IMACS | The IMACS World Congress

Computational and Applied Mathematics &
Applications in Science and Engineering

August 3-5, 2009
The University of Georgia
Center for Continuing Education
Athens, Georgia USA

Current Trends in High Performance Computing and Challenges for Mathematical Software

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

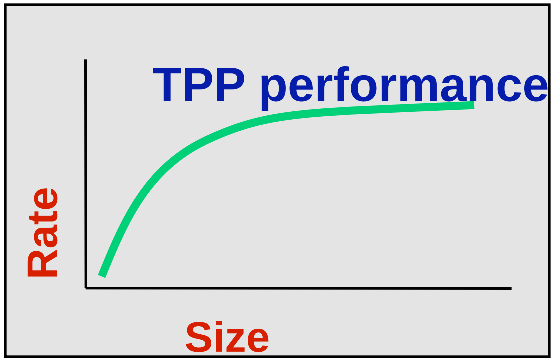
ICL UT

TOP500 superCOMPUTER

H. Meuer, H. Simon, E. Strohmaier, & JD

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

  $Ax=b$, *dense problem*



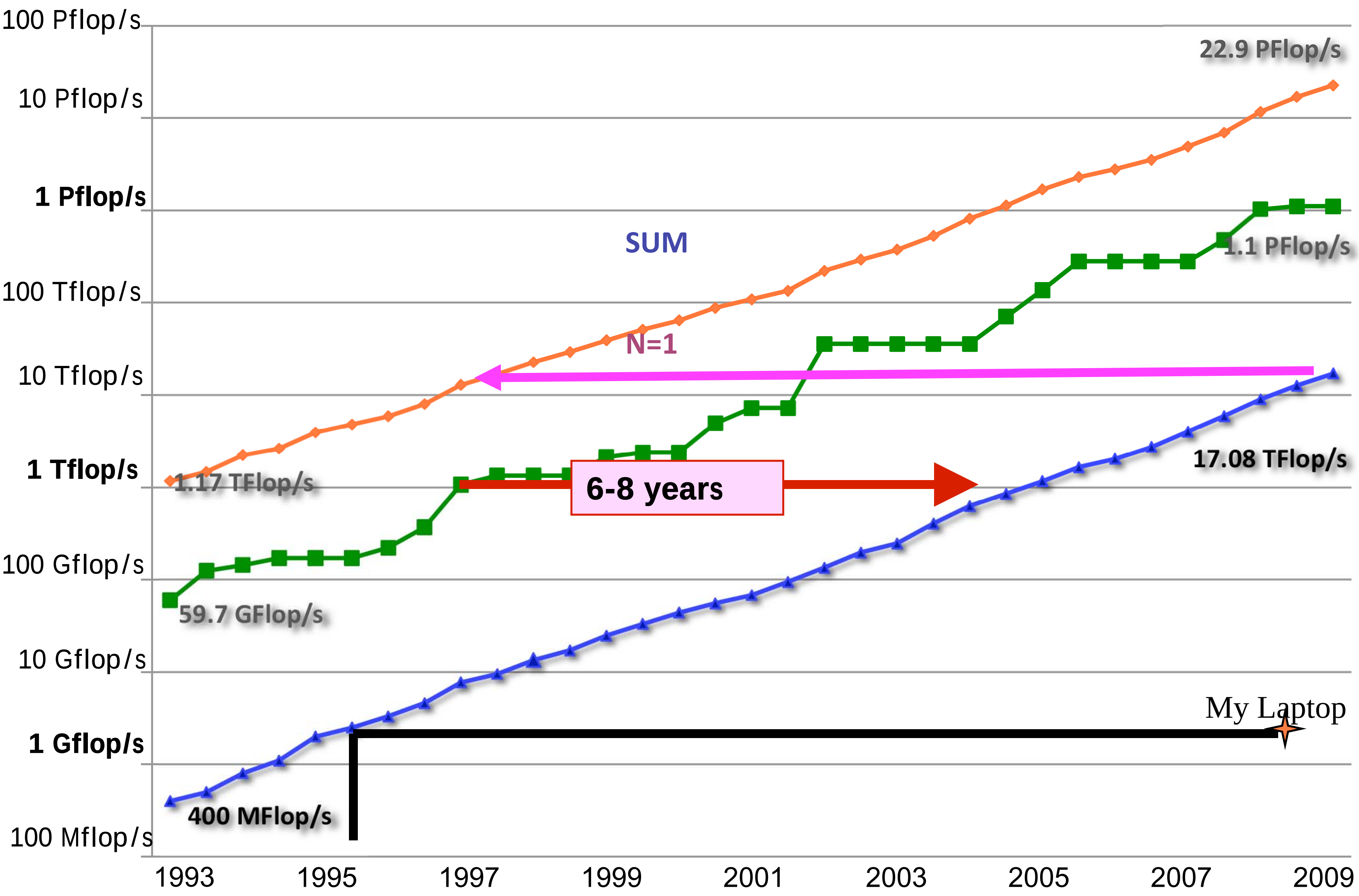
- Updated twice a year

  SC'xy in the States in November

  Meeting in Germany in June

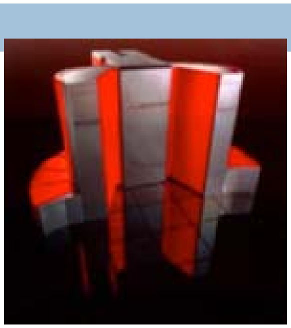
- All data available from **www.top500.org**

Performance Development



Looking at the Gordon Bell Prize

(Recognize outstanding achievement in high-performance computing applications and encourage development of parallel processing)

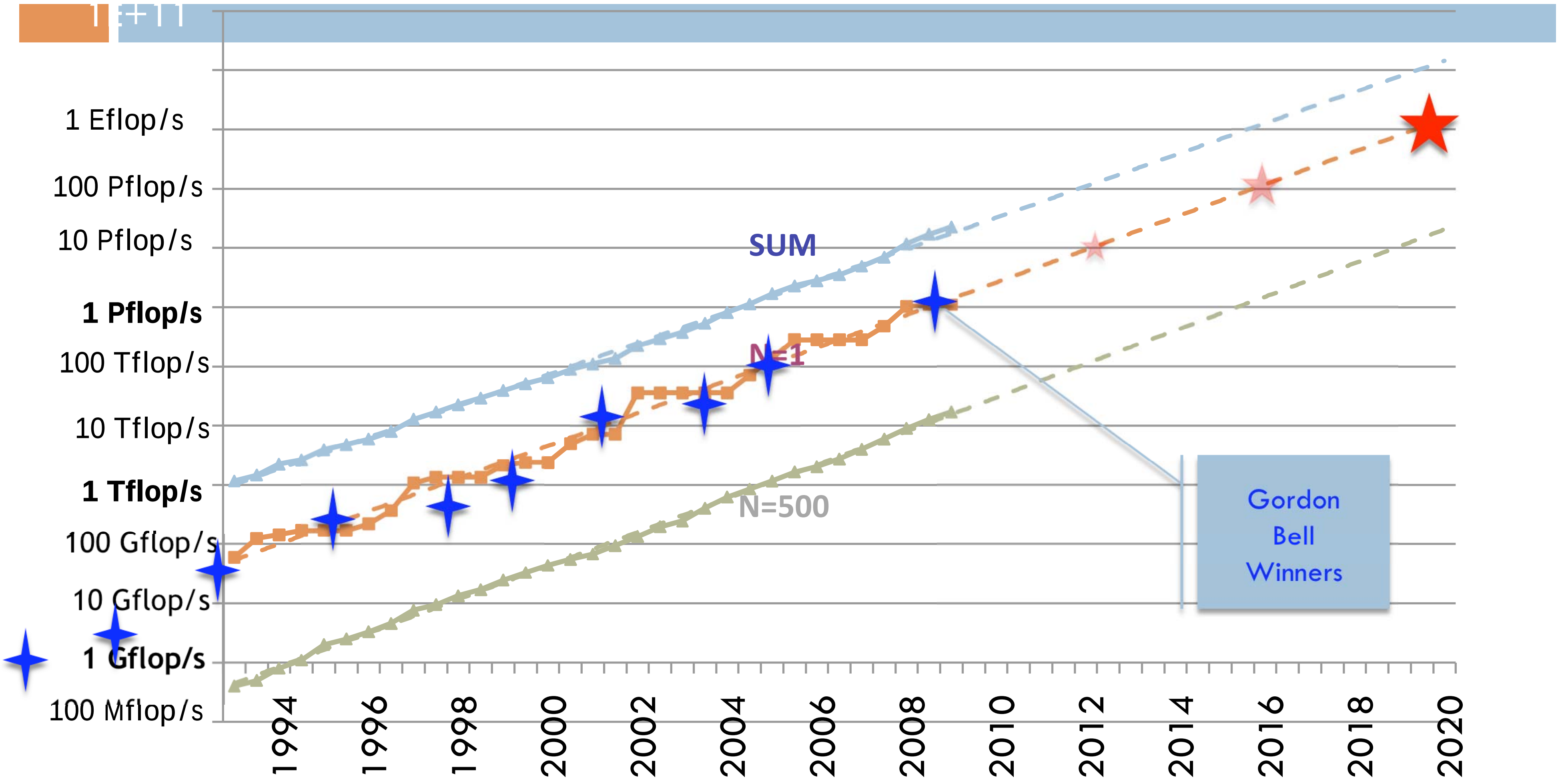


- 1 GFlop/s; 1988; Cray Y-MP; 8 Processors
  - Static finite element analysis
- 1 TFlop/s; 1998; Cray T3E; 1024 Processors
  - Modeling of metallic magnet atoms, using a variation of the locally self-consistent multiple scattering method.

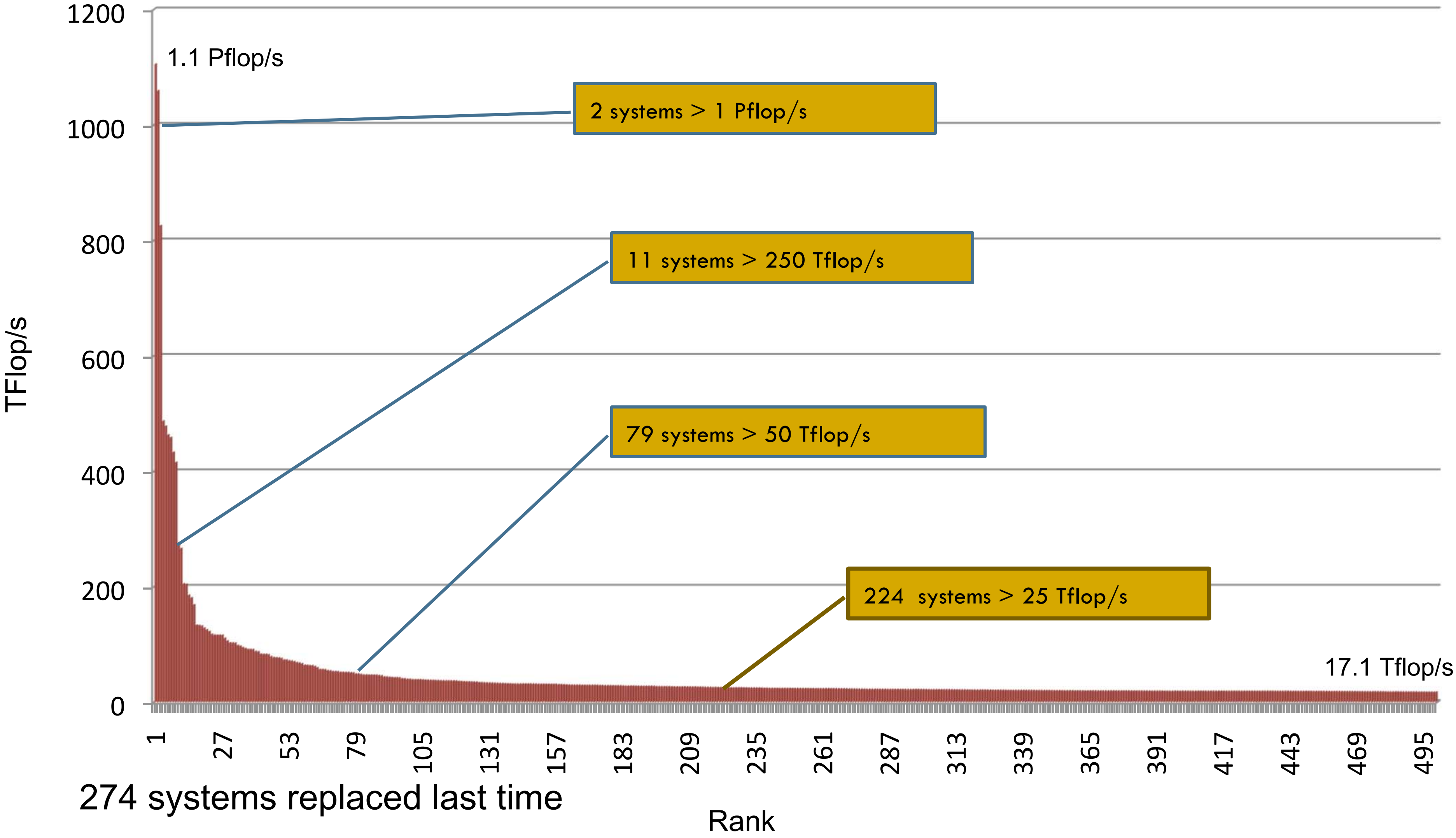
- 1 PFlop/s; 2008; Cray XT5; 1.5x10^5 Processors
  - Superconductive materials

