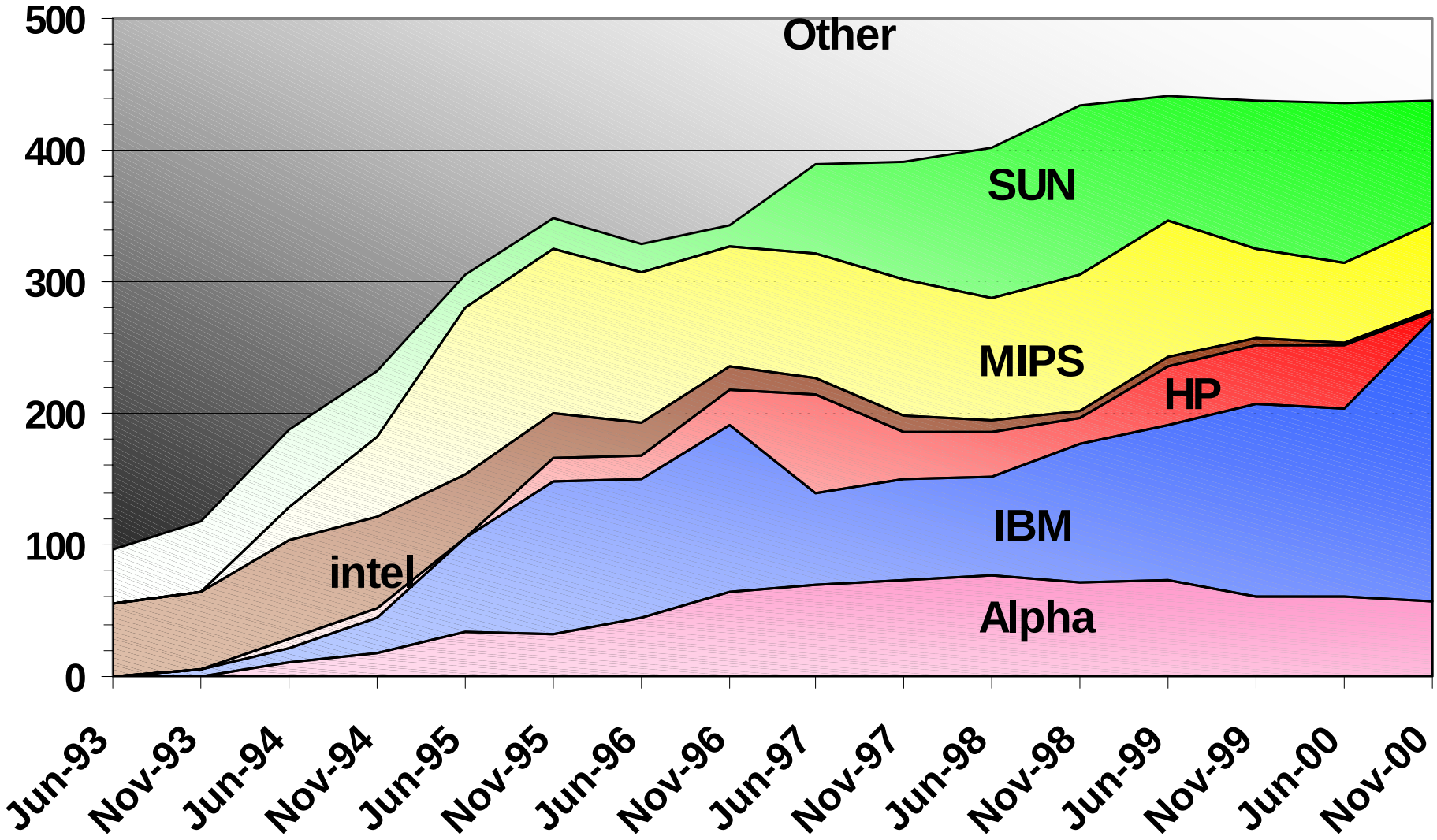
Entry 1 T 2005 and 1 P 2010

Architectures

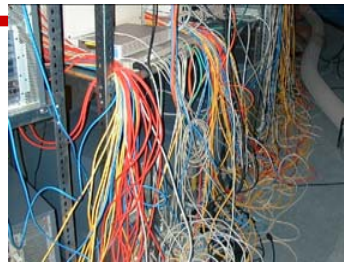


112 const, 28 clus, 343 mpp, 17 smp

Chip Technology



High-Performance Computing Directions: Beowulf-class PC Clusters



Definition:

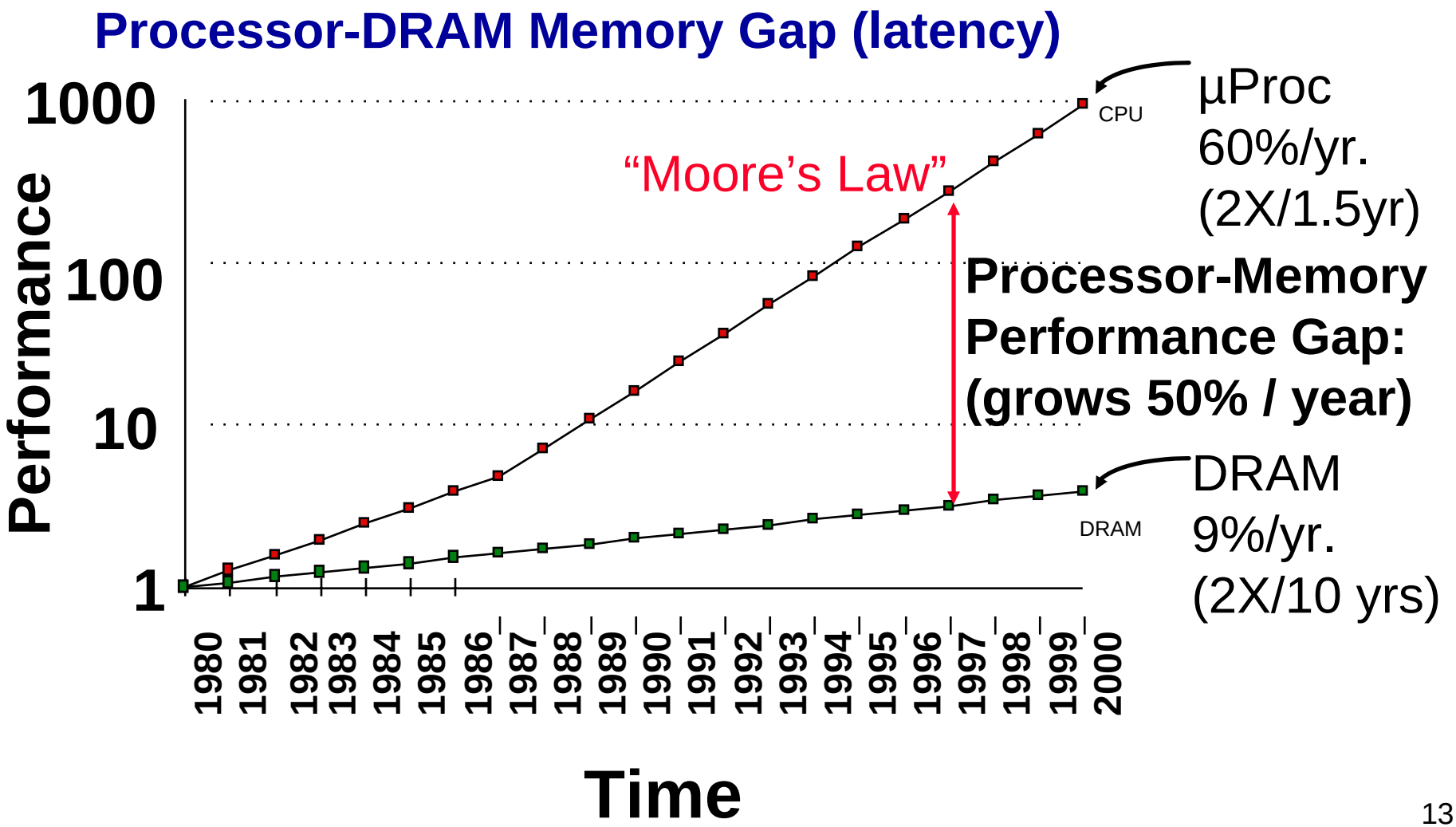
- **COTS PC Nodes**
  - **Pentium, Alpha, PowerPC, SMP**
- **COTS LAN/SAN Interconnect**
  - **Ethernet, Myrinet, Giganet, ATM**
- **Open Source Unix**
  - **Linux, BSD**
- **Message Passing Computing**
  - **MPI, PVM**
  - **HPF**

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?

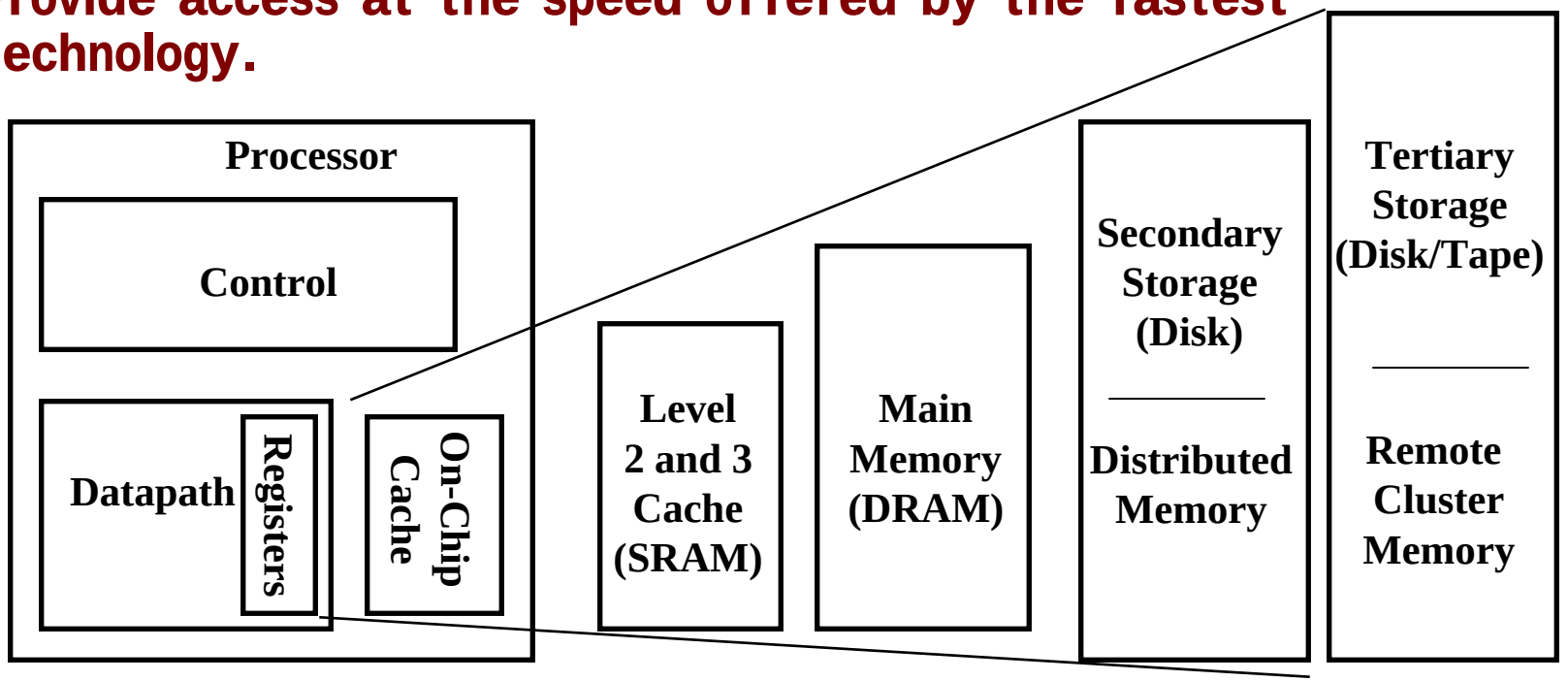


Optimizing Computation and Memory Use

- **Computational optimizations**
  - **Theoretical peak:(# fpus)*(flops/cycle) * Mhz**
    - **PIII: (1 fpu)*(1 flop/cycle)*(850 Mhz) = 850 MFLOP/s**
    - **Athlon: (2 fpu)*(1flop/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.**

