# An Overview of High Performance Computing and Challenges for the Future

**Jack Dongarra**

**INNOVATIVE COMPUTING LABORATORY**

**University of Tennessee**
**Oak Ridge National Laboratory**
**University of Manchester**

2008 SIAM ANNUAL MEETING
July 7-11, 2008
Town and Country Resort
& Convention Center
San Diego, CA

# Overview

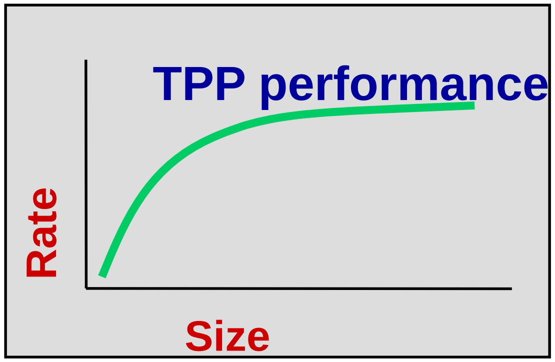
- Quick look at High Performance Computing
  - Top500
- Challenges for Math Software
  - Linear Algebra Software for Multicore and Beyond

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*



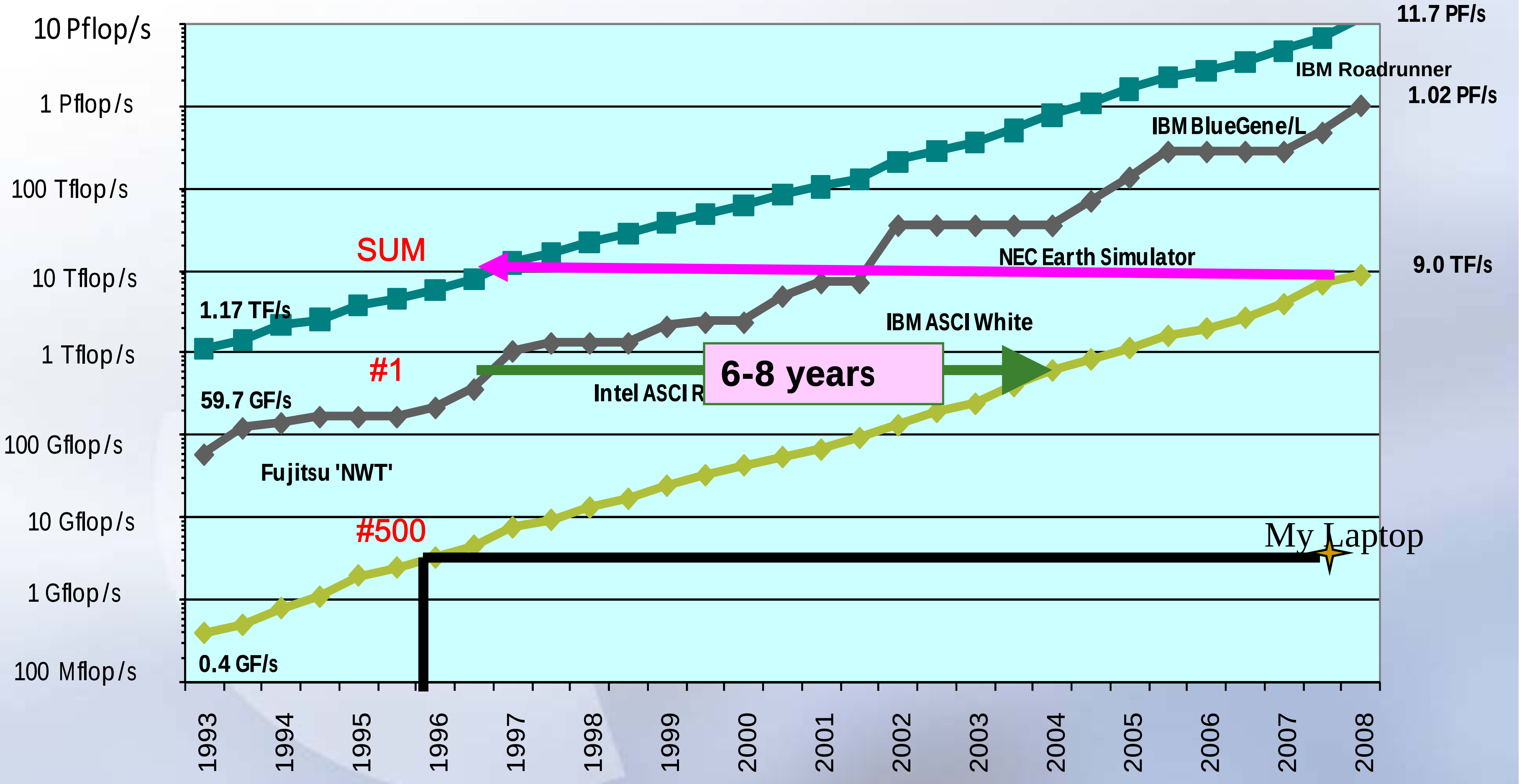
- Updated twice a year

  SC'xy in the States in November

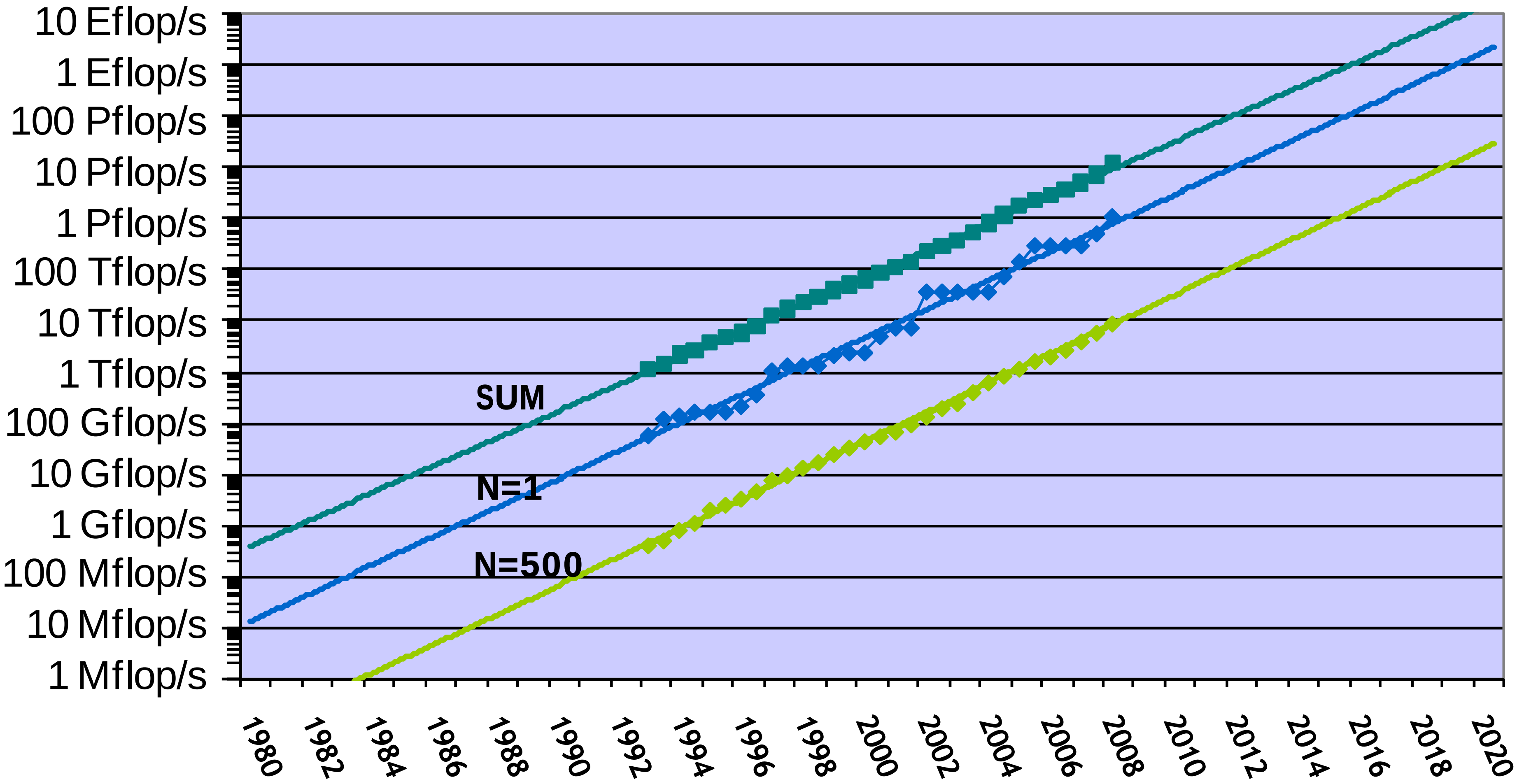
  Meeting in Germany in June

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

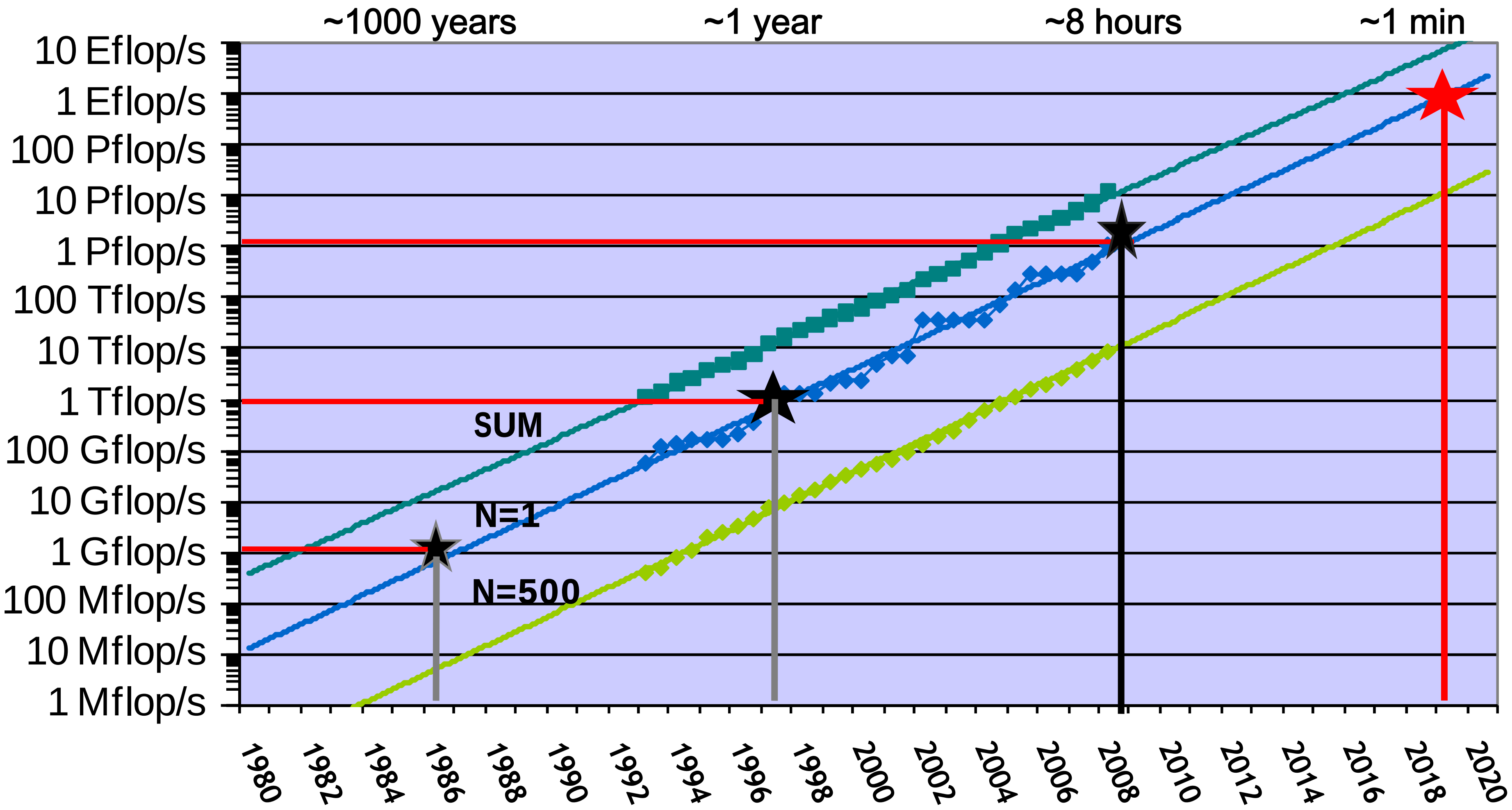
# Performance Development



# Performance Development & Projections



# Performance Development & Projections



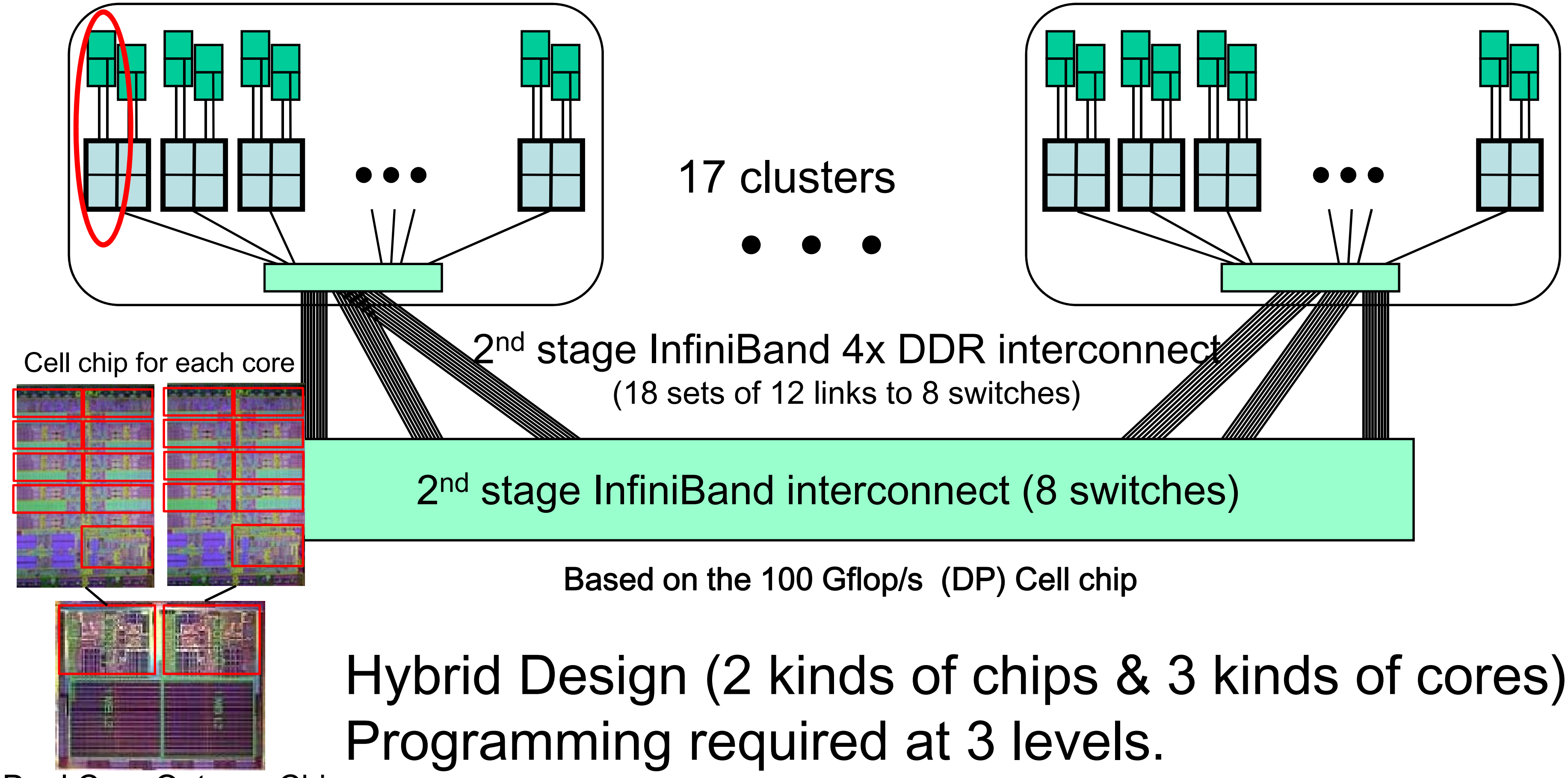
| | | | |
|---|---|---|---|
| Cray 2 | ASCI Red | Roadrunner | |
| 1 Gflop/s | 1 Tflop/s | 1 Pflop/s | 1 Eflop/s |
| O(1) Thread | O($10^3$) Threads | O($10^6$)Threads | O($10^9$) Threads |

# LANL Roadrunner
# A Petascale System in 2008

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

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

≈ 7,000 dual-core Opterons
**≈ 122,000 cores**


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.