- 1 EFlop/s; ~2018; ?; 1x10^7 Processors (10^9 threads)

Performance Development in Top500



Distribution of the Top500



1.1 Pflop/s

2 systems > 1 Pflop/s

11 systems > 250 Tflop/s

79 systems > 50 Tflop/s

224 systems > 25 Tflop/s

17.1 Tflop/s

274 systems replaced last time

33rd List: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Tflops]	% of Peak
1	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenter QS22/LS21	USA	129,600	1,105	76
2	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5 QC 2.3 GHz	USA	150,152	1,059	77
3	Forschungszentrum Juelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	825	82
4	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	51,200	480	79
5	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServer Blue Gene Solution	USA	212,992	478	80
6	NSF NICS/U of Tennessee	Kraken / Cray Cray XT5 QC 2.3 GHz	USA	66,000	463	76
7	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	458	82
8	NSF TACC/U. of Texas	Ranger / Sun SunBlade x6420	USA	62,976	433	75
9	DOE / NNSA Lawrence Livermore NL	Dawn / IBM Blue Gene/P Solution	USA	147,456	415	83
10	Forschungszentrum Juelich (FZJ)	JUROPA /Sun - Bull SA NovaScale /Sun Blade	Germany	26,304	274	89

33rd List: The TOP10

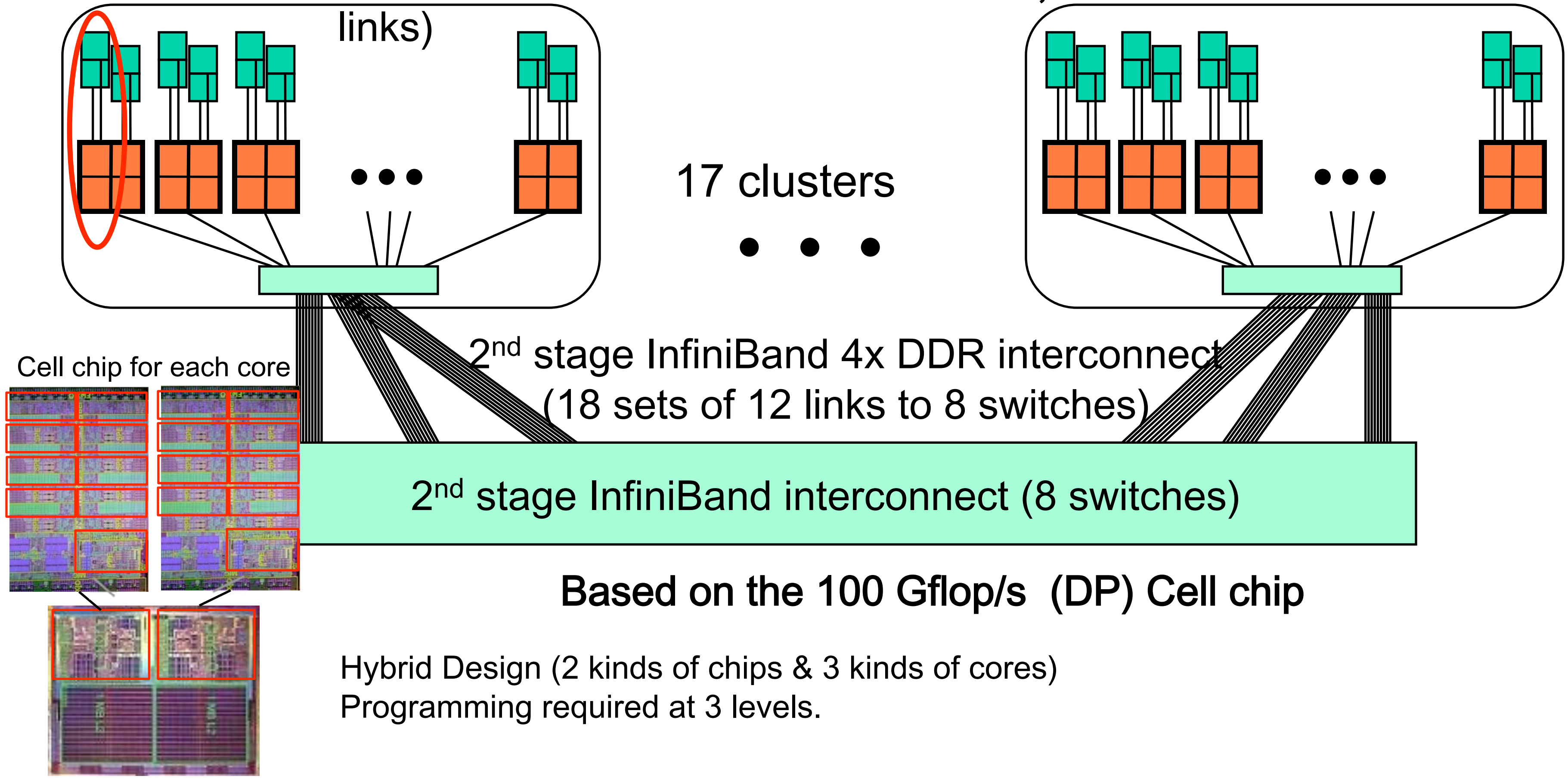
Rank	Site	Computer	Country	Cores	Rmax [Tflops]	% of Peak	Power [MW]	Flops/ Watt
1	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenter QS22/LS21	USA	129,600	1,105	76	2.48	446
2	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5 QC 2.3 GHz	USA	150,152	1,059	77	6.95	151
3	Forschungszentrum Juelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	825	82	2.26	365
4	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	51,200	480	79	2.09	230
5	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServer Blue Gene Solution	USA	212,992	478	80	2.32	206
6	NSF NICS/U of Tennessee	Kraken / Cray Cray XT5 QC 2.3 GHz	USA	66,000	463	76		
7	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	458	82	1.26	363
8	NSF TACC/U. of Texas	Ranger / Sun SunBlade x6420	USA	62,976	433	75	2.0	217
9	DOE / NNSA Lawrence Livermore NL	Dawn / IBM Blue Gene/P Solution	USA	147,456	415	83	1.13	367
10	Forschungszentrum Juelich (FZJ)	JUROPA /Sun - Bull SA NovaScale /Sun Blade	Germany	26,304	274	89	1.54	178

LANL Roadrunner
A Petascale System in 2008

≈ 13,000 Cell HPC chips
- ≈ **1.33 PetaFlop/s** (from Cell)

≈ 7,000 dual-core Opterons

≈ 122,000 cores



"Connected Unit" cluster
192 Opteron nodes
(180 w/ 2 dual-Cell blades
connected w/ 4 PCIe x8
links)

17 clusters

2nd stage InfiniBand 4x DDR interconnect
(18 sets of 12 links to 8 switches)

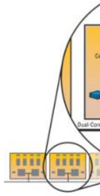
2nd stage InfiniBand interconnect (8 switches)

Cell chip for each core

Dual Core Opteron Chip

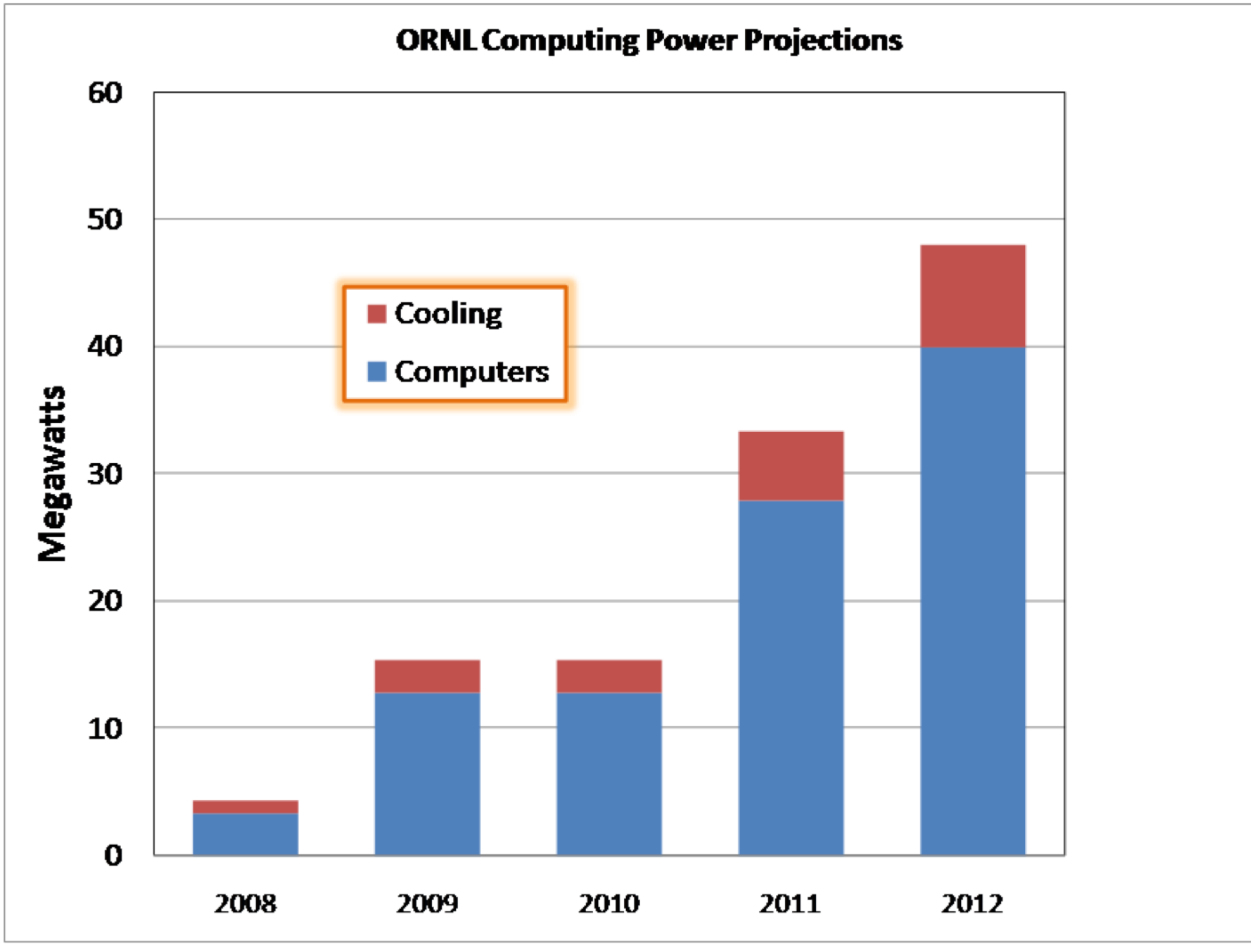
Based on the 100 Gflop/s (DP) Cell chip

Hybrid Design (2 kinds of chips & 3 kinds of cores)
Programming required at 3 levels.



ORNL/UTK Computer Power Cost Projections 2008-2012

- Over the next 5 years ORNL/UTK will deploy 2 large Petascale systems
- Using 15 MW today
- By 2012 close to 50MW!!
- Power costs greater than $10M today.
- Cost estimates based on $0.07 per KwH

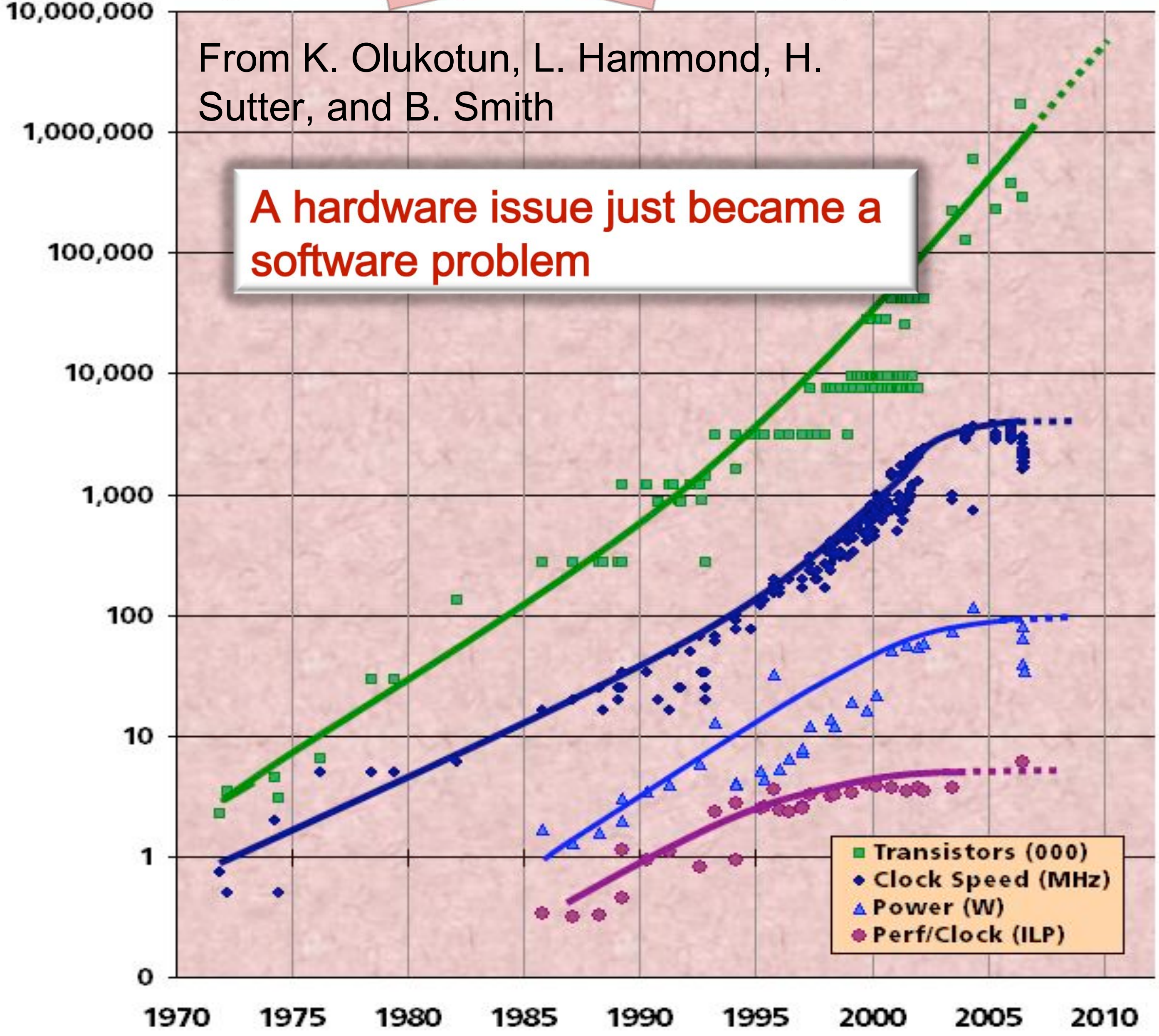


> $10M
> $20M
> $30M

Cost Per Year

Power consumption and thermal management gas becomes the architectural driver for future large systems and may be a limiting factor.