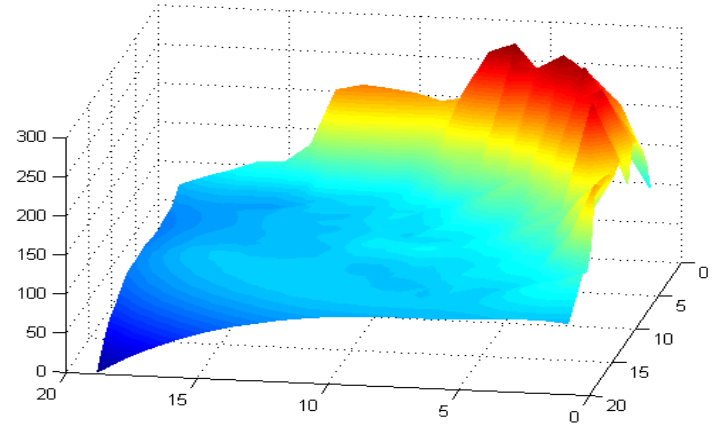


Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	100,000 s (.1s ms) Gs	10,000,000 s (10s ms) Ts

Self-Adapting Numerical Software (SANS)

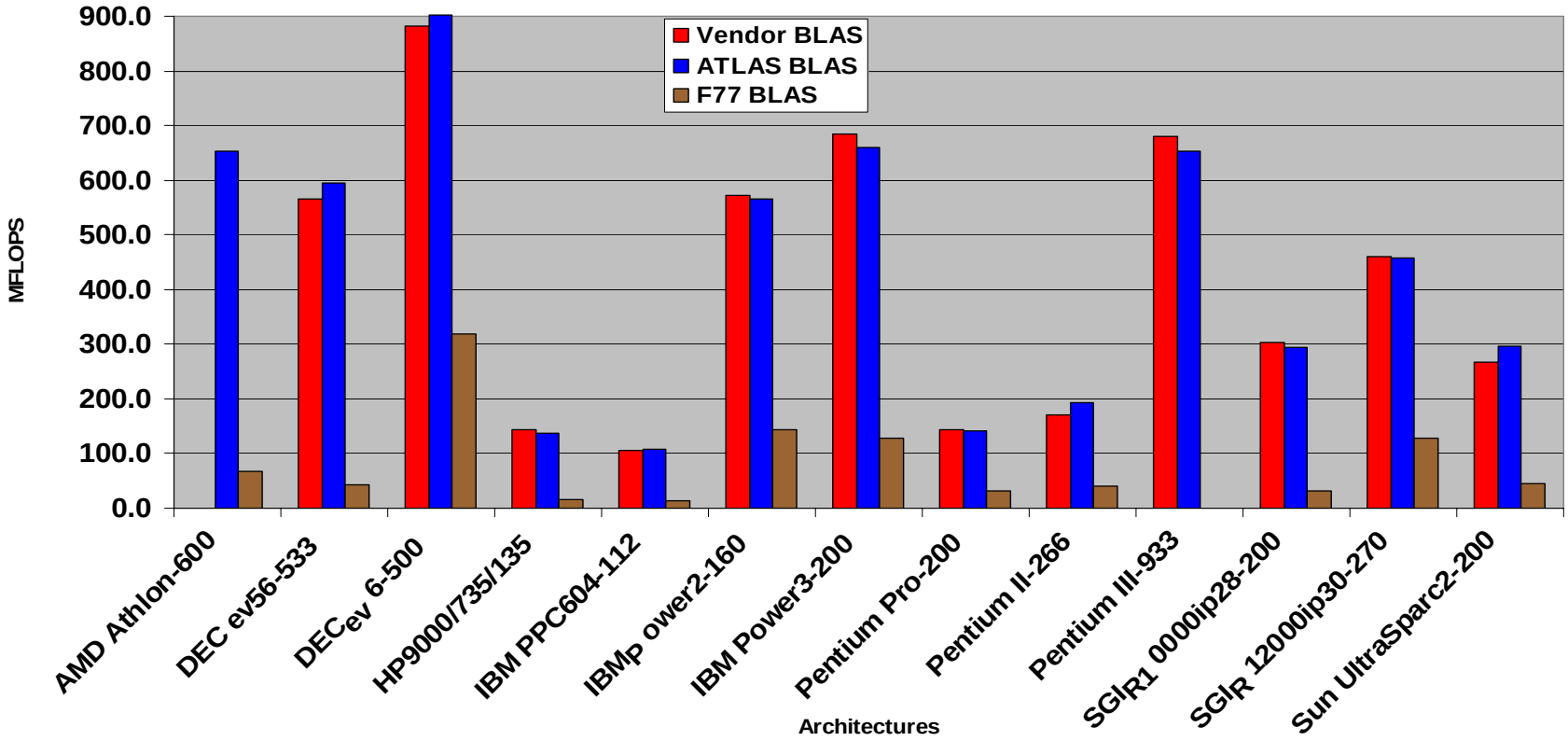
- **Today's processors can achieve high-performance, but this requires extensive machine-specific hand tuning.**
- **Operations like the BLAS require many man-hours / platform**
  - **Software lags far behind hardware introduction**
  - **Only done if financial incentive is there**
- **Hardware, compilers, 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



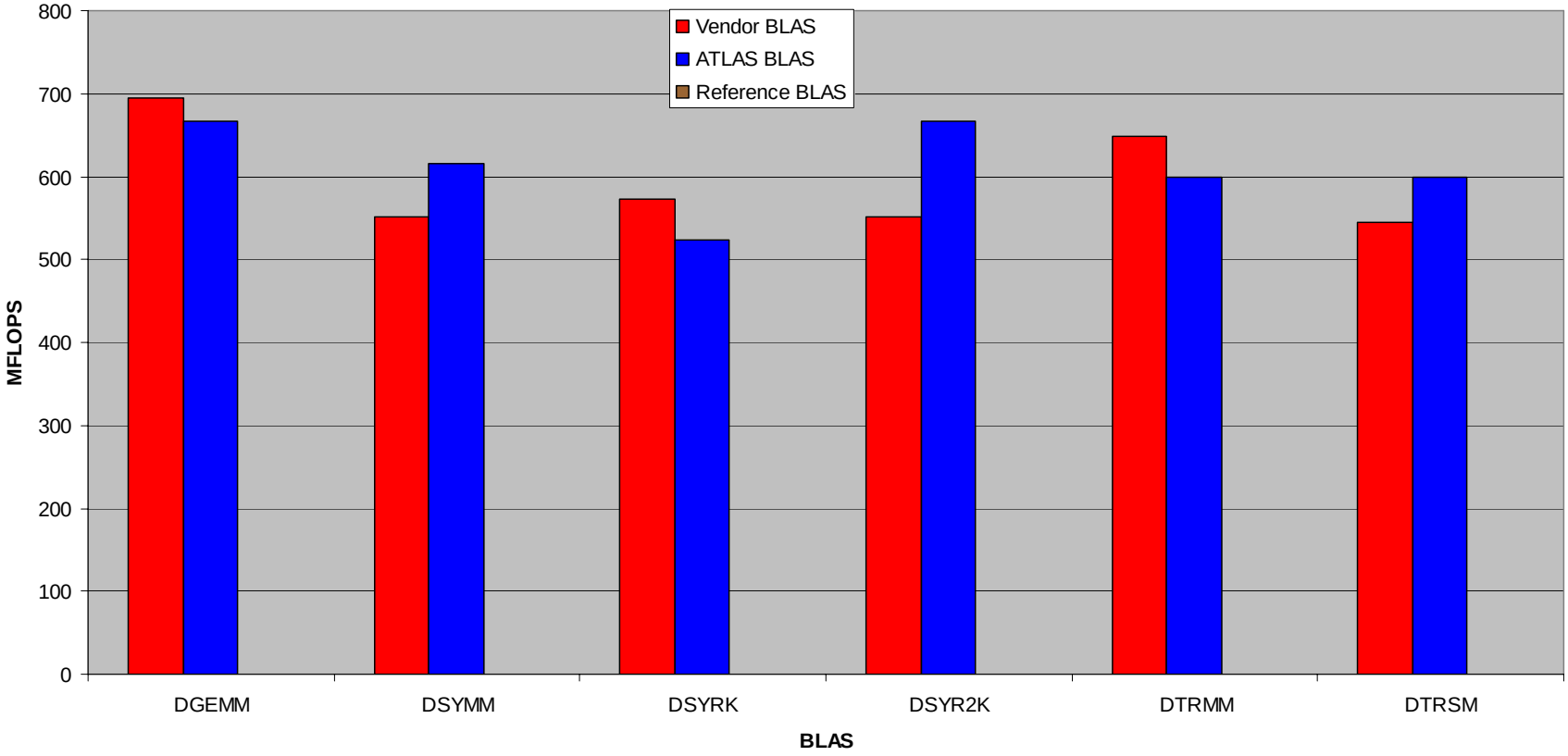
- Level 1 cache multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization
- Takes about 30 minutes to run.
- "New" model of high performance programming where critical code is machine generated using parameter optimization.
- 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 (DGEMM n = 500)