# Top10 of the June 2008 List

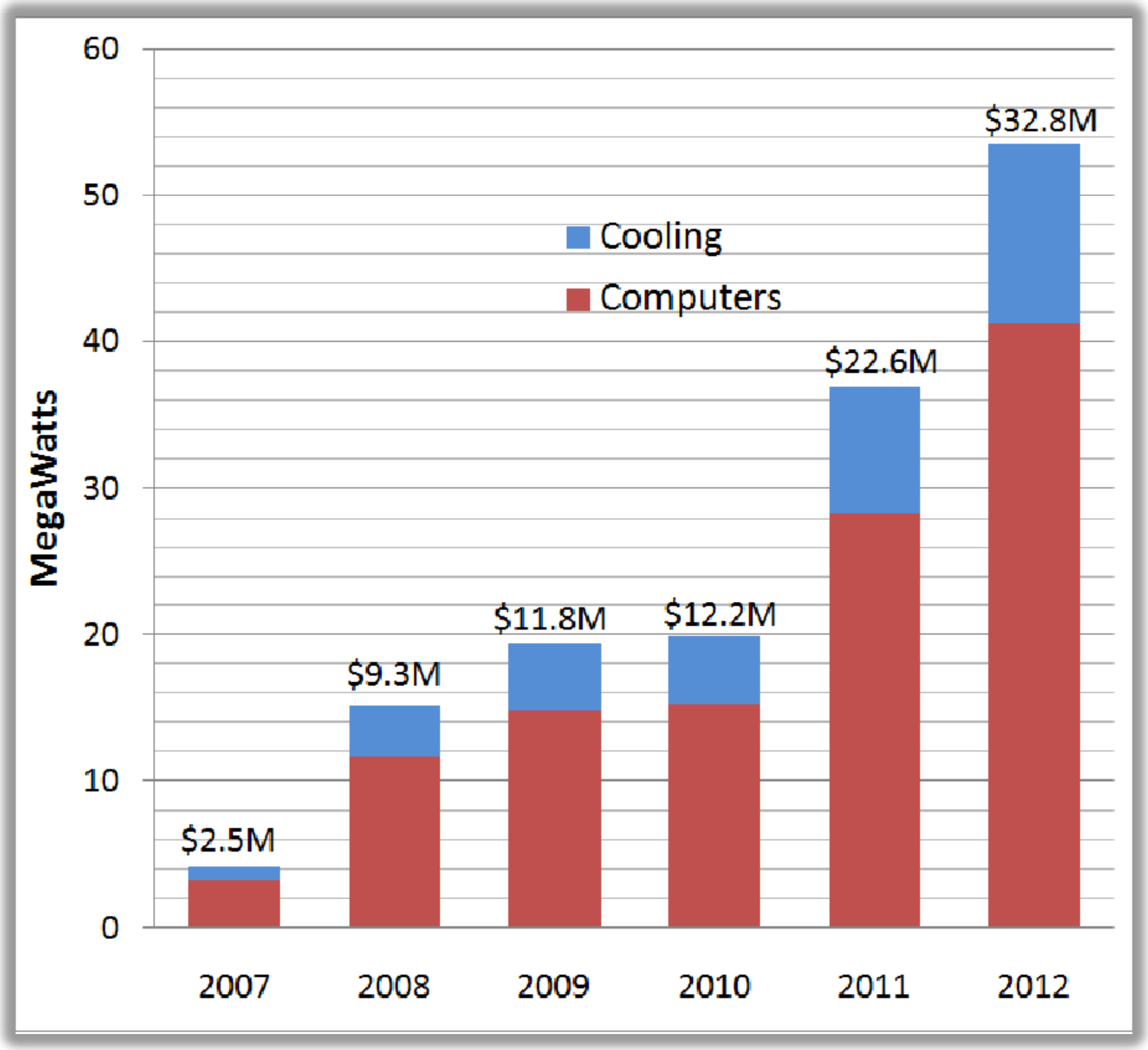
| | Computer | Rmax<br>[TF/s] | Rmax /<br>Rpeak | Installation Site | Country | #Cores |
|---|---|---|---|---|---|---|
| 1 | IBM / Roadrunner<br>BladeCenter QS22/LS21 | 1,026 | 75% | DOE/NNSA/LANL | USA | 122,400 |
| 2 | IBM / BlueGene/L<br>eServer Blue Gene Solution | 478 | 80% | DOE/NNSA/LLNL | USA | 212,992 |
| 3 | IBM / Intrepid<br>Blue Gene/P Solution | 450 | 81% | DOE/OS/ANL | USA | 163,840 |
| 4 | SUN / Ranger<br>SunBlade x6420 | 326 | 65% | NSF/TACC | USA | 62,976 |
| 5 | CRAY / Jaguar<br>Cray XT4 QuadCore | 205 | 79% | DOE/OS/ORNL | USA | 30,976 |
| 6 | IBM / JUGENE<br>Blue Gene/P Solution | 180 | 81% | Forschungszentrum Juelich (FZJ) | Germany | 65,536 |
| 7 | SGI / Encanto<br>SGI Altix ICE 8200 | 133.2 | 77% | New Mexico Computing Applications Center | USA | 14,336 |
| 8 | HP / EKA<br>Cluster Platform 3000 BL460c | 132.8 | 77% | Computational Research Lab, TATA SONS | India | 14,384 |
| 9 | IBM / Blue Gene/P Solution | 112 | 81% | IDRIS | France | 40,960 |
| 10 | SGI / Altix ICE 8200EX | 106 | 86% | Total Exploration Production | France | 10,240 |