Why All ~~Powerful~~ ~~Scientific~~ Computers are Parallel



From K. Olukotun, L. Hammond, H. Sutter, and B. Smith

A hardware issue just became a software problem

- In the "old days" it was: each year processors would become faster
- Today the clock speed is fixed or getting slower
- Things are still doubling every 18 -24 months
- Moore's Law reinterpretated.
  - Number of cores double every 18-24 months

Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^2F)
- **Frequency ∝ Voltage**
- Power ∝ Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X

Power Cost of Frequency

- Power $\propto$ Voltage^2 x Frequency (V^{2}F)
- Frequency $\propto$ Voltage
- Power $\propto$ Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X
Multicore	2X	0.75X	0.75X	1.5X	0.8X	1.88X

(Bigger # is better)

50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

Parallelism in 2009?

- **These arguments are no longer theoretical**
- **All major processor vendors are producing *multicore* chips**
  - **Every machine will soon be a parallel machine**
  - **To keep doubling performance, parallelism must double**
- **Which commercial applications can use this parallelism?**
  - **Do they have to be rewritten from scratch?**
- **Will all programmers have to be parallel programmers?**
  - **New software model needed**
  - **Try to hide complexity from most programmers – eventually**
  - **In the meantime, need to understand it**
- **Computer industry betting on this big change, but does not have all the answers**

Moore's Law Reinterpreted

- **Number of cores per chip doubles every 2 year, while clock speed remains fixed or decreases**
- **Need to deal with systems with millions of concurrent threads**
    - **Future generation will have billions of threads!**
- **Number of threads of execution doubles every 2 year**

Major Changes to Software

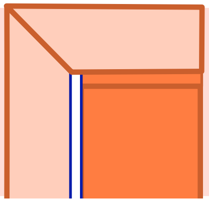
- **Must rethink the design of our software**
    - **Another disruptive technology**
        - **Similar to what happened with cluster computing and message passing**
    - **Rethink and rewrite the applications, algorithms, and software**
- **Numerical libraries for example will change**
    - **For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this**

Five Important Features to Consider When Computing at Scale

- **Effective Use of Many-Core and Hybrid architectures**
  - **Dynamic Data Driven Execution**
  - **Block Data Layout**
- **Exploiting Mixed Precision in the Algorithms**
  - **Single Precision is 2X faster than Double Precision**
  - **With GP-GPUs 10x**
- **Self Adapting / Auto Tuning of Software**
  - **Too hard to do by hand**
- **Fault Tolerant Algorithms**
  - **With 1,000,000's of cores things will fail**
- **Communication Avoiding Algorithms**
  - **For dense computations from O(n *log* p) to O(*log* p) communications**
  - **GMRES s-step compute (x, Ax, A^2x, ... A^sx)**

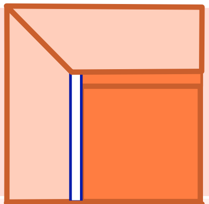
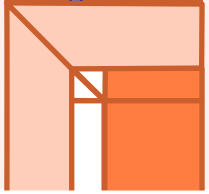
A New Generation of Software:

Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations

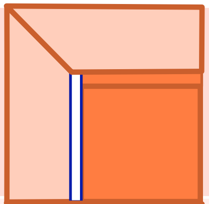
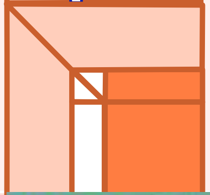
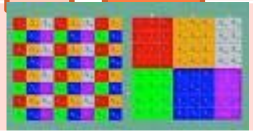
A New Generation of Software:

Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations

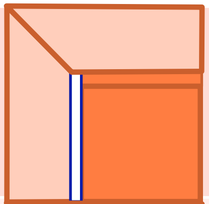
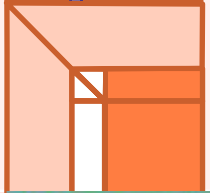
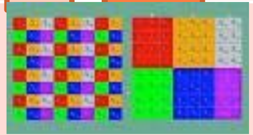
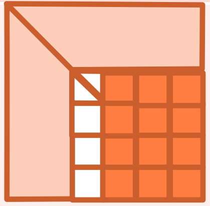
A New Generation of Software:

Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing

A New Generation of Software:

Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (00's) New Algorithms (many-core friendly)		Rely on - a DAG/scheduler - block data layout - some extra kernels

Those new algorithms

- have a very low granularity, they scale very well (multicore, petascale computing, …)
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

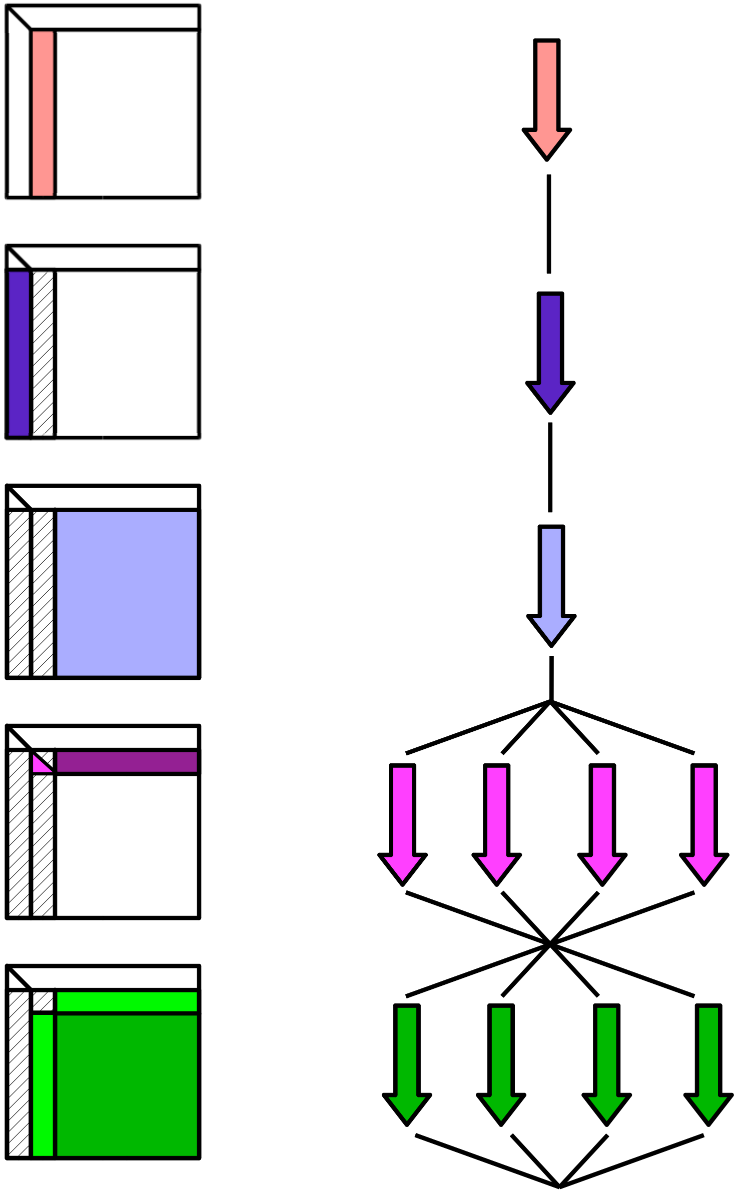
Those new algorithms need new kernels and rely on efficient scheduling algorithms.

Coding for an Abstract Multicore

Parallel software for multicores should have two characteristics:

- Fine granularity:
  - High level of parallelism is needed
  - Cores will probably be associated with relatively small local memories. This requires splitting an operation into tasks that operate on small portions of data in order to reduce bus traffic and improve data locality.
- Asynchronicity:
  - As the degree of thread level parallelism grows and granularity of the operations becomes smaller, the presence of synchronization points in a parallel execution seriously affects the efficiency of an algorithm.

Steps in the LAPACK LU



DGETF2
(Factor a panel)

LAPACK

DLSWP
(Backward swap)

LAPACK

DLSWP
(Forward swap)

LAPACK

DTRSM
(Triangular solve)

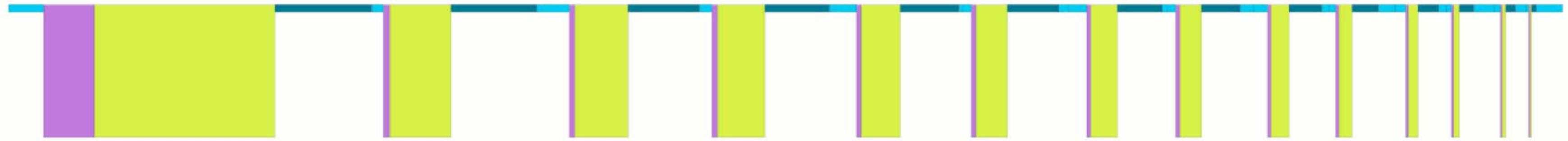
BLAS

DGEMM
(Matrix multiply)

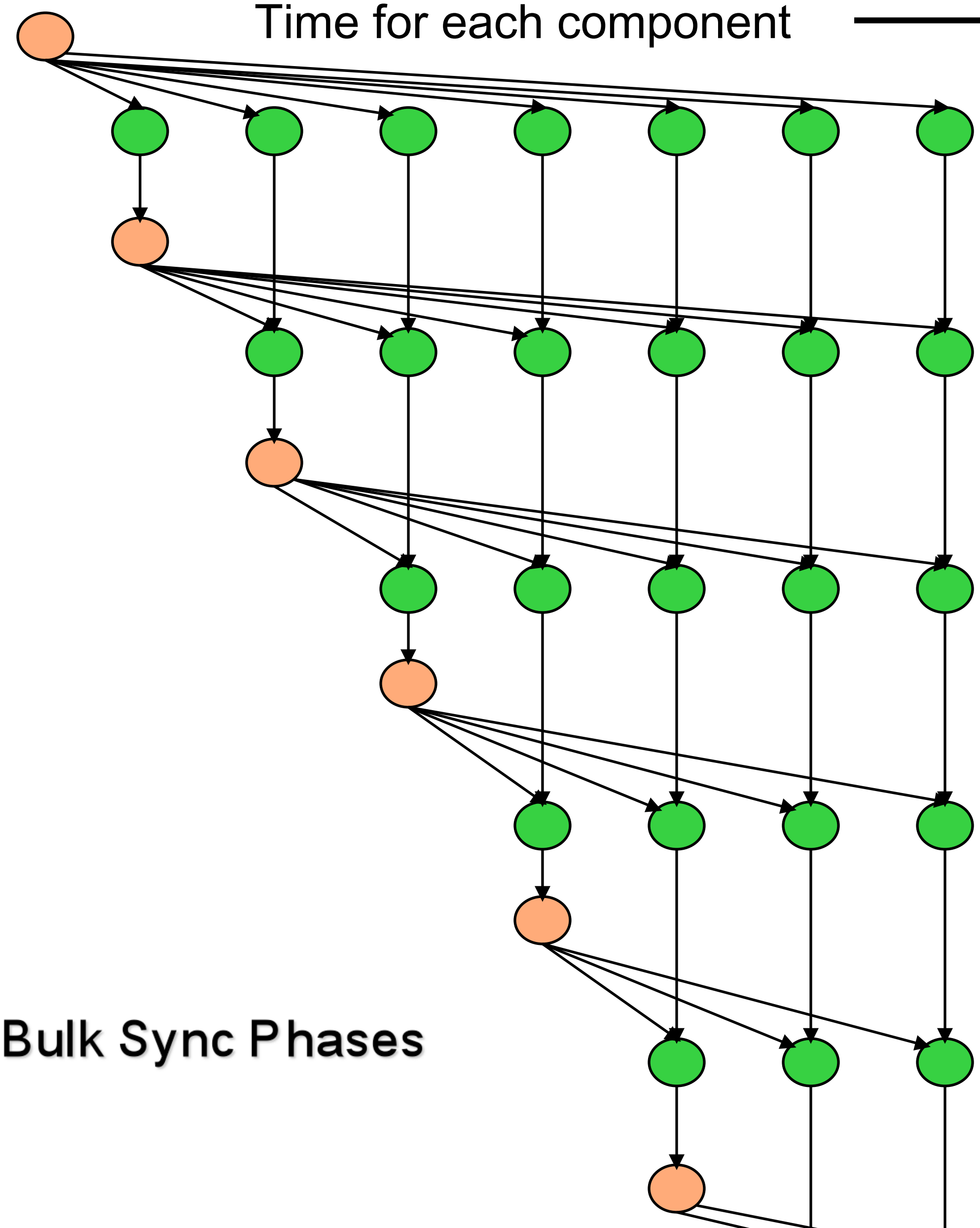
BLAS

LU Timing Profile (16 core system)