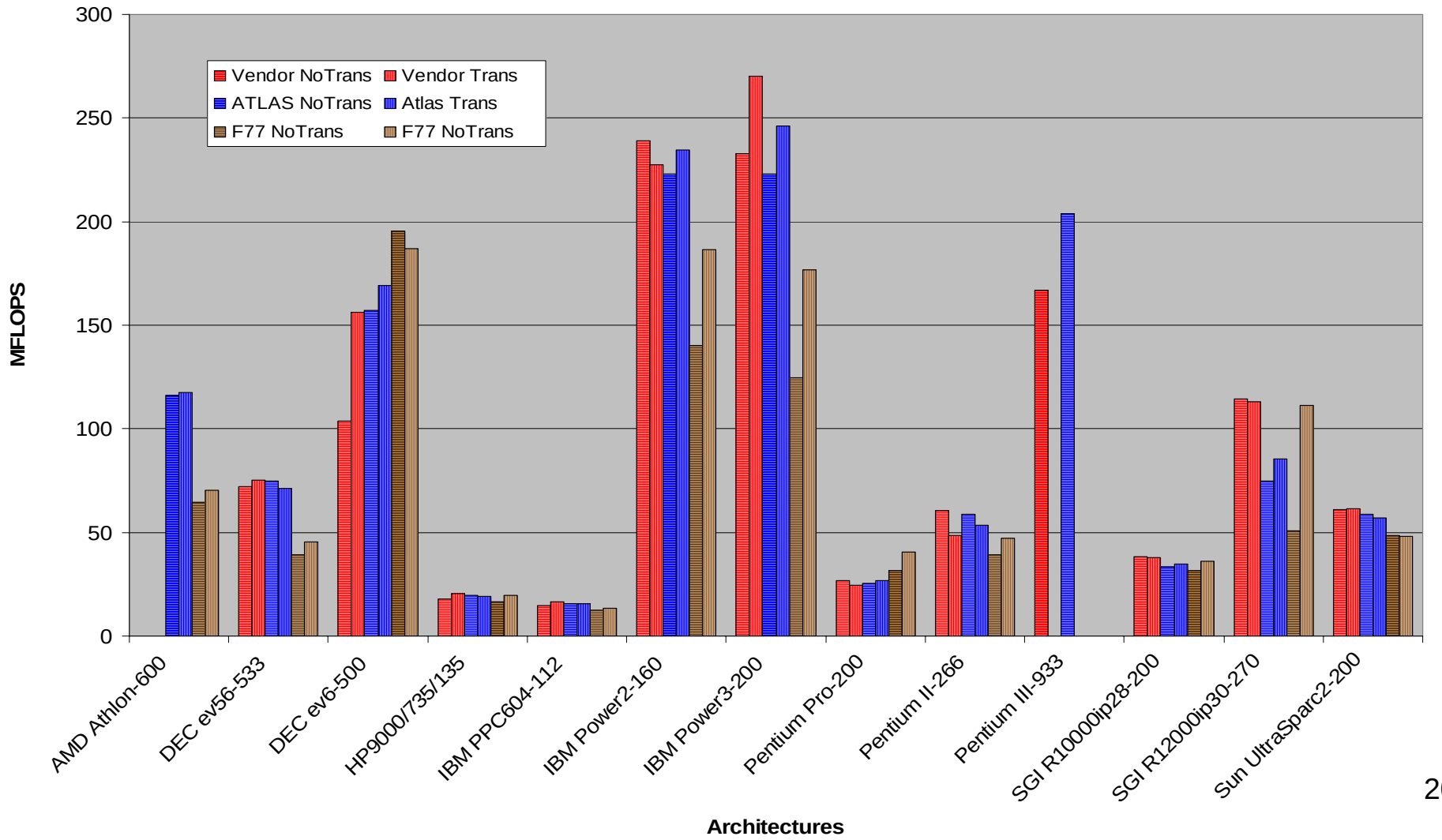
- **ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.**

Intel PIII 933 MHz
MKL 5.0 vs ATLAS 3.2.0 using Windows 2000

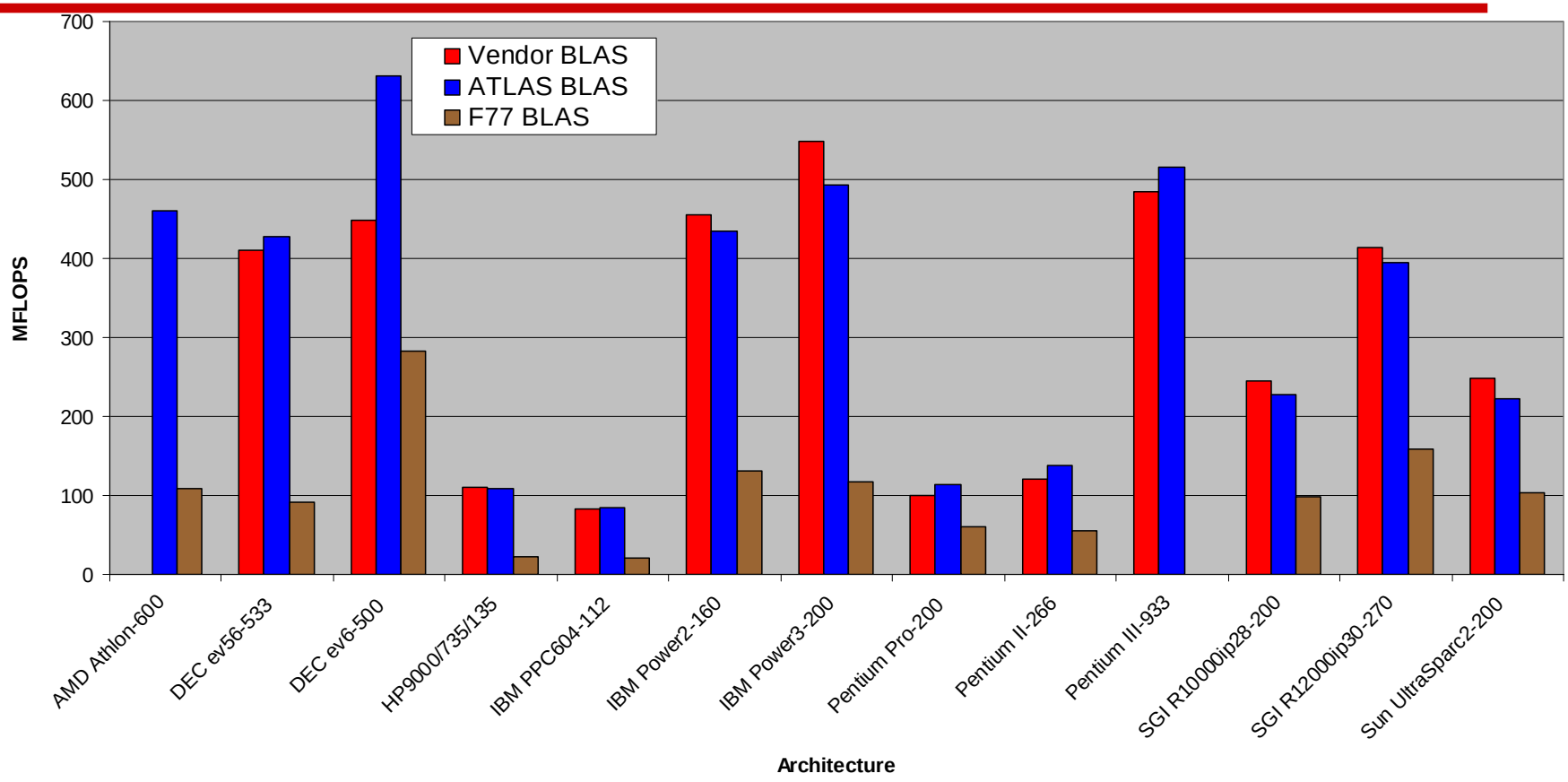


- **ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.**

Matrix Vector Multiply DGEMV



LU Factorization, Recursive w/ATLAS

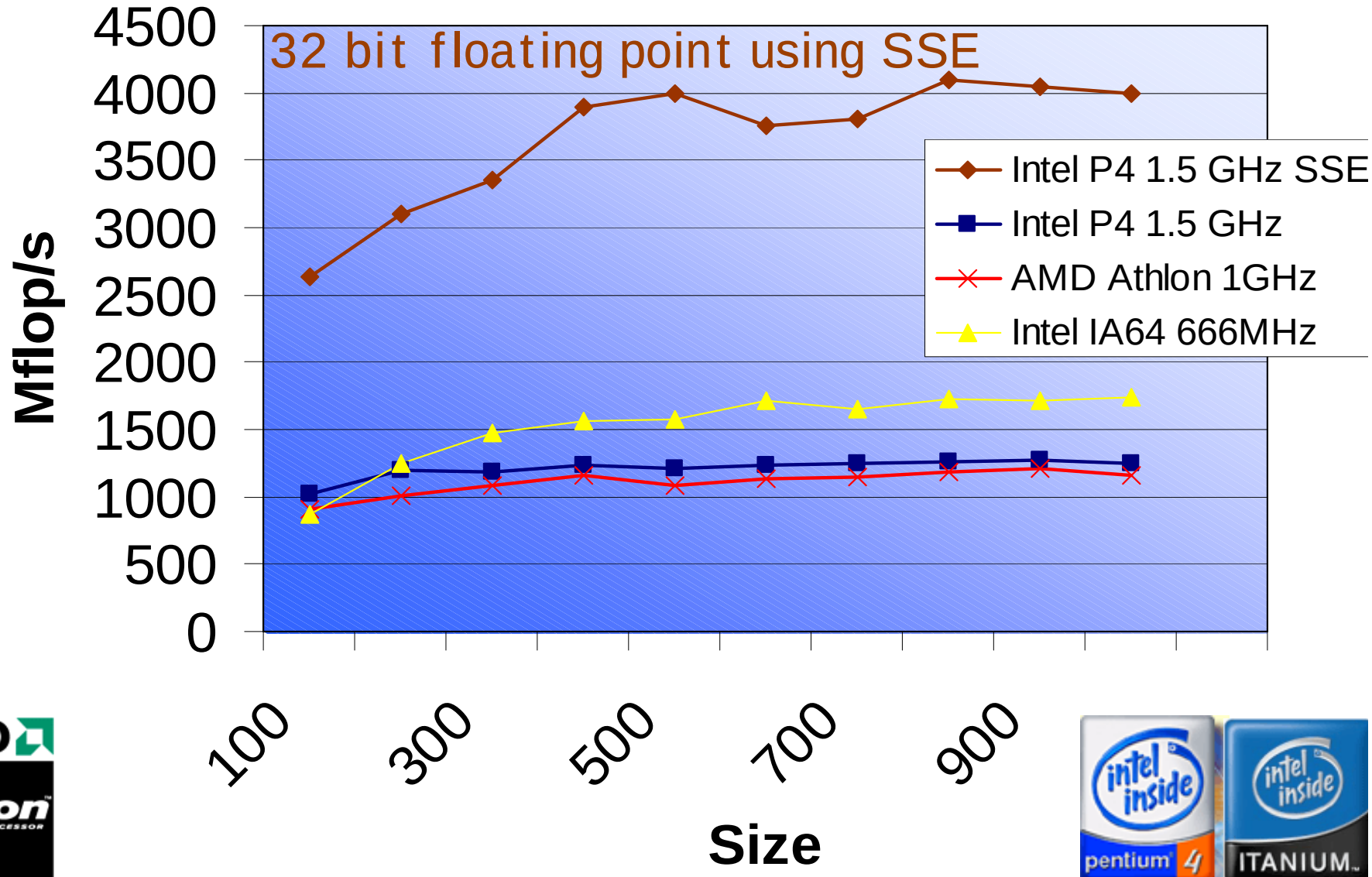


- **ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.**

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)