# Top10 of the June 2008 List

| | Computer | Rmax [TF/s] | Rmax / Rpeak | Installation Site | Country | #Cores | Power [MW] | **MFlops/ Watt** |
|---|---|---|---|---|---|---|---|---|
| 1 | IBM / Roadrunner<br>BladeCenter QS22/LS21 | 1,026 | 75% | DOE/NNSA/LANL | USA | 122,400 | 2.35 | 437 |
| 2 | IBM / BlueGene/L<br>eServer Blue Gene Solution | 478 | 80% | DOE/NNSA/LLNL | USA | 212,992 | 2.33 | 205 |
| 3 | IBM / Intrepid<br>Blue Gene/P Solution | 450 | 81% | DOE/OS/ANL | USA | 163,840 | 1.26 | 357 |
| 4 | SUN / Ranger<br>SunBlade x6420 | 326 | 65% | NSF/TACC | USA | 62,976 | 2.00 | 163 |
| 5 | CRAY / Jaguar<br>Cray XT4 QuadCore | 205 | 79% | DOE/OS/ORNL | USA | 30,976 | 1.58 | 130 |
| 6 | IBM / JUGENE<br>Blue Gene/P Solution | 180 | 81% | Forschungszentrum Juelich (FZJ) | Germany | 65,536 | 0.50 | 357 |
| 7 | SGI / Encanto<br>SGI Altix ICE 8200 | 133.2 | 77% | New Mexico Computing Applications Center | USA | 14,336 | 0.86 | 155 |
| 8 | HP / EKA<br>Cluster Platform 3000 BL460c | 132.8 | 77% | Computational Research Lab, TATA SONS | India | 14,384 | 0.79 | 169 |
| 9 | IBM / Blue Gene/P Solution | 112 | 81% | IDRIS | France | 40,960 | 0.32 | 357 |
| 10 | SGI / Altix ICE 8200EX | 106 | 86% | Total Exploration Production | France | 10,240 | 0.44 | 240 |

# ORNL/UTK Computer Power Cost Projections 2007-2012

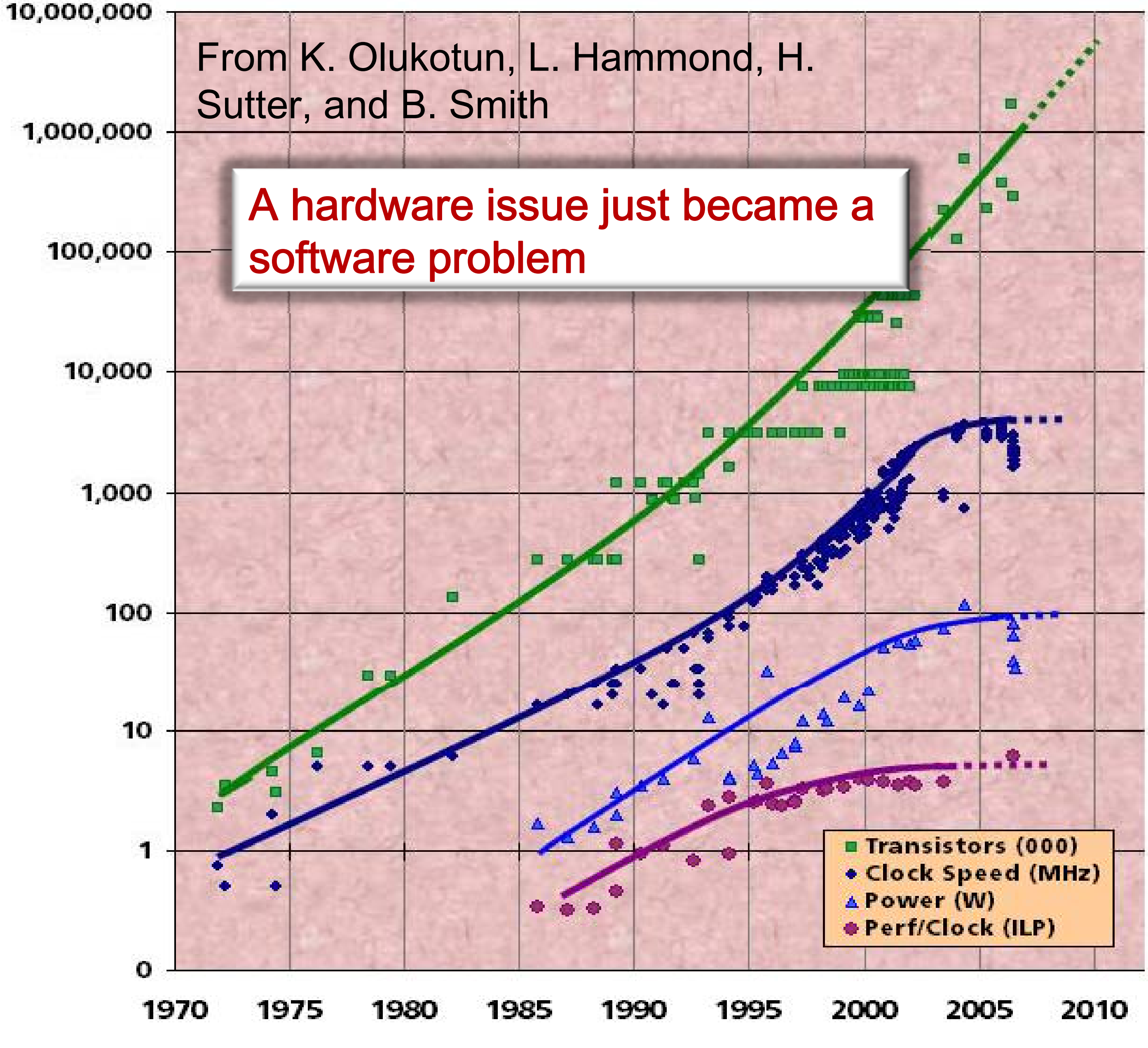
- Over the next 5 years ORNL/UTK will deploy 2 large Petascale systems
- Using 4 MW today, going to 15MW before year end
- By 2012 could be using more than 50MW!!
- Cost estimates based on $0.07 per KwH