Threads – no lookahead



Time for each component →



Bulk Sync Phases

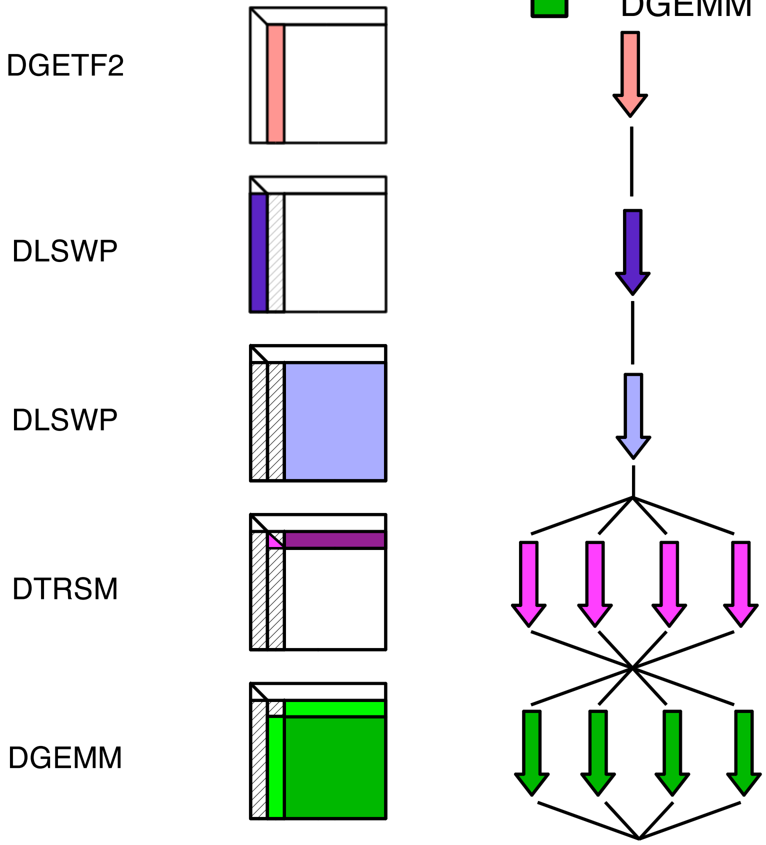
🟧 DGETF2

🟦 DLASWP(L)

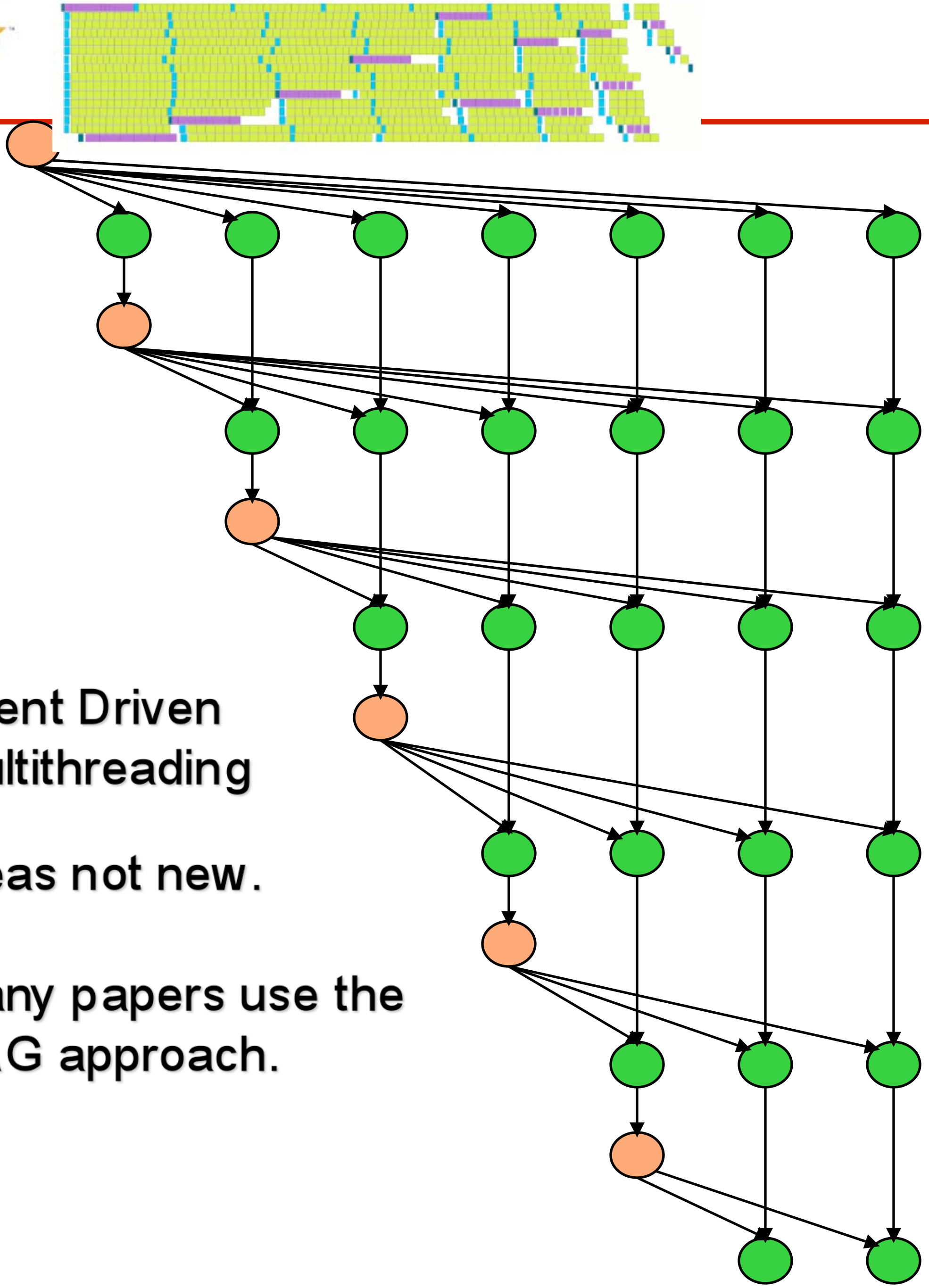
🟦 DLASWP(R)

🟪 DTRSM

🟩 DGEMM



Adaptive Lookahead - Dynamic



Event Driven Multithreading

Ideas not new.

Many papers use the DAG approach.

```
while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:                        //
            dgetf2();
            update_progress();
        case COLUMN:                       //
            dlaswp();                      //
            dtrsm();
            dgemm();
            update_progress();
        case END:                          //
            for()
                dlaswp();
            return;
    }
}
```

Reorganizing algorithms to use this approach

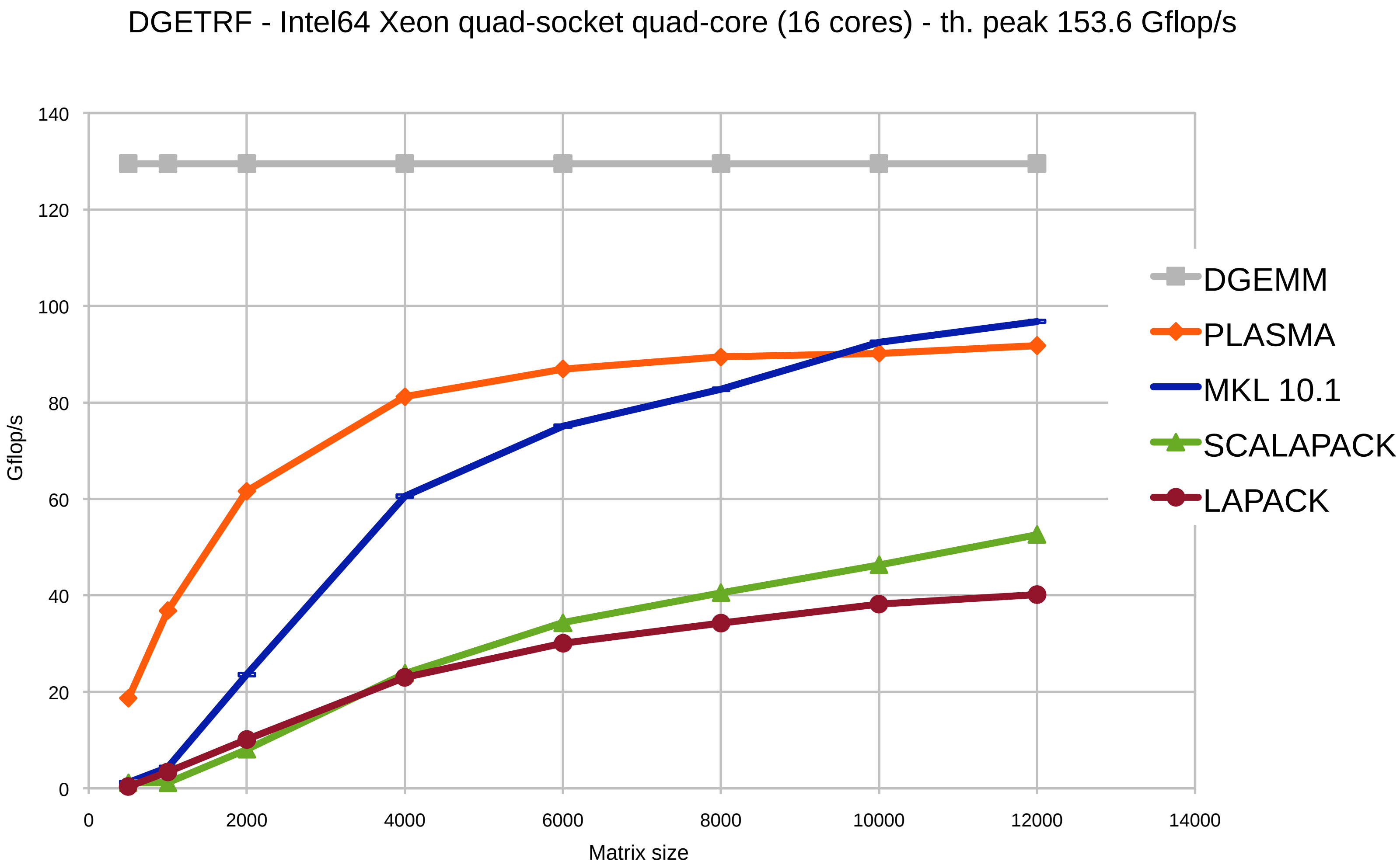
PLASMA (Redesign LAPACK/ScaLAPACK)

Parallel Linear Algebra Software for Multicore Architectures

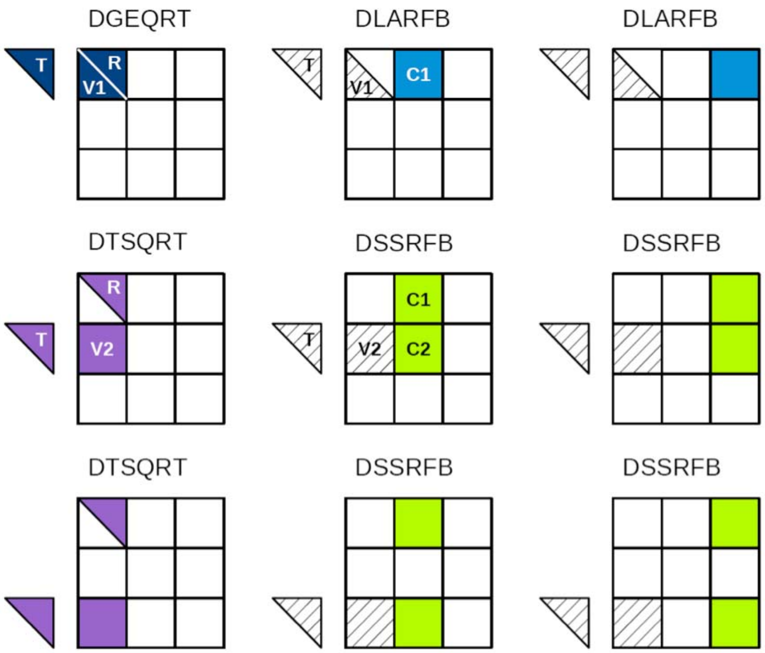
- **Asychronicity**
  - **Avoid fork-join (Bulk sync design)**
- **Dynamic Scheduling**
  - **Out of order execution**
- **Fine Granularity**
  - **Independent block operations**
- **Locality of Reference**
  - **Data storage – Block Data Layout**