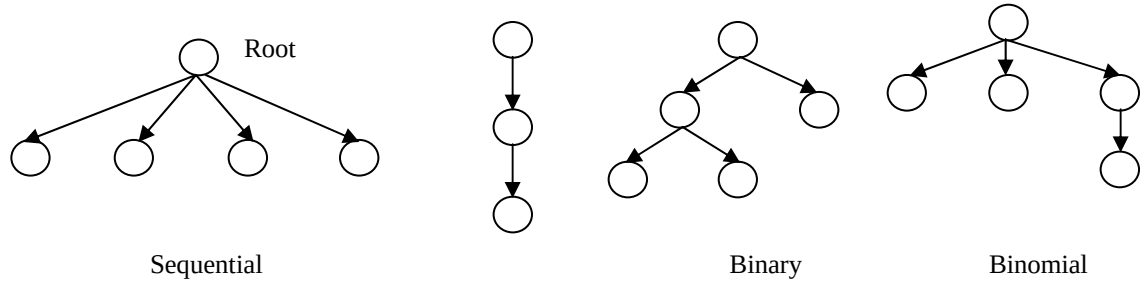


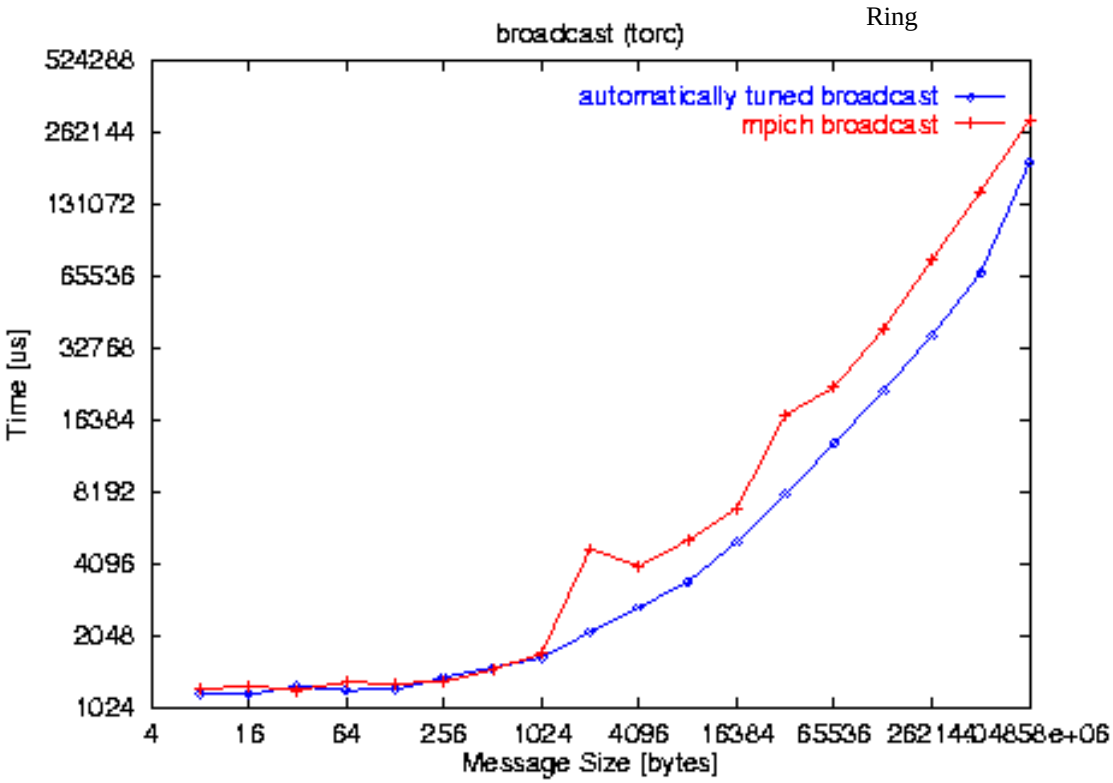
Machine-Assisted Application Development and Adaptation

- **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)





Message Size (bytes)	Optimal algorithm	Buffer Size (bytes)
8	binomial	8
16	binomial	16
32	binary	32
64	binomial	64
128	binomial	128
256	binomial	256
512	binomial	512
1K	sequential	1K
2K	binary	2K
4K	binary	2K
8K	binary	2K
16K	binary	4K
32K	binary	4K
64K	ring	4K
128K	ring	4K
256K	ring	4K
512K	ring	4K
1M	binary	4K

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.**

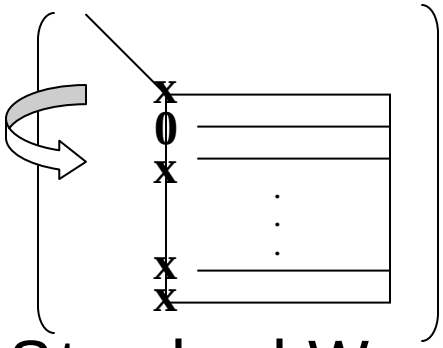
Classical
Norm calculation:
$\boxed{error = \sqrt{r^t r}}$
Preconditioner application:
$z \leftarrow M^{-1} r$
Matrix-vector product:
Inner products 1:
$\boxed{\rho \leftarrow z^t r}$
$\beta \leftarrow \rho / \rho_{\text{old}}$
Search direction update:
$p \leftarrow z + \beta p$
Matrix-vector product:
$ap \leftarrow A \times p$
Preconditioner application:
Inner products 2:
$\boxed{\pi \leftarrow p^t ap}$
$\alpha = \rho / \pi$
Residual update:
$r \leftarrow r - \alpha Ap$
3 separate inner products

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.**

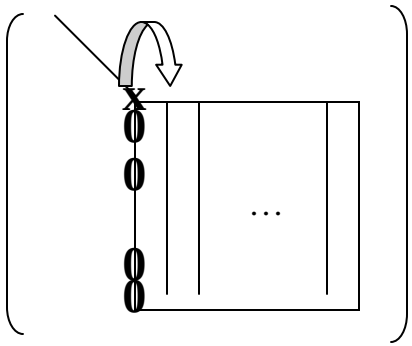
Classical	Saad/Meurant	Chronopoulos/Gear	Eijkhout
Norm calculation:			
$\boxed{error = \sqrt{r^t r}}$			
Preconditioner application:			
$z \leftarrow M^{-1} r$	$z \leftarrow z - \alpha q$	$z \leftarrow M^{-1} r$	id
Matrix-vector product:			
		$az \leftarrow A \times z$	id
Inner products 1:			
$\boxed{\rho \leftarrow z^t r}$	$\rho_{\text{predict}} \leftarrow -\rho_{\text{true}} + \alpha^2 \mu$	$\boxed{\begin{array}{c} error = \sqrt{r^t r} \\ \rho \leftarrow z^t r \\ \zeta \leftarrow z^t a z \end{array}}$	$\boxed{\begin{array}{c} error = \sqrt{r^t r} \\ \rho \leftarrow z^t r \\ \zeta \leftarrow z^t a z \\ \epsilon \leftarrow (M^{-1} r)^t (Ap) \end{array}}$
$\beta \leftarrow \rho / \rho_{\text{old}}$	$\beta = \rho_{\text{predict}} / \rho_{\text{old}}$	$\beta \leftarrow \rho / \rho_{\text{old}}$	id
Search direction update:			
$p \leftarrow z + \beta p$	id	id	id
Matrix-vector product:			
$ap \leftarrow A \times p$	id	$ap \leftarrow az + \beta ap$	id
Preconditioner application:			
	$q \leftarrow M^{-1} ap$		
Inner products 2:			
$\boxed{\pi \leftarrow p^t ap}$	$\boxed{\begin{array}{c} \pi \leftarrow p^t ap \\ \mu \leftarrow ap^t q \\ error = \sqrt{r^t r} \\ \rho_{\text{true}} = z^t r \end{array}}$	$\pi \leftarrow \zeta - \beta^2 \pi$	$\pi \leftarrow \zeta + \beta \epsilon$
$\alpha = \rho / \pi$	$\cdots \rho_{\text{true}} \cdots$	$\alpha = \rho / \pi$	
Residual update:			
$r \leftarrow r - \alpha Ap$	id	id	id
3 separate inner products	4 combined	3 combined	4 combined
	1 extra vector update	id	id

Gaussian Elimination



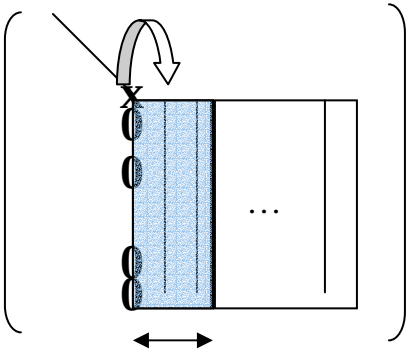
Standard Way

subtract a multiple of a row

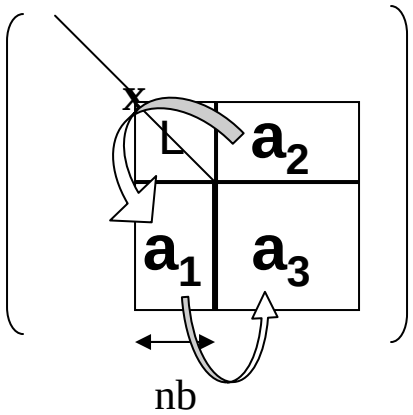


LINPACK

apply sequence to a column



LAPACK



$a_2 = L^{-1} a_2$

$a_3 = a_3 - a_1 * a_2$

apply sequence to nb

then apply nb to rest of matrix

Gaussian Elimination via a Recursive Algorithm

F. Gustavson and S. Toledo

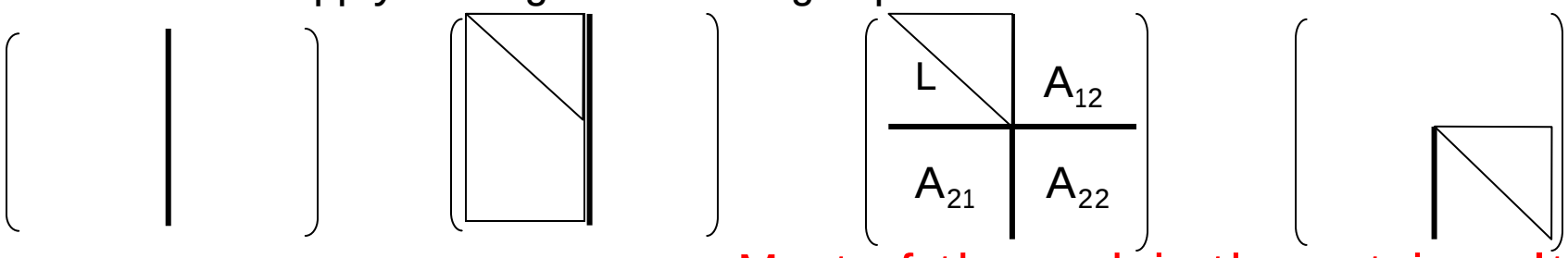
LU Algorithm:

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 $A_{22}=A_{22} - A_{21}*A_{12}$)

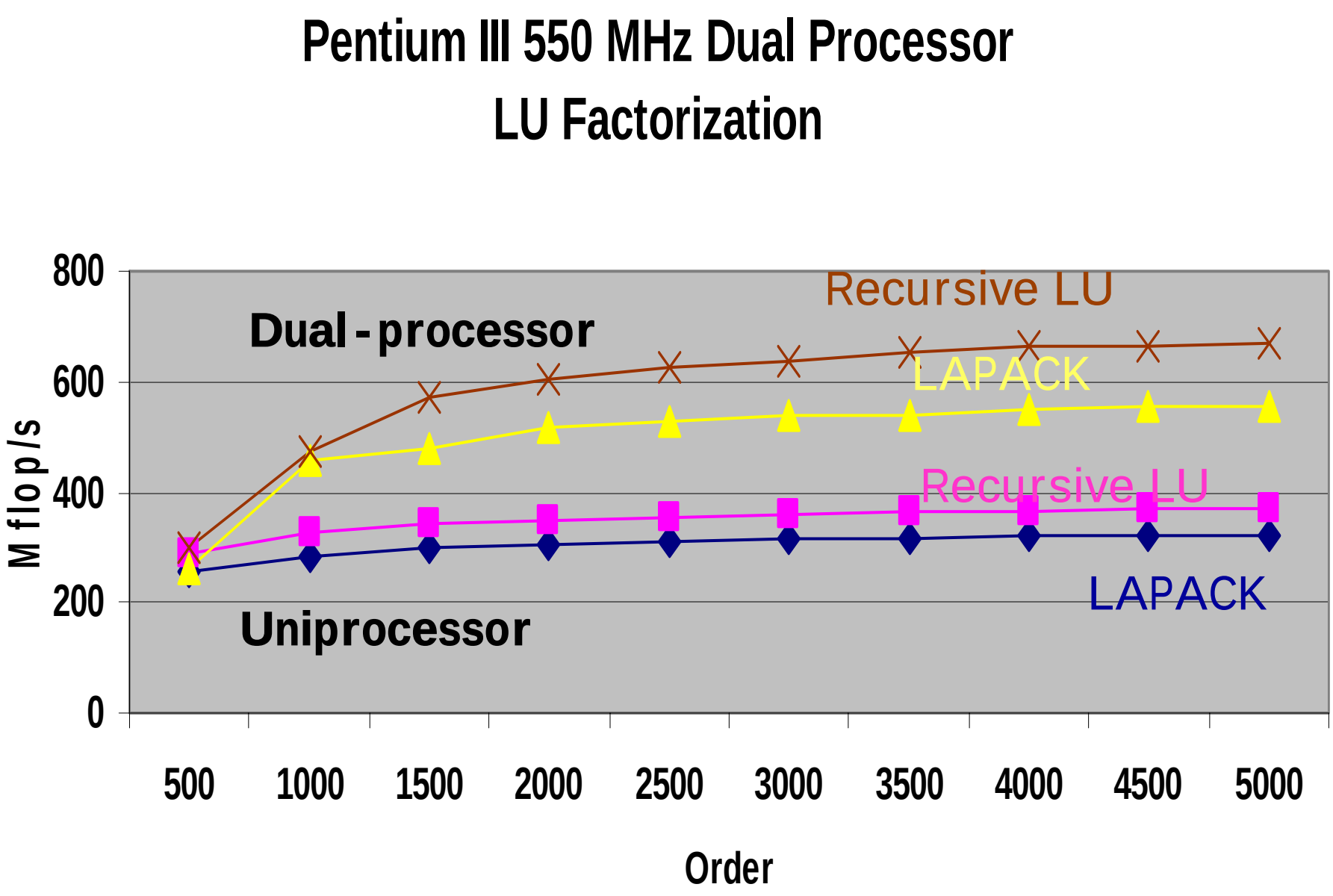
4: Apply LU Algorithm to right part

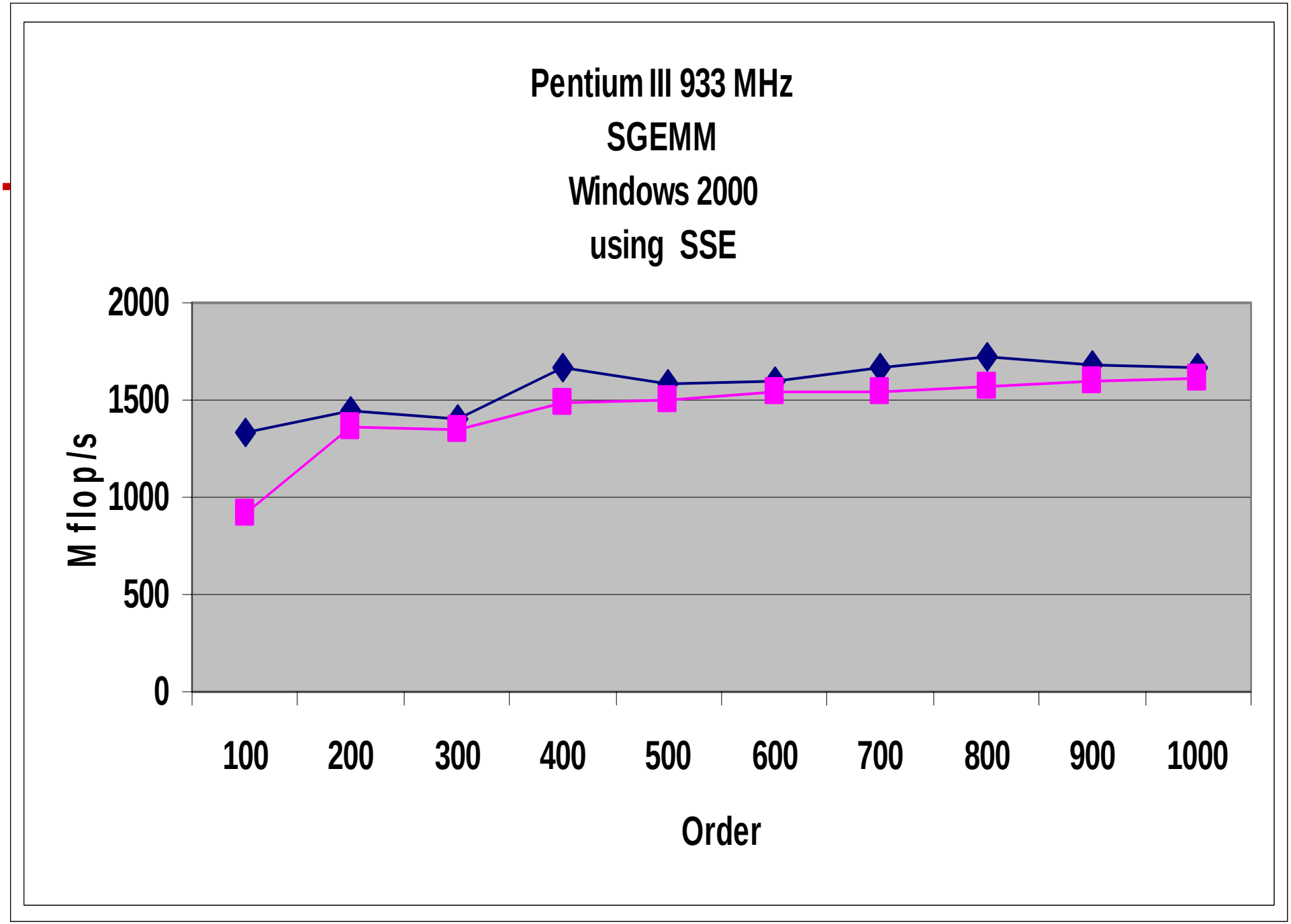


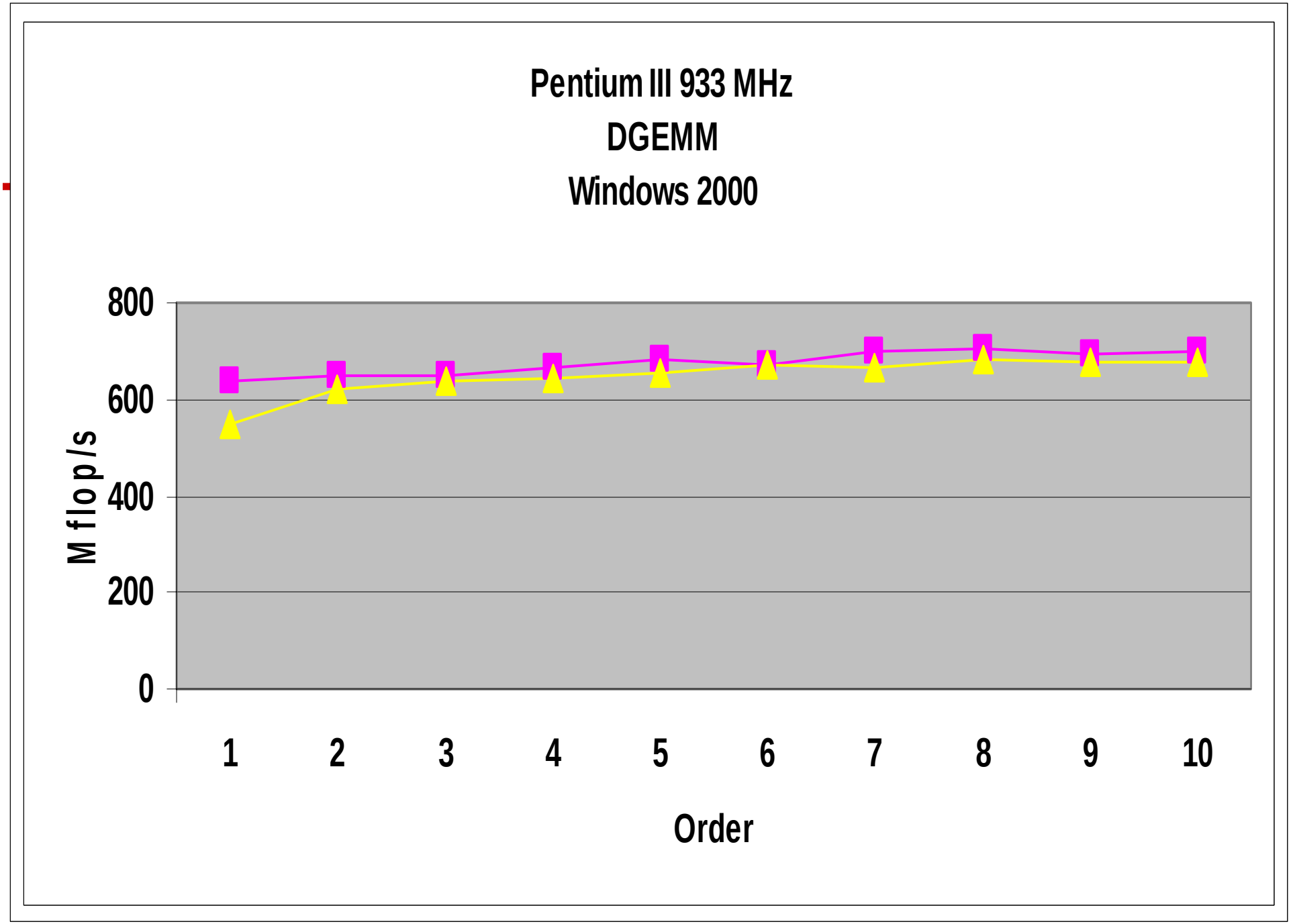
Most of the work in the matrix multiply
Matrices of size n/2, n/4, n/8, …