**Power becomes the architectural driver for future large systems**

Cost Per Year
Includes both DOE and NSF systems.

# Something's Happening Here…

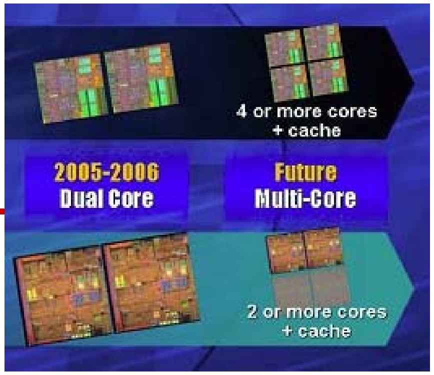


From K. Olukotun, L. Hammond, H. Sutter, and B. Smith

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

# Multicore



- **What is multicore?**
  - **A multicore chip is a single chip (socket) that combines two or more independent processing units that provide independent threads of control**
- **Why multicore?**
  - **The race for ever higher clock speeds is over.**
    - **In the old days, new the chips where faster**
      - Applications ran faster on the new chips
    - **Today new chips are not faster, just have more processors per chip**
      - Applications and software must use those extra processors to become faster

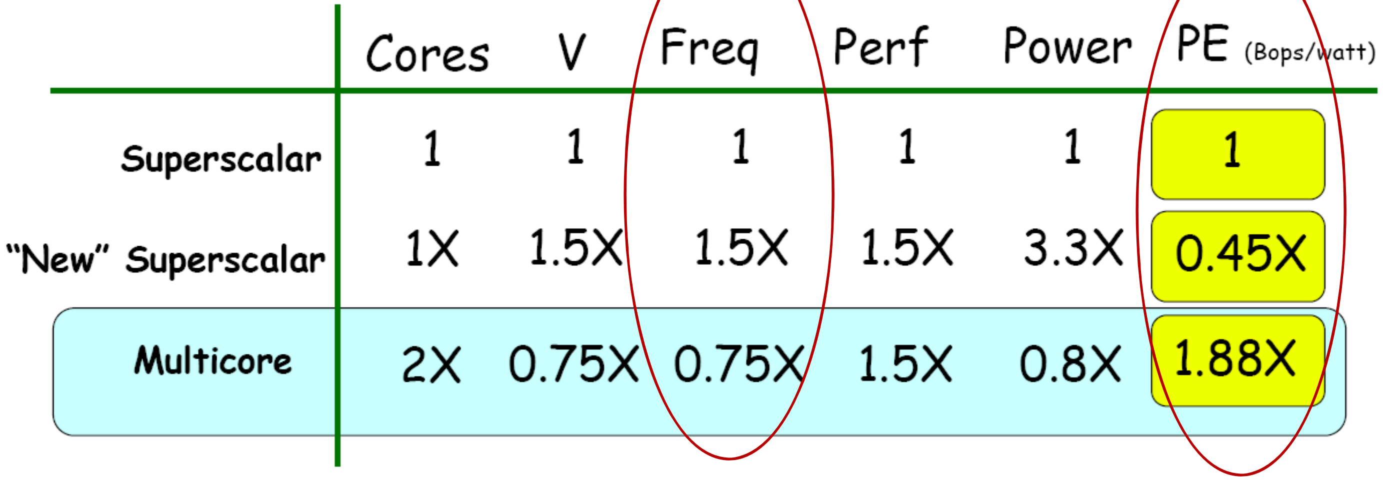
# Power Cost of Frequency

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

| | 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 ∝ Voltage$^2$ x Frequency (V$^2$F)
- 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 |
| Multicore | 2X | 0.75X | 0.75X | 1.5X | 0.8X | 1.88X |

(Bigger # is better)

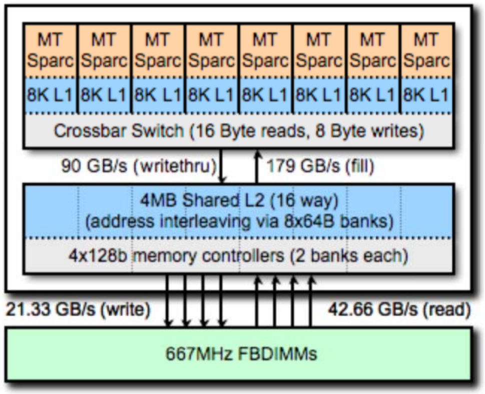
50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