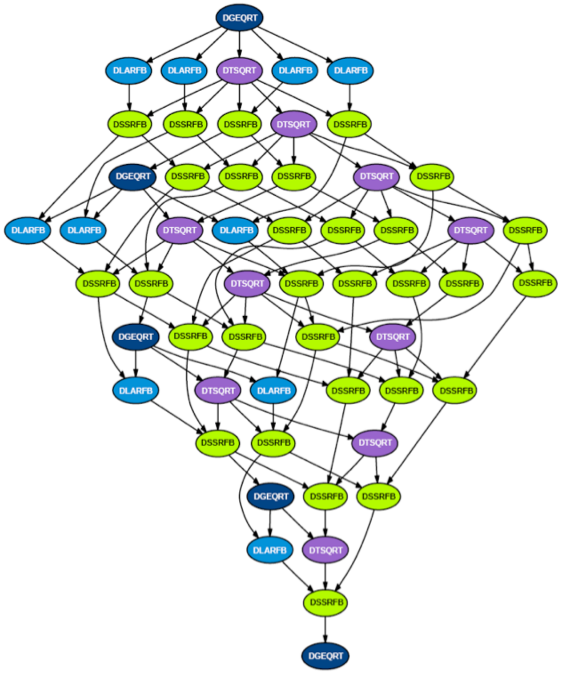
Lead by Tennessee and Berkeley similar to LAPACK/ScaLAPACK as a community effort

DGETRF - Intel64 - 16 cores



Tile QR (&LU) Algorithms

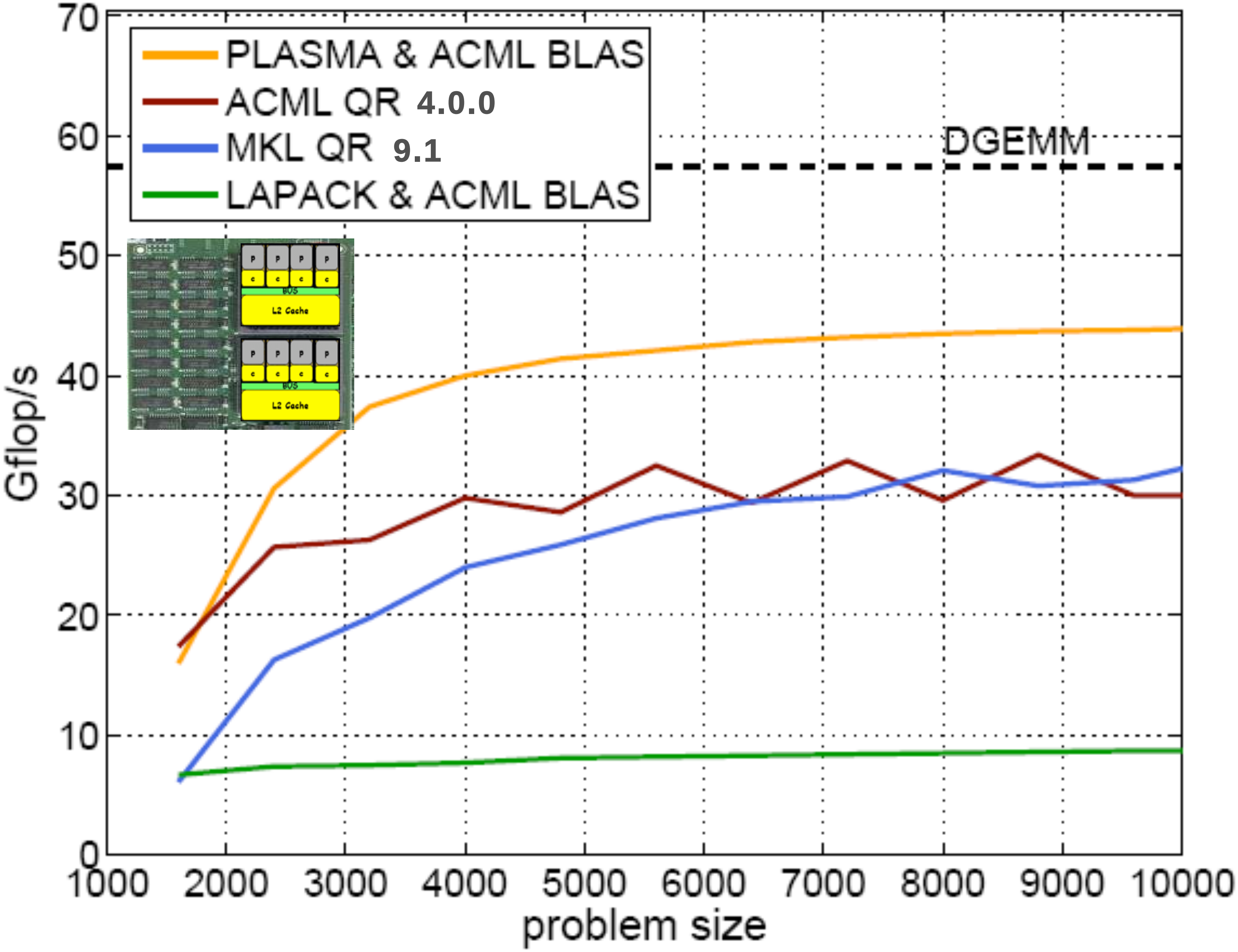



DGEQRT
DLARFB DLARFB DTSQRT DLARFB DLARFB
DSSRFB DSSRFB DSSRFB DTSQRT DSSRFB
DGEQRT DSSRFB DSSRFB DSSRFB DTSQRT DSSRFB
DLARFB DLARFB DTSQRT DLARFB DSSRFB DSSRFB DSSRFB DTSQRT DSSRFB
DSSRFB DSSRFB DTSQRT DSSRFB DSSRFB DSSRFB DSSRFB DSSRFB
DGEQRT DSSRFB DSSRFB DSSRFB DTSQRT
DLARFB DTSQRT DLARFB DSSRFB DSSRFB DSSRFB
DSSRFB DSSRFB DTSQRT
DGEQRT DSSRFB DSSRFB
DLARFB DTSQRT
DSSRFB
DGEQRT

```
FOR k = 0..TILES-1
  A[k][k], T[k][k] ← DGRQRT(A[k][k])
  FOR m = k+1..TILES-1
    A[k][k], A[m][k], T[m][k] ← DTSQRT(A[k][k], A[m][k], T[m][k])
  FOR n = k+1..TILES-1
    A[k][n] ← DLARFB(A[k][k], T[k][k], A[k][n])
    FOR m = k+1..TILES-1
      A[k][n], A[m][n] ← DSSRFB(A[m][k], T[m][k], A[k][n], A[m][n])
```

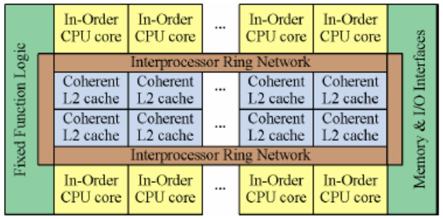
- input matrix stored and processed by square tiles
- complex DAG

QR -- quad-socket, dual-core Opteron



Future Computer Systems

- Most likely be a hybrid design
- Think standard multicore chips and accelerator (GPUs)
- Today accelerators are attached
- Next generation more integrated
- Intel's Larrabee in 2010
  - 8,16,32,or 64 x86 cores
- AMD's Fusion in 2011
  - Multicore with embedded graphics ATI
- Nvidia's plans?

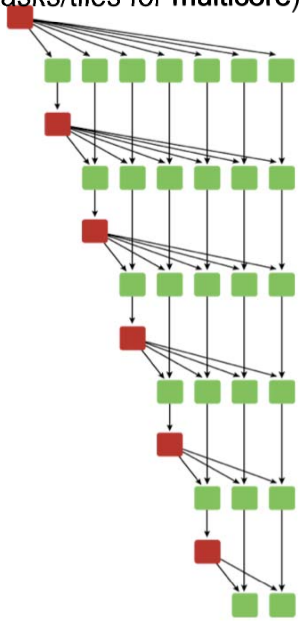


Intel Larrabee

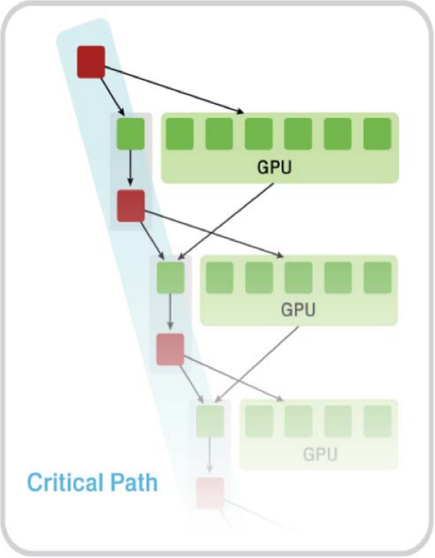
Hybrid Computing

- **Match algorithmic requirements to architectural strengths of the hybrid components**
  Multicore : small tasks/tiles
  Accelerator: large data parallel tasks

Algorithms as DAGs
(small tasks/tiles for **multicore**)



Current hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)



- **e.g. split the computation into tasks; define critical path that "clears" the way for other large data parallel tasks; proper schedule the tasks execution**
- **Design algorithms with well defined "*search space*" to facilitate auto-tuning**

Performance of Single Precision on Conventional Processors

- **Realized have the similar situation on our commodity processors.**
  - **That is, SP is 2X as fast as DP on many systems**
- **The Intel Pentium and AMD Opteron have SSE2**
  - **2 flops/cycle DP**
  - **4 flops/cycle SP**
- **IBM PowerPC has AltiVec**
  - **8 flops/cycle SP**
  - **4 flops/cycle DP**
    - **No DP on AltiVec**

	Size	SGEMM/ DGEMM	Size	SGEMV/ DGEMV
AMD Opteron 246	3000	2.00	5000	1.70
UltraSparc-IIe	3000	1.64	5000	1.66
Intel PIII Coppermine	3000	2.03	5000	2.09
PowerPC 970	3000	2.04	5000	1.44
Intel Woodcrest	3000	1.81	5000	2.18
Intel XEON	3000	2.04	5000	1.82
Intel Centrino Duo	3000	2.71	5000	2.21

Single precision is faster because:

- Operations are faster
- Reduced data motion
- Larger blocks gives higher locality in cache

Idea Goes Something Like This…

- **Exploit 32 bit floating point as much as possible.**
  - **Especially for the bulk of the computation**
- **Correct or update the solution with selective use of 64 bit floating point to provide a refined results**
- **Intuitively:**
  - **Compute a 32 bit result,**
  - **Calculate a correction to 32 bit result using selected higher precision and,**
  - **Perform the update of the 32 bit results with the correction using high precision.**

Mixed-Precision Iterative Refinement

- **Iterative refinement for dense systems, *Ax* = *b*, can work this way.**

```
L U = lu(A)                                   $O(n^3)$
x = L\(U\b)                                   $O(n^2)$
r = b – Ax                                    $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)                           $O(n^2)$
        x = x + z                             $O(n^1)$
        r = b – Ax                            $O(n^2)$
END
```

- **Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.**

Mixed-Precision Iterative Refinement

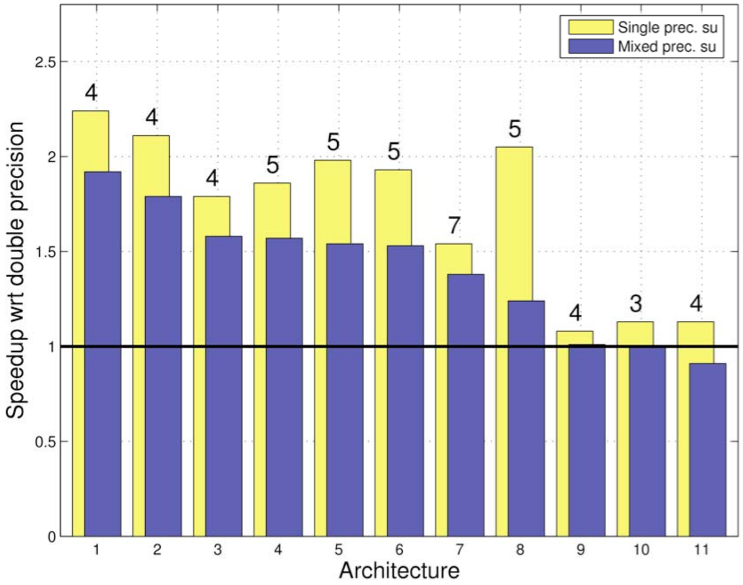
- **Iterative refinement for dense systems, *Ax = b*, can work this way.**