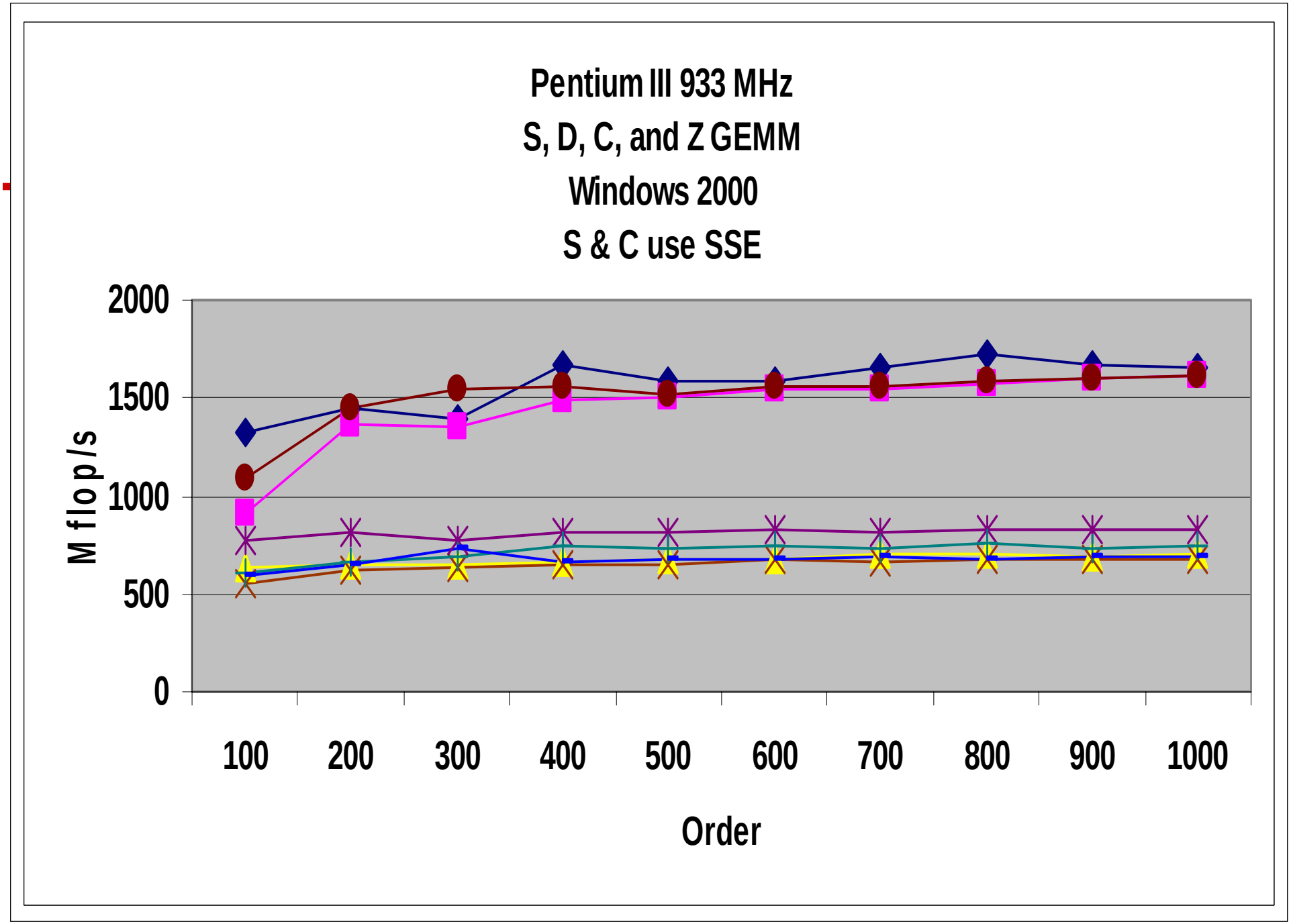
Recursive Factorizations

- 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
Uniprocessor
Recursive LU
LAPACK
Recursive LU
LAPACK
800
600
400
200
0
Mflop/s
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Order

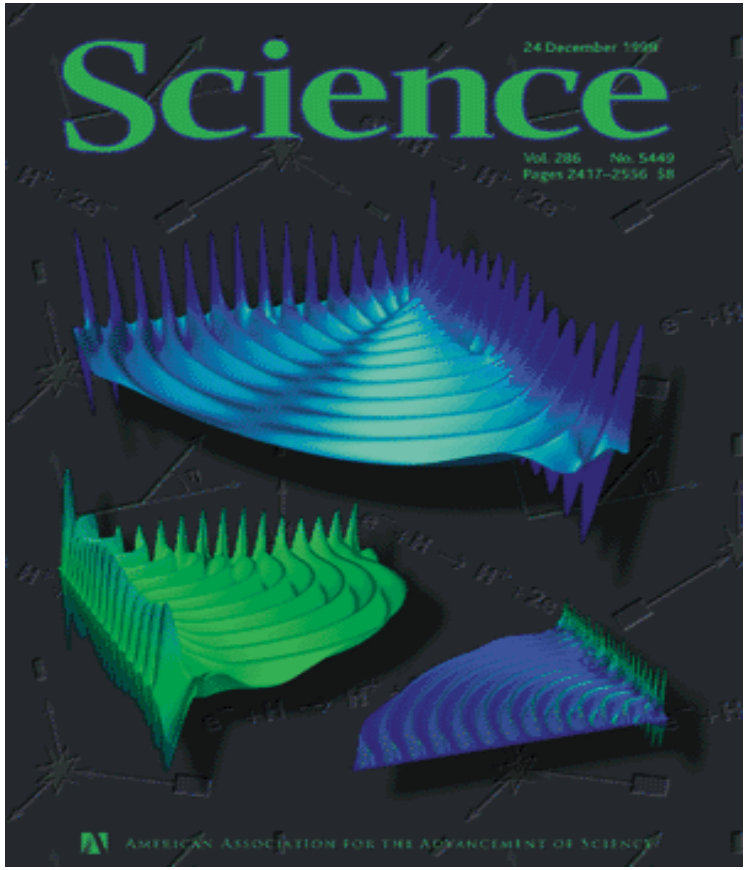

Pentium III 933 MHz
SGEMM
Windows 2000
using SSE
2000
1500
1000
500
0
M flop/s
100
200
300
400
500
600
700
800
900
1000
Order


Pentium III 933 MHz
DGEMM
Windows 2000
800
600
400
200
0
M flop/s
1
2
3
4
5
6
7
8
9
10
Order


Pentium III 933 MHz
S, D, C, and Z GEMM
Windows 2000
S & C use SSE
2000
1500
1000
500
0
M flop/s
100
200
300
400
500
600
700
800
900
1000
Order

SuperLU - High Performance Sparse Solvers

- SuperLU; X. Li and J. Demmel
  - Solve sparse linear system A 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
  - First solution to quantum scattering of 3 charged particles. [Recigno, Baertschy, Isaacs & McCurdy, Science, 24 Dec 1999]
  - SuperLU solved complex unsymmetric systems of order up to 1.79 million, on the ASCI Blue Pacific Computer at LLNL.

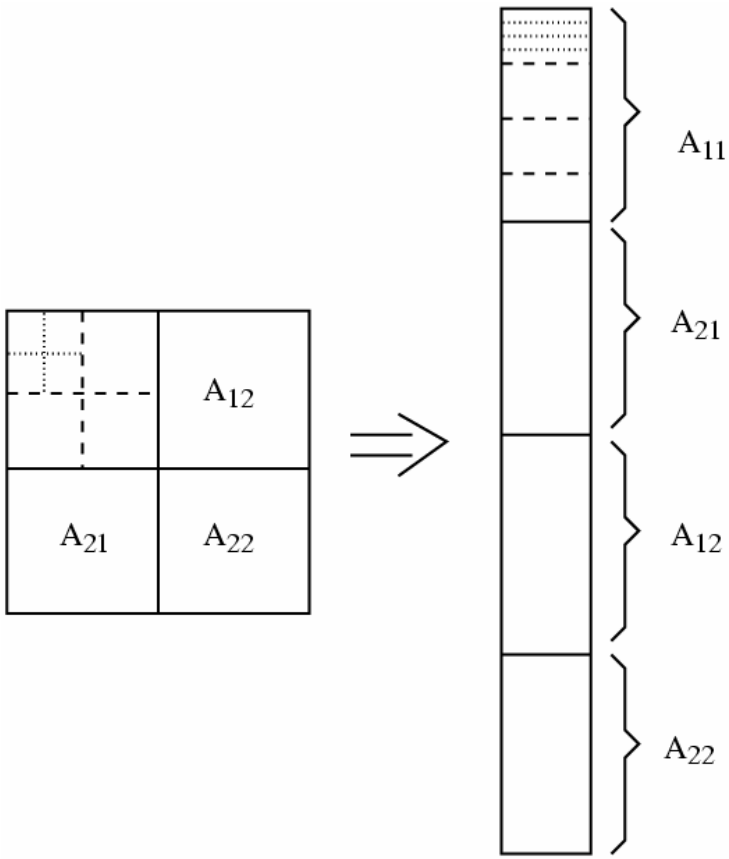


Recursive Factorization Applied to Sparse Direct Methods

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
$A_{21} \leftarrow A_{21} \cdot U^{-1}(A_{11})$;	xTRSM() on upper triangular submatrix
$A_{12} \leftarrow L_1^{-1}(A_{11}) \cdot A_{12}$;	xTRSM() on lower triangular submatrix
$A_{22} \leftarrow A_{22} - A_{21} \cdot A_{12}$;	xGEMM()
rlu(A_{22});	recursive call

end.