# Today’s Multicores

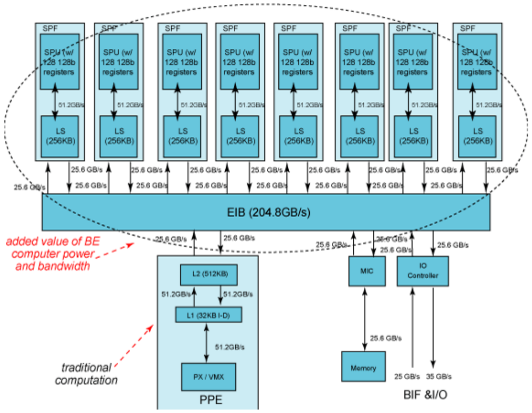
## 98% of Top500 Systems Are Based on Multicore

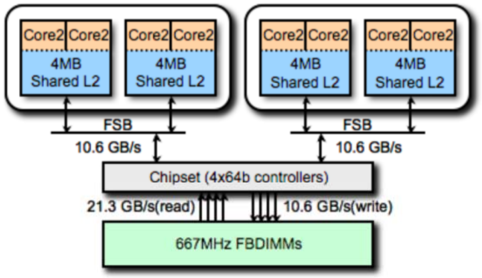
282 use Quad-Core
204 use Dual-Core
3 use Nona-core


MT Sparc
8K L1
Crossbar Switch (16 Byte reads, 8 Byte writes)
90 GB/s (writethru)
179 GB/s (fill)
4MB Shared L2 (16 way)
(address interleaving via 8x64B banks)
4x128b memory controllers (2 banks each)
21.33 GB/s (write)
42.66 GB/s (read)
667MHz FBDIMMs

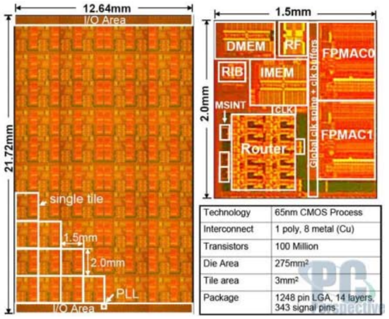
Sun Niagra2 (8 cores)

IBM Cell (9 cores)



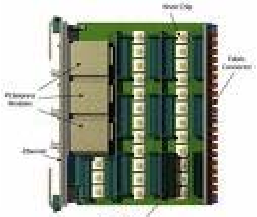


Intel Clovertown (4 cores)

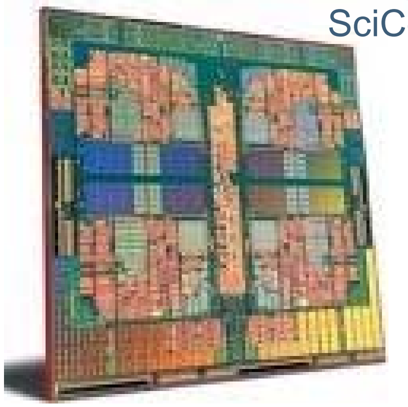


| | |
|---|---|
| Technology | 65nm CMOS Process |
| Interconnect | 1 poly, 8 metal (Cu) |
| Transistors | 100 Million |
| Die Area | 275mm² |
| Tile area | 3mm² |
| Package | 1248 pin LGA, 14 layers, 343 signal pins |

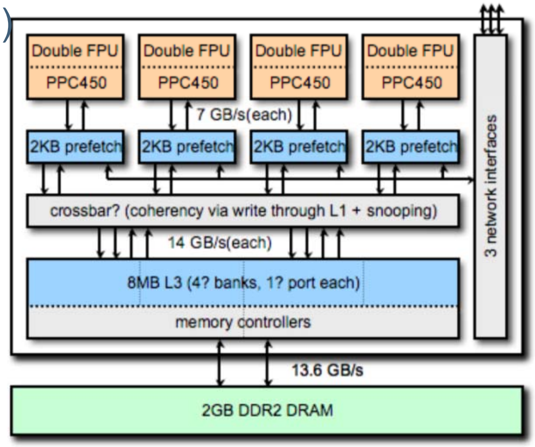
Intel Polaris (80 cores)



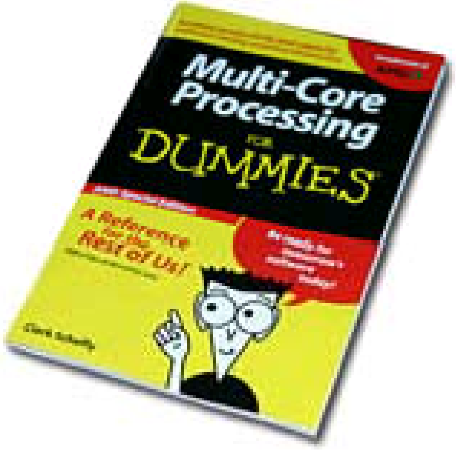
SciCortex (6 cores)



AMD Opteron (4 cores)

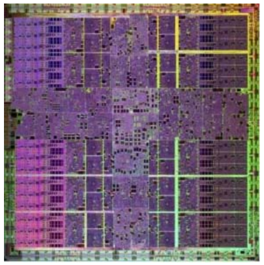


IBM BG/P (4 cores)



# And then there's the GPU's NVIDIA's Tesla T10P

- **T10P chip**
  - **240 cores; 1.5 GHz**
  - **Tpeak 1 Tflop/s - 32 bit floating point**
  - **Tpeak 100 Gflop/s - 64 bit floating point**