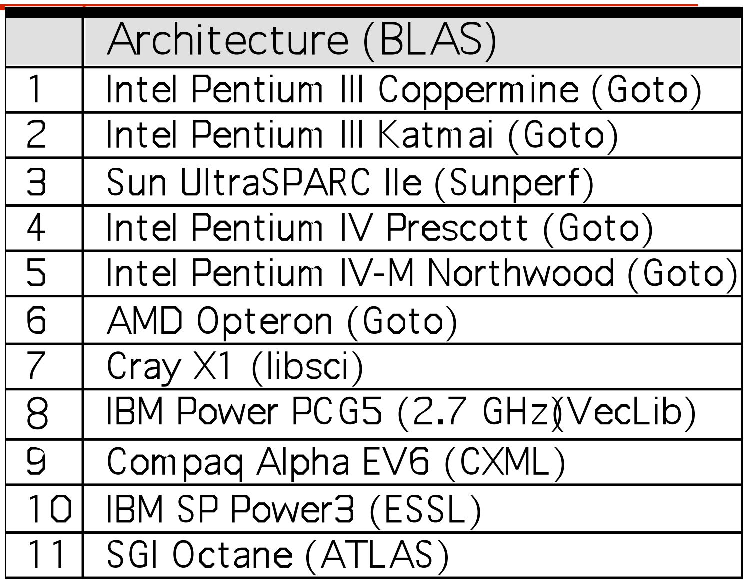
```
L U = lu(A)                 SINGLE      $O(n^3)$
x = L\(U\b)                 SINGLE      $O(n^2)$
r = b - Ax                  DOUBLE      $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)         SINGLE      $O(n^2)$
        x = x + z           DOUBLE      $O(n^1)$
        r = b - Ax          DOUBLE      $O(n^2)$
END
```

- **Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.**
- **It can be shown that using this approach we can compute the solution to 64-bit floating point precision.**

- **Requires extra storage, total is 1.5 times normal;**
- **$O(n^3)$ work is done in lower precision**
- **$O(n^2)$ work is done in high precision**
- **Problems if the matrix is ill-conditioned in sp; $O(10^8)$**

Results for Mixed Precision Iterative Refinement for Dense *Ax* = *b*

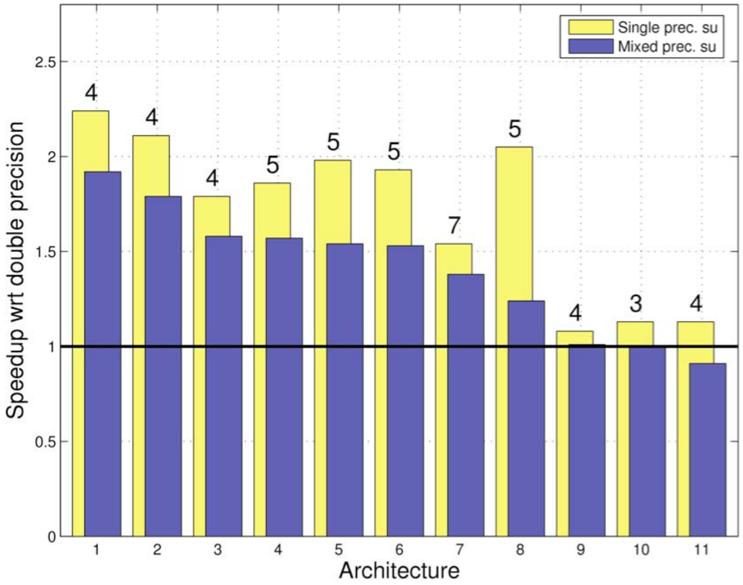


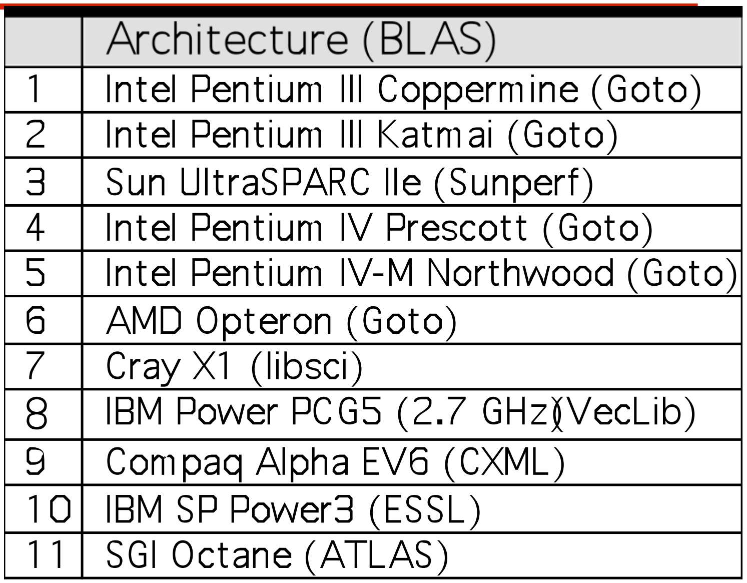


	Architecture (BLAS)
1	Intel Pentium III Coppermine (Goto)
2	Intel Pentium III Katmai (Goto)
3	Sun UltraSPARC IIe (Sunperf)
4	Intel Pentium IV Prescott (Goto)
5	Intel Pentium IV-M Northwood (Goto)
6	AMD Opteron (Goto)
7	Cray X1 (libsci)
8	IBM Power PC G5 (2.7 GHz)(VecLib)
9	Compaq Alpha EV6 (CXML)
10	IBM SP Power3 (ESSL)
11	SGI Octane (ATLAS)

- Single precision is faster than DP because:
  - **Higher parallelism within vector units**
    - **4 ops/cycle (usually) instead of 2 ops/cycle**
  - **Reduced data motion**
    - **32 bit data instead of 64 bit data**
  - **Higher locality in cache**
    - **More data items in cache**

Results for Mixed Precision Iterative Refinement for Dense *Ax* = *b*





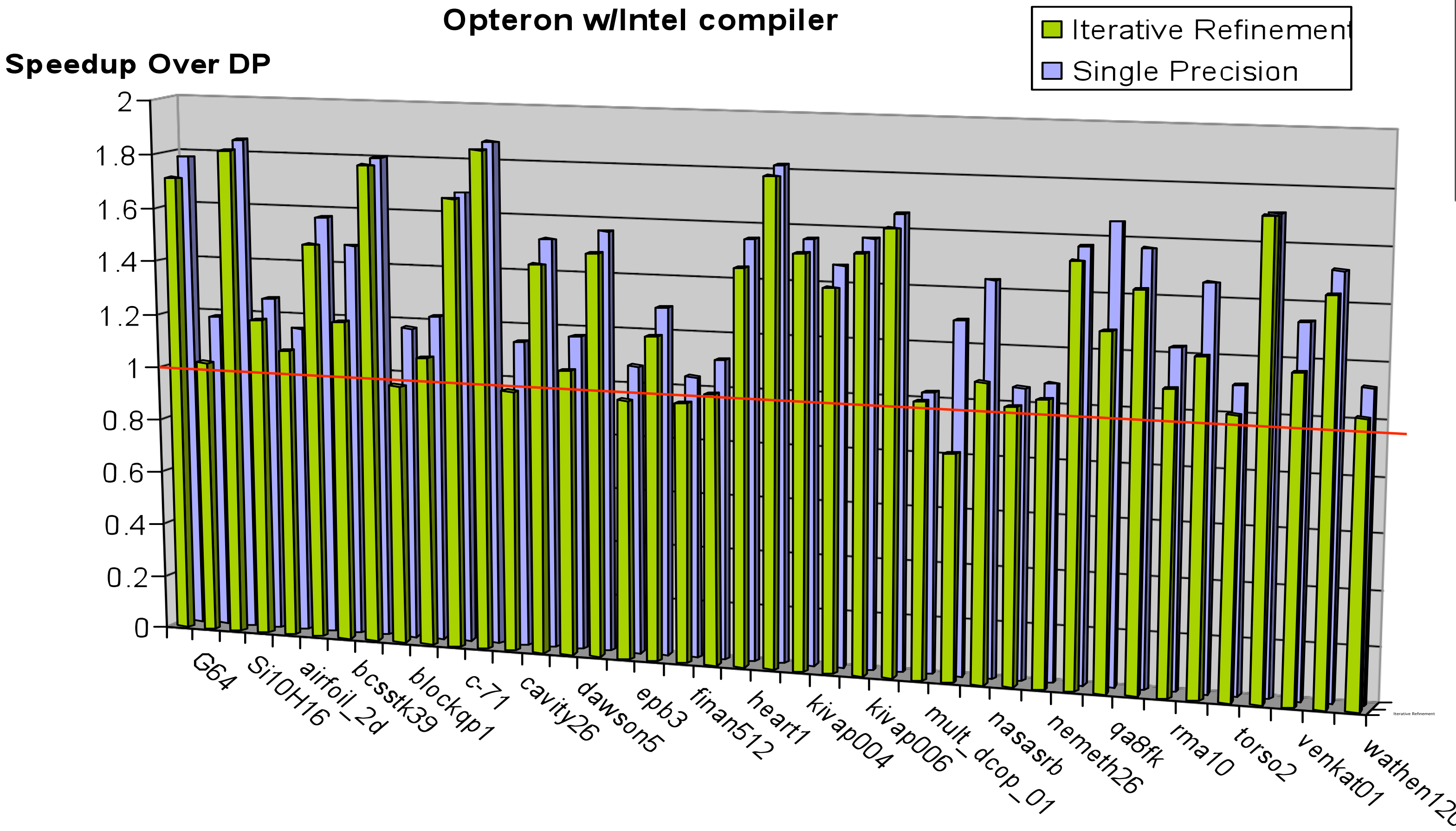
	Architecture (BLAS)
1	Intel Pentium III Coppermine (Goto)
2	Intel Pentium III Katmai (Goto)
3	Sun UltraSPARC IIe (Sunperf)
4	Intel Pentium IV Prescott (Goto)
5	Intel Pentium IV-M Northwood (Goto)
6	AMD Opteron (Goto)
7	Cray X1 (libsci)
8	IBM Power PC G5 (2.7 GHz)(VecLib)
9	Compaq Alpha EV6 (CXML)
10	IBM SP Power3 (ESSL)
11	SGI Octane (ATLAS)

Architecture (BLAS-MPI)	# procs	*n*	DP Solve /SP Solve	DP Solve /Iter Ref	# iter
AMD Opteron (Goto – OpenMPI MX)	32	22627	1.85	1.79	6
AMD Opteron (Goto – OpenMPI MX)	64	32000	1.90	1.83	6

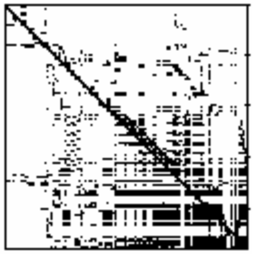
- Single precision is faster than DP because:
  - Higher parallelism within vector units
    - 4 ops/cycle (usually) instead of 2 ops/cycle
  - Reduced data motion
    - 32 bit data instead of 64 bit data
  - Higher locality in cache
    - More data items in cache

Sparse Direct Solver and Iterative Refinement

MUMPS package based on multifrontal approach which generates small dense matrix multiplies



Tim Davis's Collection, n=100K - 3M

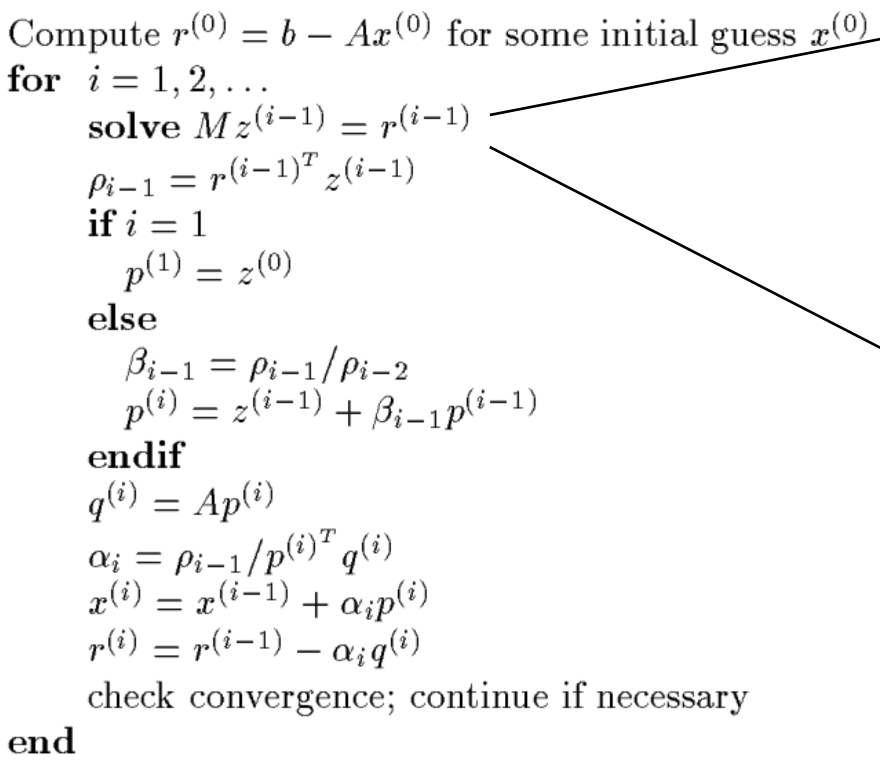


Sparse Iterative Methods (PCG)