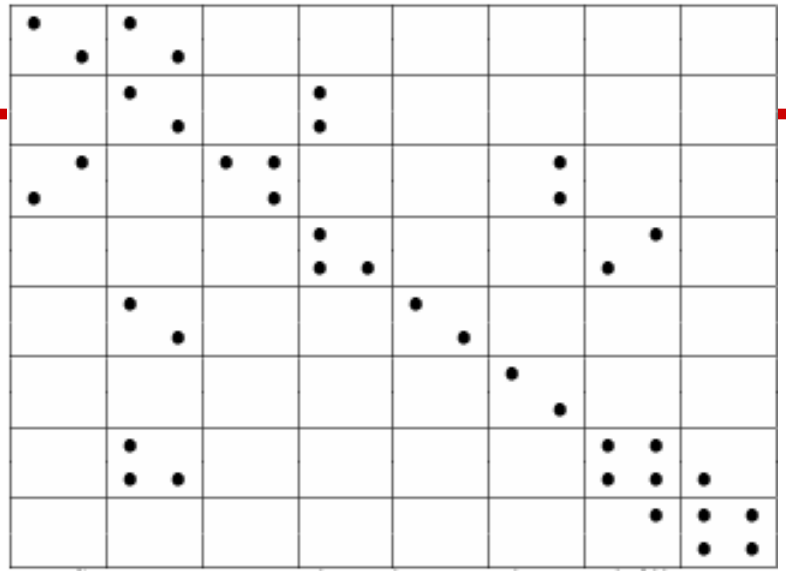
- **Replace xTRSM and xGEMM with sparse implementations that are themselves recursive**

Sparse Recursive Factorization Algorithm

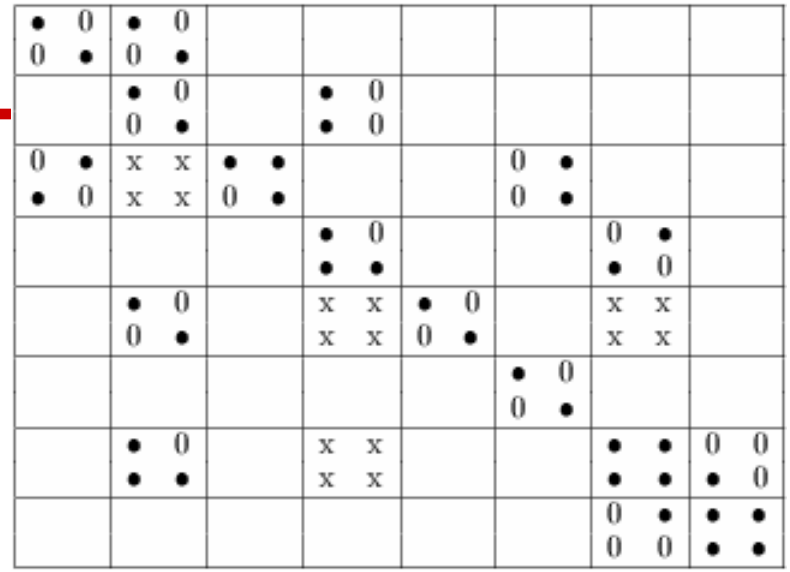
- **Solutions - continued**
  - **fast sparse xGEMM() is two-level algorithm**
    - **recursive operation on sparse data structures**
    - **dense xGEMM() call when recursion reaches single block**
- **Uses Reverse Cuthill-McKee ordering causing fill-in around the band**
- **No partial pivoting**
  - **use iterative improvement or**
  - **pivot only within blocks**

Recursive storage conversion steps

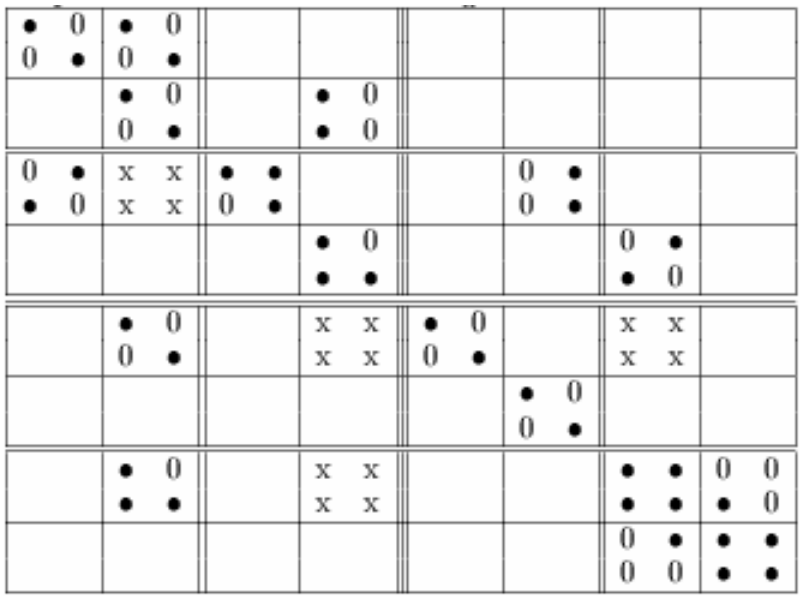
Matrix divided into 2x2 blocks



Matrix with explicit 0's and fill-in



Recursive algorithm division lines



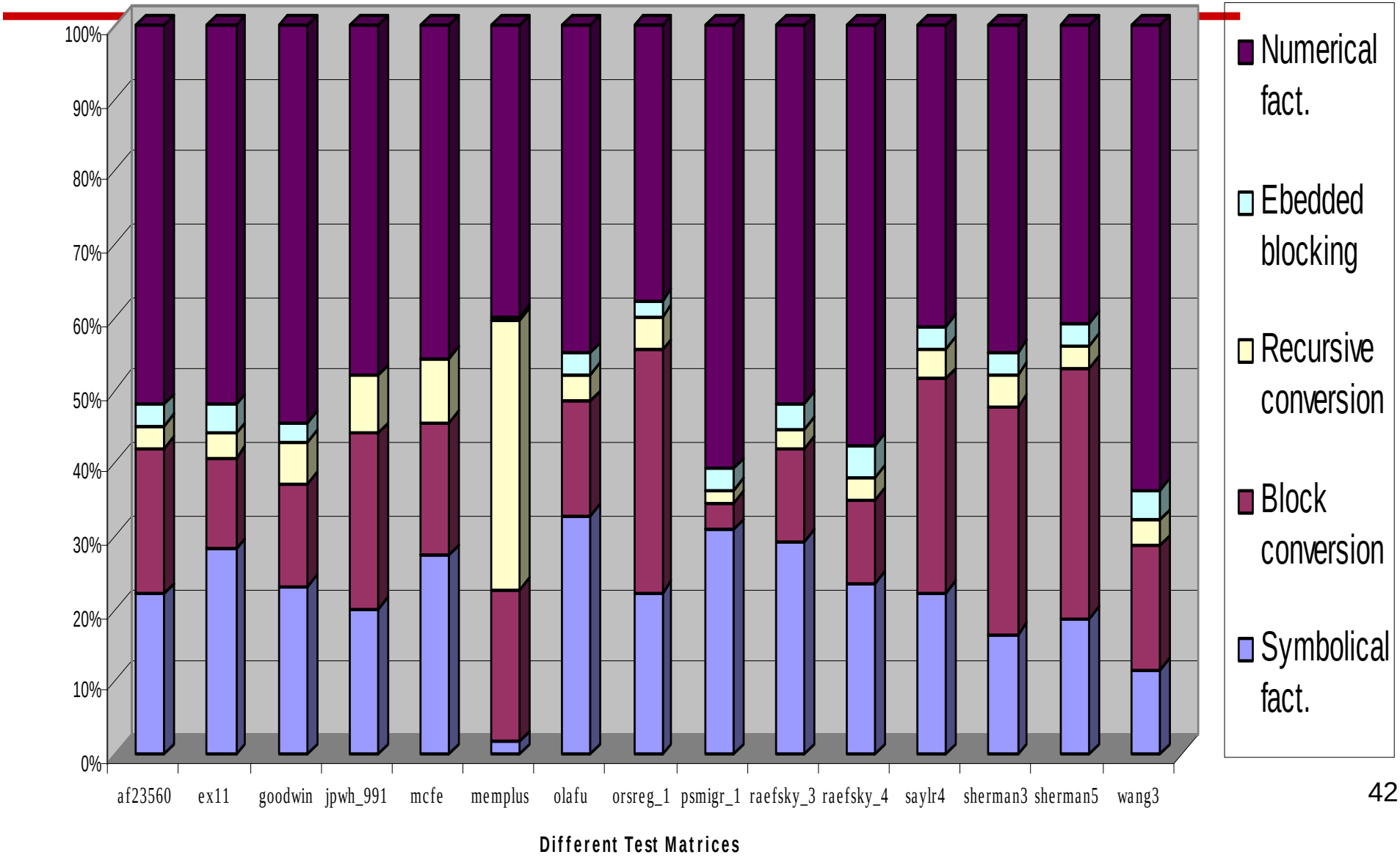
• - original nonzero value

0 - zero value introduced due to blocking

x - zero value introduced due to fill-in

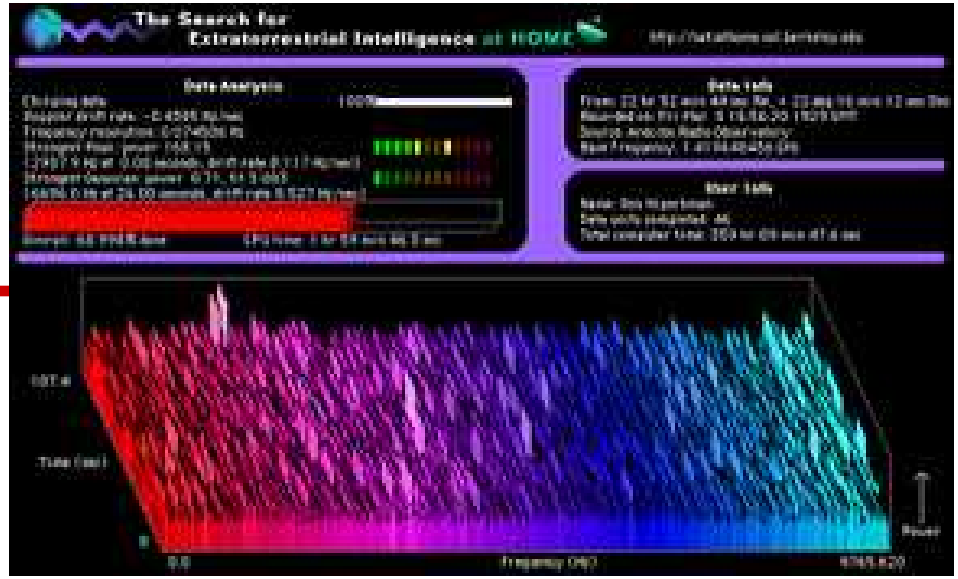
Architecture: Pentium III						
Matrix name	order	nonzeros	SuperLU		LUSR	
			time [s]	FERR	time [s]	FERR
af23560	23560	460598	44.19	5.8e-14	31.34	1.8e-14
ex11	16614	109694	109.67	2.5e-05	55.3	1.3e-06
goodwin	7320	324772	6.49	1.2e-08	6.74	4.6e-06
jpwh_991	991	6027	0.19	2.9e-15	0.25	2.6e-15
mcfe	765	24382	0.07	1.2e-13	0.22	9.1e-13
memplus	17758	126150	0.29	2.1e-12	12.67	6.6e-13
olafu	16146	1015156	26.16	1.1e-06	22.1	3.7e-09
orsreg_1	2205	14133	0.46	1.3e-13	0.45	2.1e-13
psmigr_1	3140	543162	110.79	7.9e-11	88.61	1.2e-05
raefsky3	21200	1488768	62.07	1.4e-09	69.67	4.4e-13
raefsky4	19779	1316789	82.45	2.3e-06	104.29	3.5e-06
saylr4	3564	22316	0.85	3.2e-11	0.95	1.2e-11
sherman3	5005	20033	0.61	6.0e-13	0.67	4.8e-13
sherman5	3312	20793	0.28	1.4e-13	0.32	6.2e-15
wang3	26064	177168	84.14	2.4e-14	79.18	1.6e-14

Breakdown of Time Across Phases For the Recursive Sparse Factorization



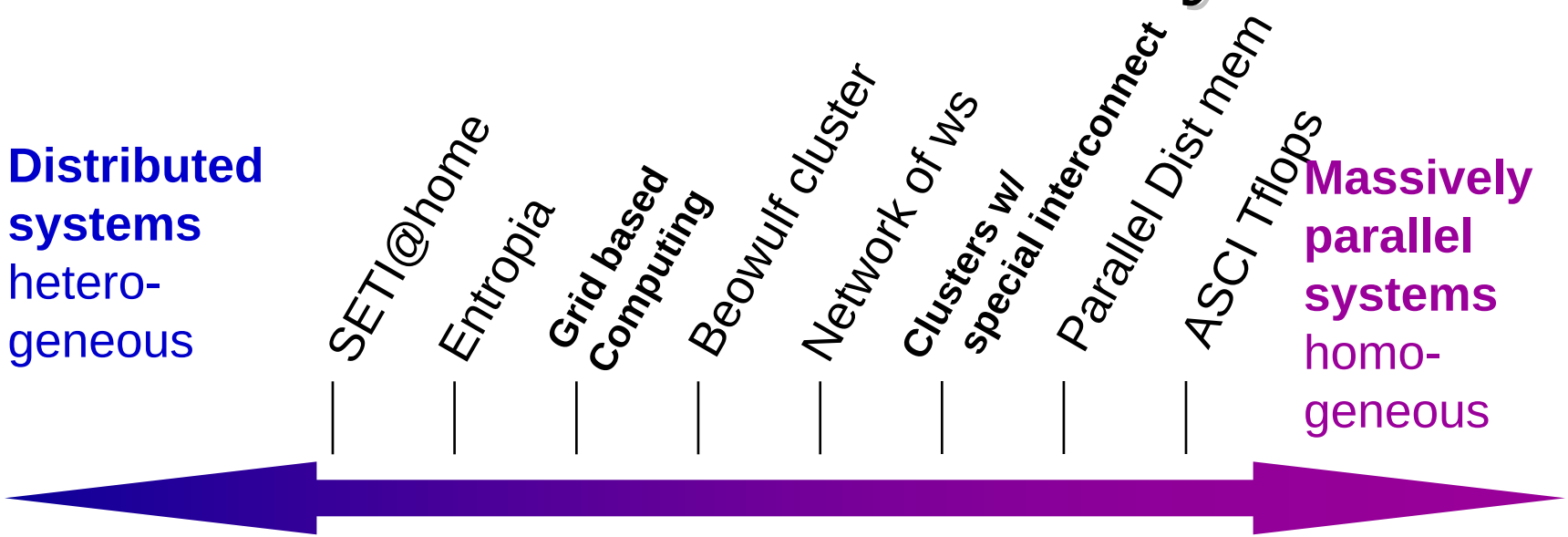
SETI@home

- 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.

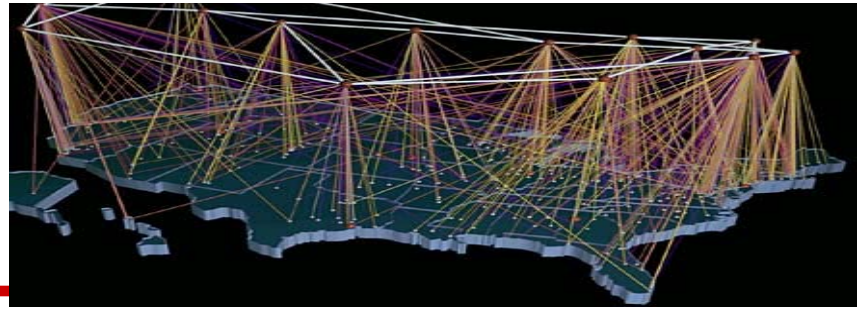
Distributed and Parallel Systems



- 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

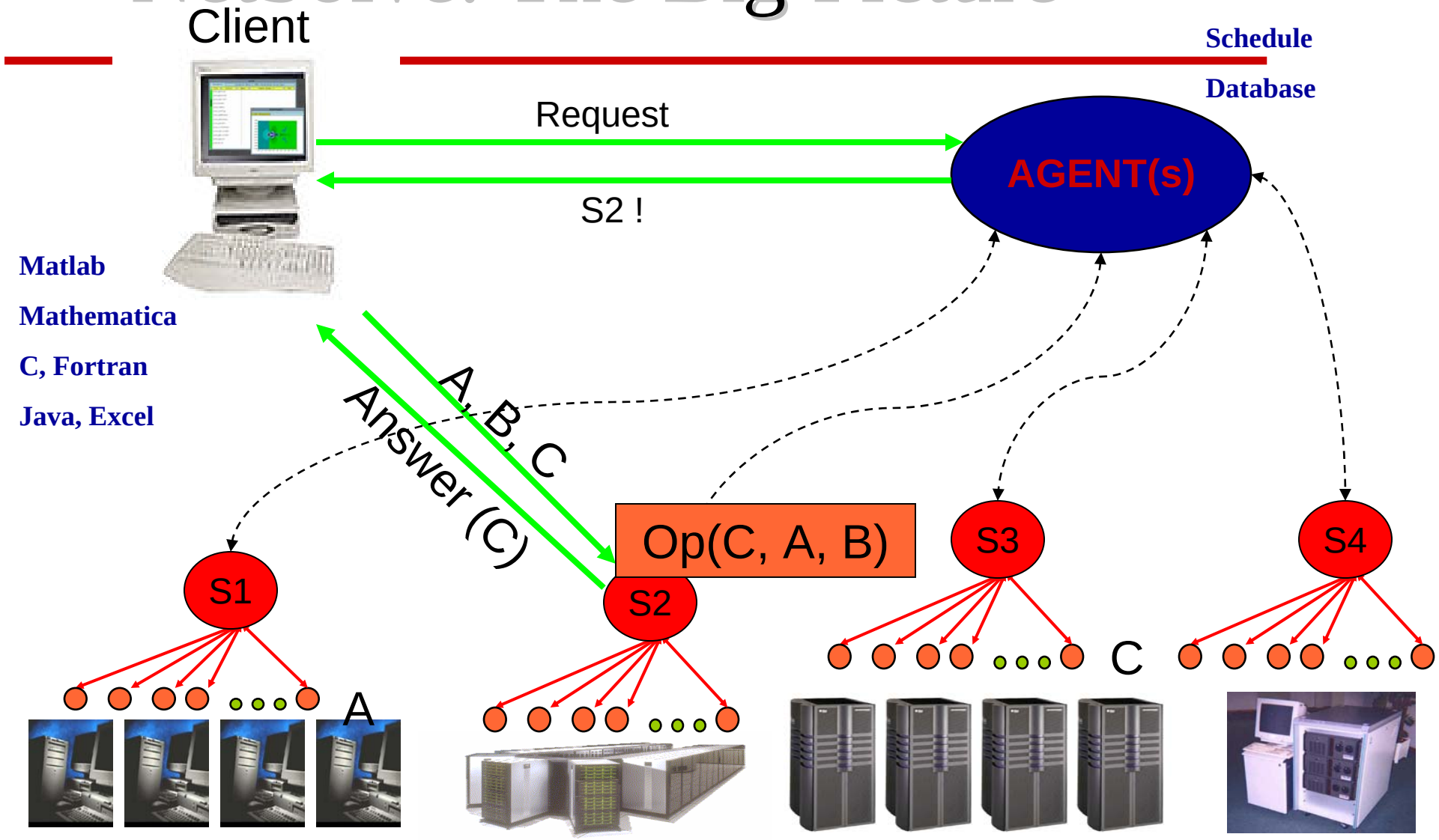


- To treat CPU cycles and software like commodities.
- Napster on steroids.
- Enable the coordinated use of geographically distributed resources – in the absence of central control and existing trust relationships.
- Computing power is produced much like utilities such as power and water are produced for consumers.
- Users will have access to “power” on demand

NetSolve
Network Enabled Server

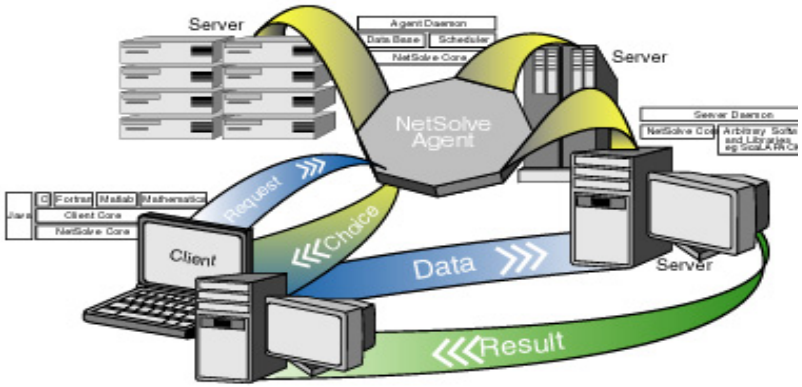
- **NetSolve is an example of a grid based hardware/software server.**
- **Easy-of-use paramount**
- **Based on a RPC model but with …**
  - **resource discovery, dynamic problem solving capabilities, load balancing, fault tolerance asynchronicity, security, …**
- **Other examples are NEOS from Argonne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

NetSolve: The Big Picture



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, its 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

Intel® Math Kernel Library 5.1 for IA32 and Itanium™ Processor-based Linux applications Beta License Agreement PRE-RELEASE

- **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:**
  http://developer.intel.com/software/products/mkl/mkllicense51_lnx.htm