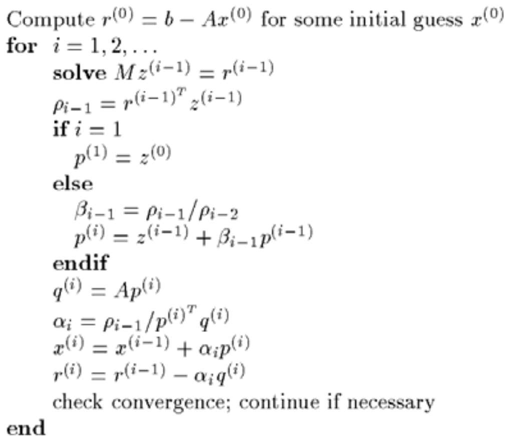
- **Outer/Inner Iteration**

Outer iterations using 64 bit floating point

Inner iteration:
In 32 bit floating point



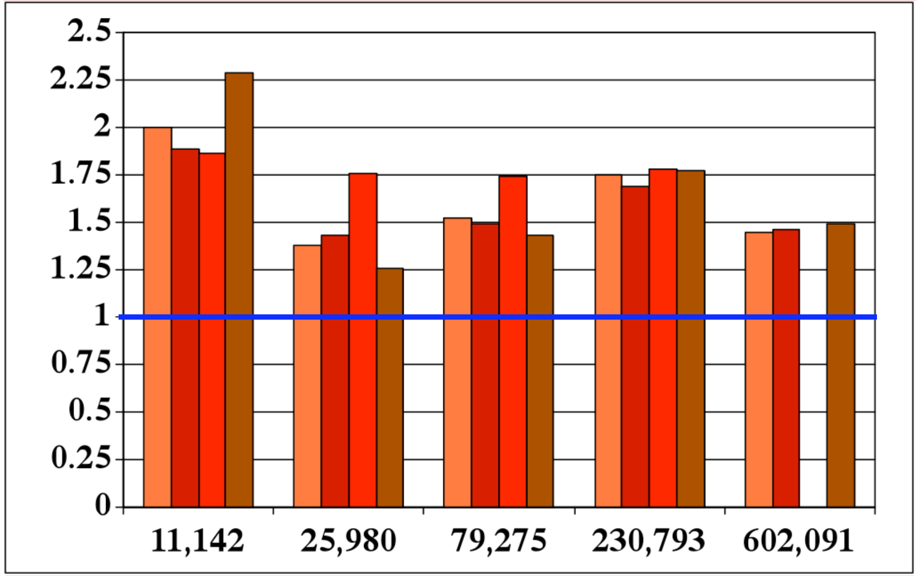
```
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
    solve $Mz^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
    if $i = 1$
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence; continue if necessary
end
```



```
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
    solve $Mz^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
    if $i = 1$
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence; continue if necessary
end
```

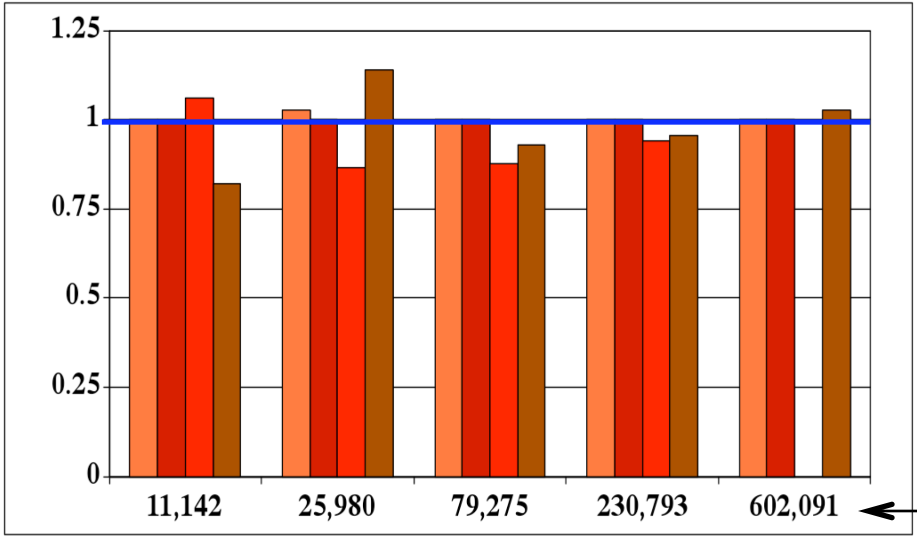
- **Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**

Mixed Precision Computations for Sparse Inner/Outer-type Iterative Solvers



Speedups for mixed precision
Inner SP/Outer DP (SP/DP) iter. methods *vs* DP/DP
(CG^2, GMRES^2, PCG^2, and PGMRES^2 with diagonal prec.)
(***Higher is better***)

🟧 CG^2
🟥 PCG^2
🟥 GMRES^2
🟫 PGMRES^2



Iterations for mixed precision
SP/DP iterative methods *vs* DP/DP
(***Lower is better***)

Machine:
Intel Woodcrest (3GHz, 1333MHz bus)

Stopping criteria:
Relative to r_0 residual reduction (10^{-12})

← Matrix size

6,021 18,000 39,000 120,000 240,000 ← Condition number

Intriguing Potential

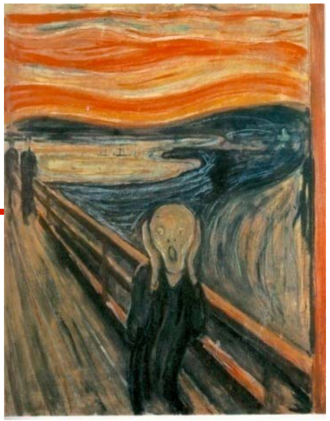
- **Exploit lower precision as much as possible**
  - **Payoff in performance**
    - **Faster floating point**
    - **Less data to move**
- **Automatically switch between SP and DP to match the desired accuracy**
  - **Compute solution in SP and then a correction to the solution in DP**
- **Potential for GPU, FPGA, special purpose processors**
  - **Use as little you can get away with and improve the accuracy**
- **Applies to sparse direct and iterative linear systems and Eigenvalue, optimization problems, where Newton's method is used.**

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$\boxed{x_{i+1} - x_i} = -\frac{f(x_i)}{f'(x_i)}$$

Correction = - A\(b – Ax)

Fault Tolerance



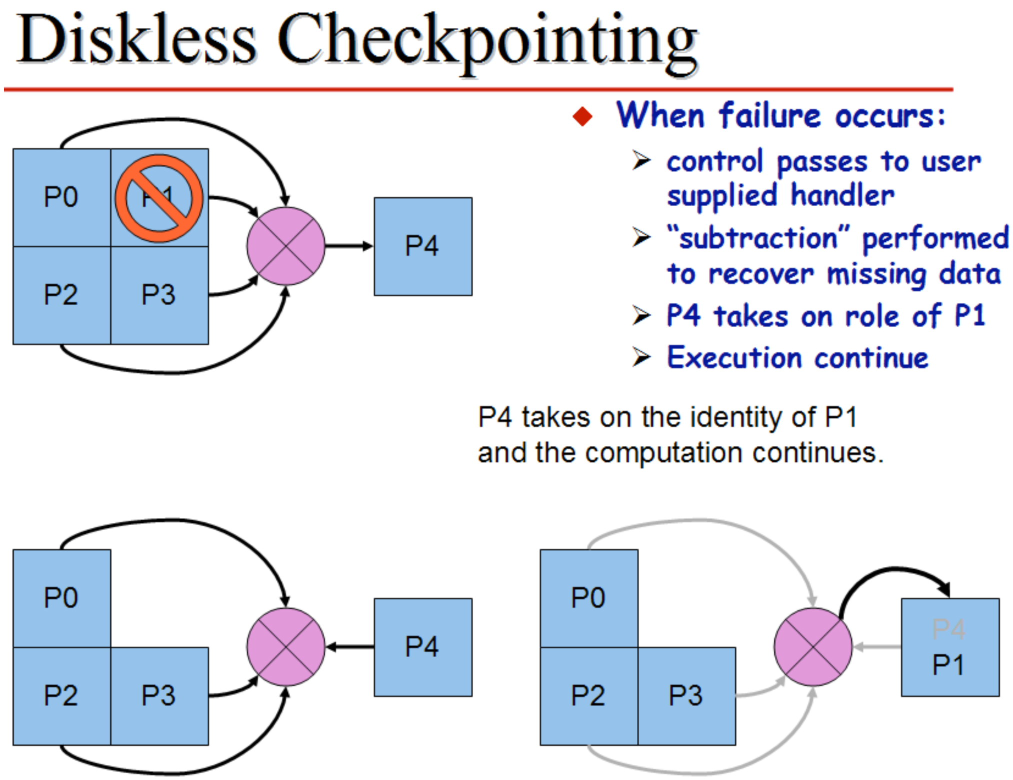
- Trends in HPC:
  - High end systems with thousand of processors.
- Increased probability of a system. failure

  - Most nodes today are robust, 3 year life.
  - Mean Time to Failure is growing shorter as systems grow and devices shrink.
- MPI widely accepted in scientific computing.
  - Process faults not tolerated in MPI model.

Mismatch between hardware and (non fault-tolerant) programming paradigm of MPI.

Three Ideas for Fault Tolerant Linear Algebra Algorithms

- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data

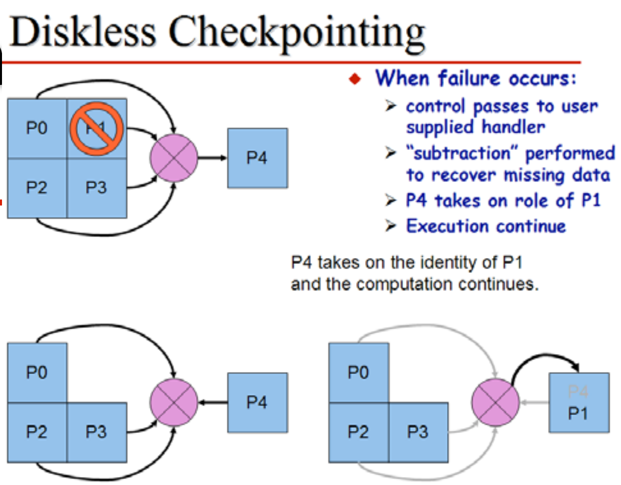


Diskless Checkpointing

- When failure occurs:
  - control passes to user supplied handler
  - "subtraction" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1 and the computation continues.

Three Ideas for Fault Tolera
Linear Algebra Algorithms

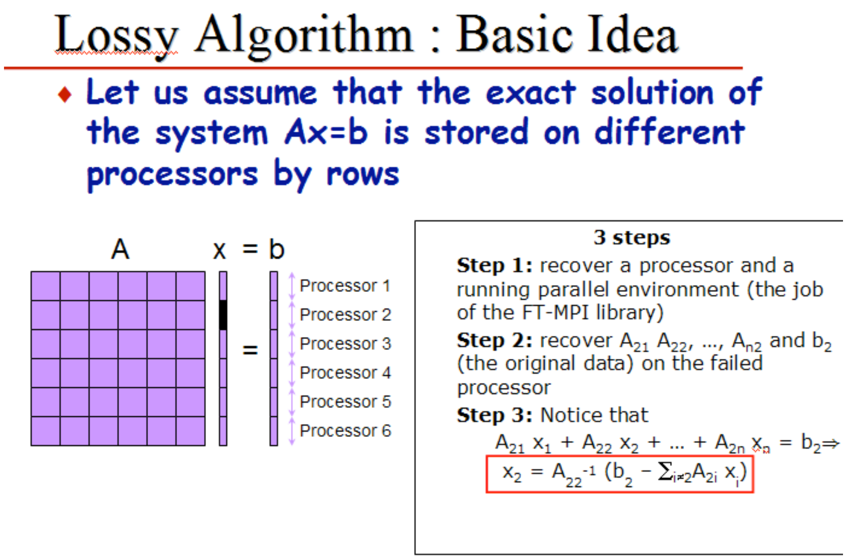


Diskless Checkpointing

- When failure occurs:
  - control passes to user supplied handler
  - "subtraction" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1 and the computation continues.

- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- Lossy approach for iterative methods
  - No checkpoint for computed data maintained
  - On failure, approximate missing data and carry on
  - Lost data but use approximation to recover



Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows

3 steps

Step 1: recover a processor and a running parallel environment (the job of the FT-MPI library)

Step 2: recover A_{21} A_{22}, …, A_{n2} and b_2 (the original data) on the failed processor

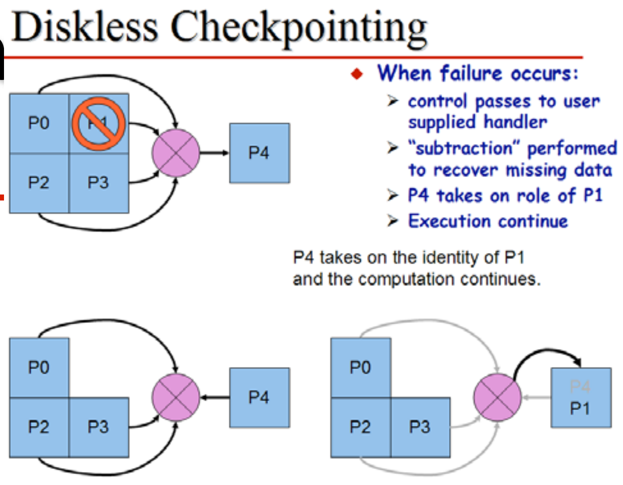
Step 3: Notice that

$$A_{21} x_1 + A_{22} x_2 + \ldots + A_{2n} x_n = b_2 \Rightarrow$$

$$\boxed{x_2 = A_{22}^{-1} (b_2 - \Sigma_{i \neq 2} A_{2i} x_i)}$$

Three Ideas for Fault Tolera Linear Algebra Algorithms

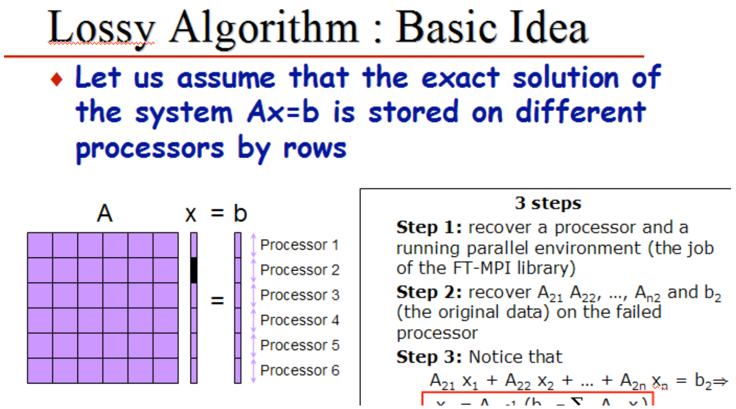
- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- Lossy approach for iterative methods
  - No checkpoint maintained
  - On failure, approximate missing data and carry on
  - Lost data but use approximation to recover
- **Check-pointless methods for dense algorithms**
  - **Checksum maintained as part of computation**
  - **No roll back needed; No lost data**



Diskless Checkpointing

- When failure occurs:
  - control passes to user supplied handler
  - "subtraction" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1 and the computation continues.



Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows

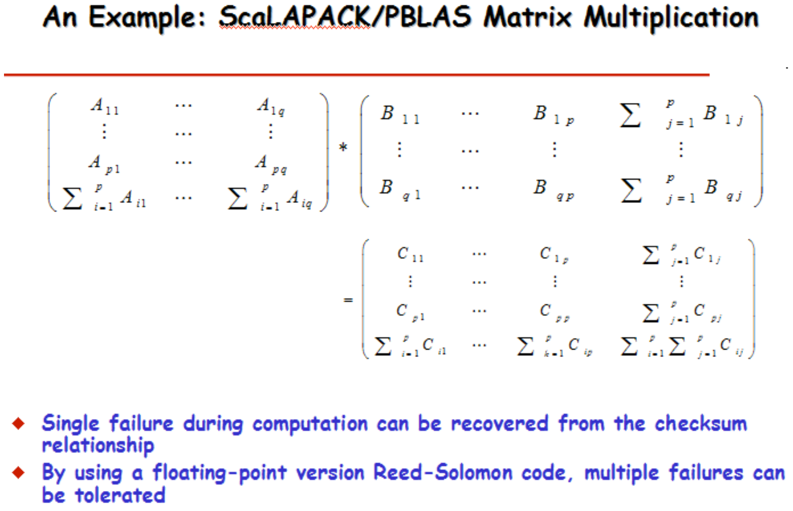
3 steps

Step 1: recover a processor and a running parallel environment (the job of the FT-MPI library)

Step 2: recover A_{21} A_{22}, …, A_{n2} and b_2 (the original data) on the failed processor

Step 3: Notice that

$A_{21}\, x_1 + A_{22}\, x_2 + \ldots + A_{2n}\, x_n = b_2 \Rightarrow$



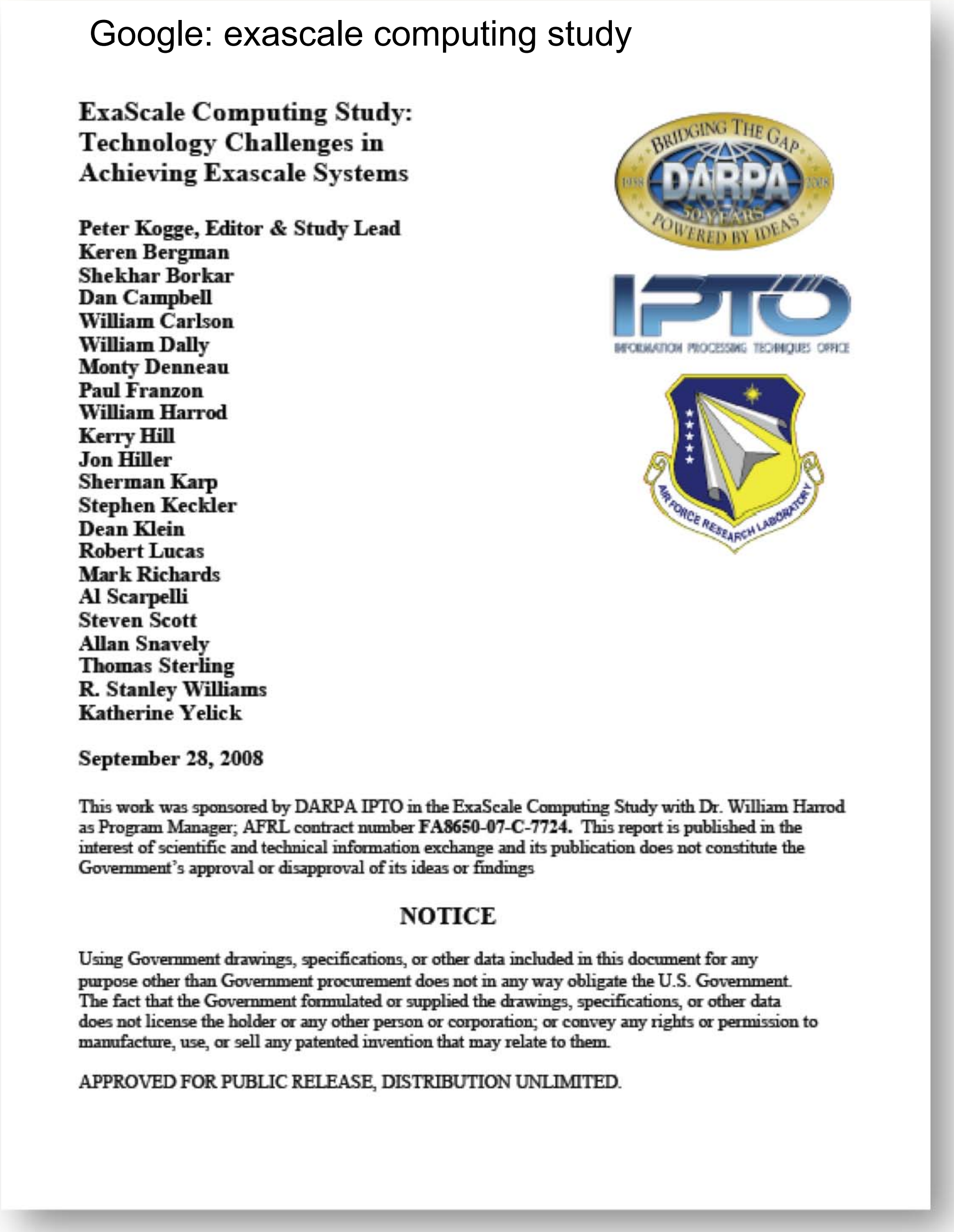
An Example: ScaLAPACK/PBLAS Matrix Multiplication

$$\begin{pmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & \cdots & \vdots \\ A_{p1} & \cdots & A_{pq} \\ \sum_{i=1}^{p} A_{i1} & \cdots & \sum_{i=1}^{p} A_{iq} \end{pmatrix} * \begin{pmatrix} B_{11} & \cdots & B_{1p} & \sum_{j=1}^{p} B_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ B_{q1} & \cdots & B_{qp} & \sum_{j=1}^{p} B_{qj} \end{pmatrix}$$

$$= \begin{pmatrix} C_{11} & \cdots & C_{1p} & \sum_{j=1}^{p} C_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ C_{p1} & \cdots & C_{pp} & \sum_{j=1}^{p} C_{pj} \\ \sum_{i=1}^{p} C_{i1} & \cdots & \sum_{k=1}^{p} C_{ip} & \sum_{i=1}^{p} \sum_{j=1}^{p} C_{ij} \end{pmatrix}$$

- Single failure during computation can be recovered from the checksum relationship
- By using a floating-point version Reed-Solomon code, multiple failures can be tolerated

Exascale Computing



Google: exascale computing study

ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

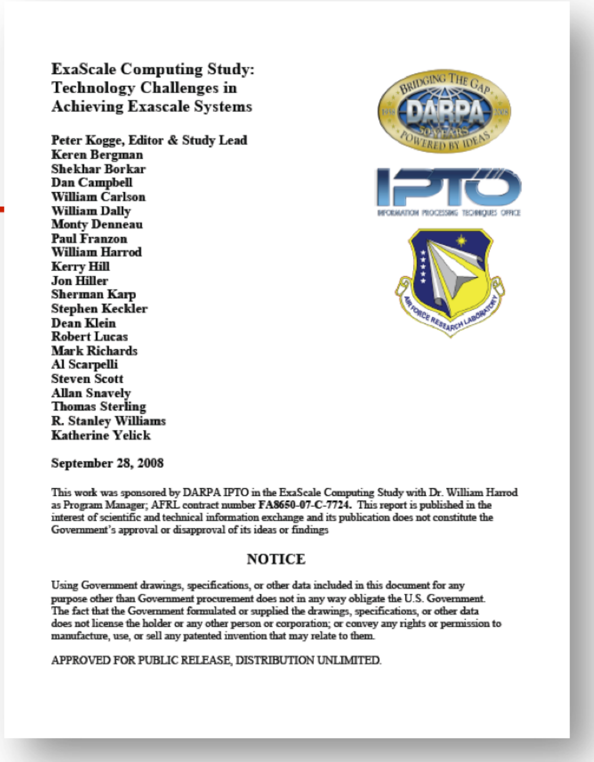
This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

Exascale Computing



ExaScale Computing Study:
Technology Challenges in
Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

- **Exascale systems are likely feasible by 2017±2**
- **10-100 Million processing elements (cores or mini-cores) with chips perhaps as dense as 1,000 cores per socket, clock rates will grow more slowly**
- **3D packaging likely**
- **Large-scale optics based interconnects**
- **10-100 PB of aggregate memory**
- **Hardware and software based fault management**
- **Heterogeneous cores**
- **Performance per watt — stretch goal 100 GF/watt of sustained performance ⇒ >> 10 - 100 MW Exascale system**
- **Power, area and capital costs will be significantly higher than for today's fastest systems**

Google: exascale computing study

Conclusions

- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
    - Hardware has a half-life measured in years, while software has a half-life measured in decades.
- High Performance Ecosystem out of balance
    - Hardware, OS, Compilers, Software, Algorithms, Applications
        - No Moore's Law for software, algorithms and applications

Collaborators / Support

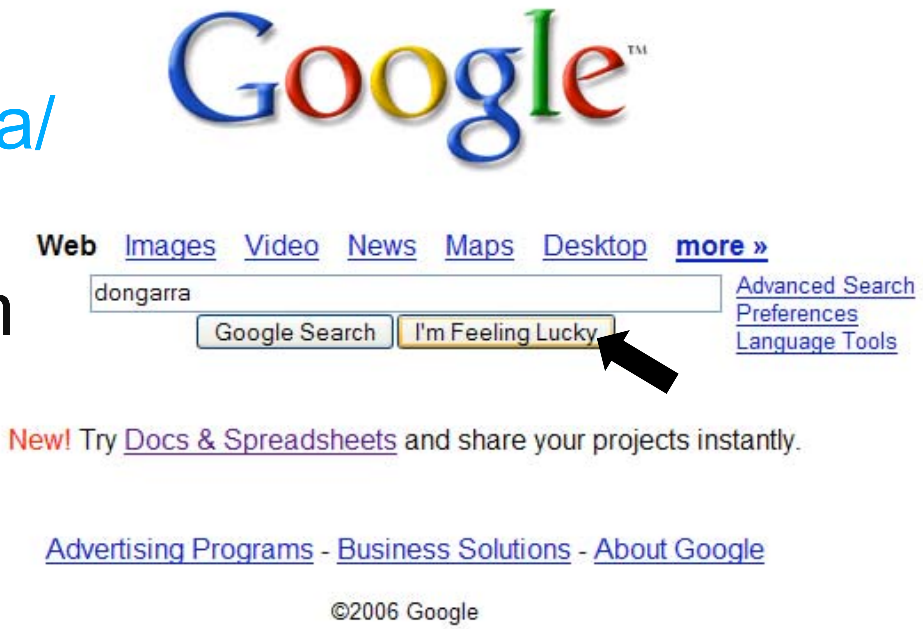
Employment opportunities for post-docs in the ICL group at Tennessee

Microsoft

The MathWorks

PLASMA Parallel Linear Algebra Software for Multicore Architectures

http://icl.cs.utk.edu/plasma/



dongarra

New! Try Docs & Spreadsheets and share your projects instantly.

Advertising Programs - Business Solutions - About Google

©2006 Google

MAGMA Matrix Algebra on GPU and Multicore Architectures

Contact Jack Dongarra

If you are wondering what's beyond ExaFlops

Mega, Giga, Tera, Peta, Exa, Zetta …

10^{3}	kilo
10^{6}	mega
10^{9}	giga
10^{12}	tera
10^{15}	peta
10^{18}	exa
10^{21}	zetta

10^{24}	yotta
10^{27}	xona
10^{30}	weka
10^{33}	vunda
10^{36}	uda
10^{39}	treda
10^{42}	sorta
10^{45}	rinta
10^{48}	quexa
10^{51}	pepta
10^{54}	ocha
10^{57}	nena
10^{60}	minga
10^{63}	luma