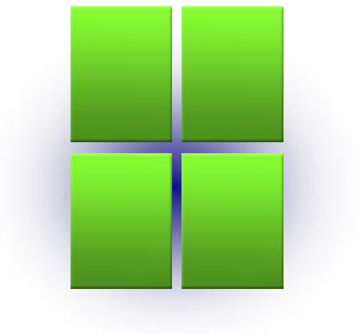


- **S1070 board**
  - **4 - T10P devices;**
  - **700 Watts**

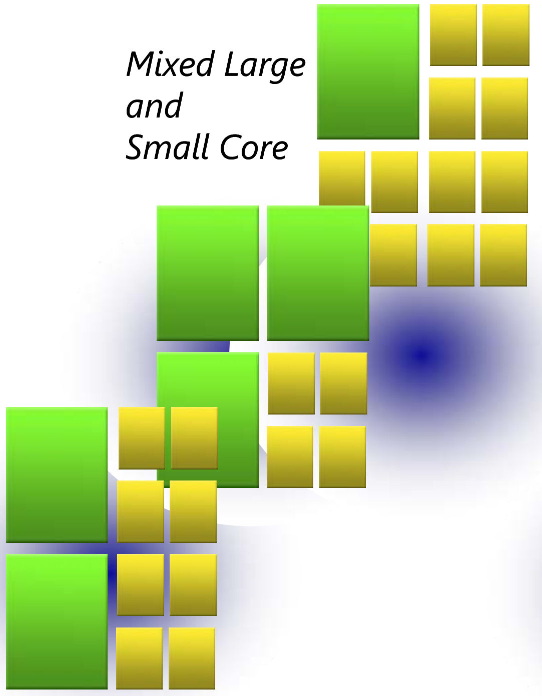
- **C1060 card**
  - **1 – T10P; 1.33 GHz**
  - **160 Watts**
  - **Tpeak 887 Gflop/s - 32 bit floating point**
  - **Tpeak 88.7 Gflop/s - 64 bit floating point**

# What's Next? Multicore to Manycore

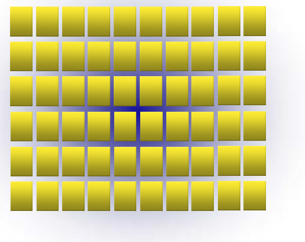
*All Large Core*



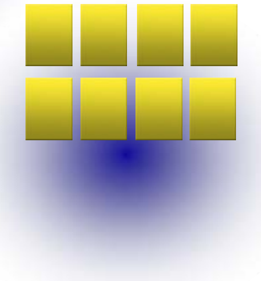
*Mixed Large and Small Core*



*Many Small Cores*

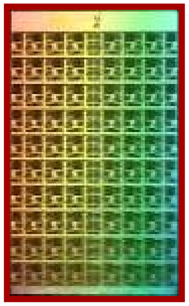


*All Small Core*

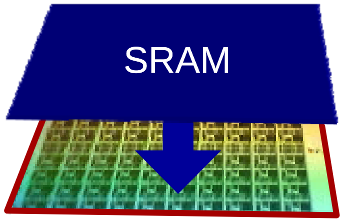


Different Classes of Chips
- Home
- Games / Graphics
- Business
- Scientific

Many Floating-Point Cores



+ 3D Stacked Memory



The question is not whether this will happen but whether we are ready

# 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.

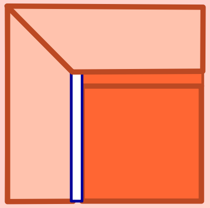
# ManyCore - Parallelism for the Masses

- **We are looking at the following concepts in designing the next numerical library implementation**
  - **Dynamic Data Driven Execution**
  - **Self Adapting**
  - **Block Data Layout**
  - **Mixed Precision in the Algorithm**
  - **Exploit Hybrid Architectures**
  - **Fault Tolerant Methods**

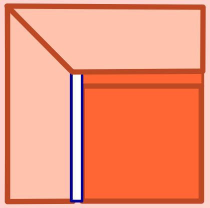
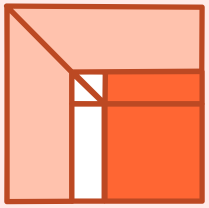
# Major Changes to Software

- **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**

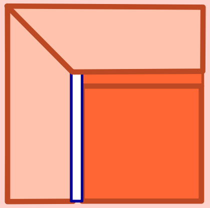
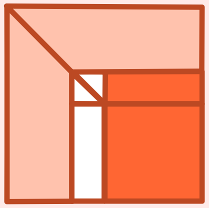
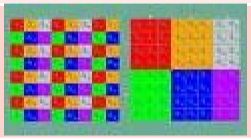
# A New Generation of Software:

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

# A New Generation of Software:

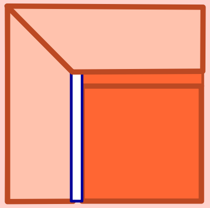
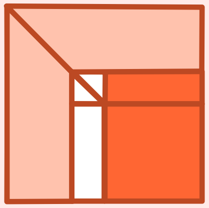
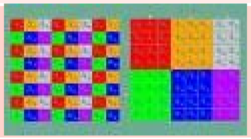
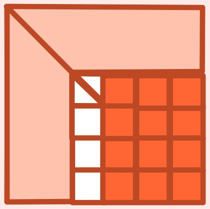
| Software/Algorithms follow hardware evolution in time | | |
|---|---|---|
| LINPACK (70’s)<br>(Vector operations) |  | Rely on<br>- Level-1 BLAS operations |
| LAPACK (80’s)<br>(Blocking, cache friendly) |  | Rely on<br>- Level-3 BLAS operations |

# A New Generation of Software:

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

# A New Generation of Software:

## Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

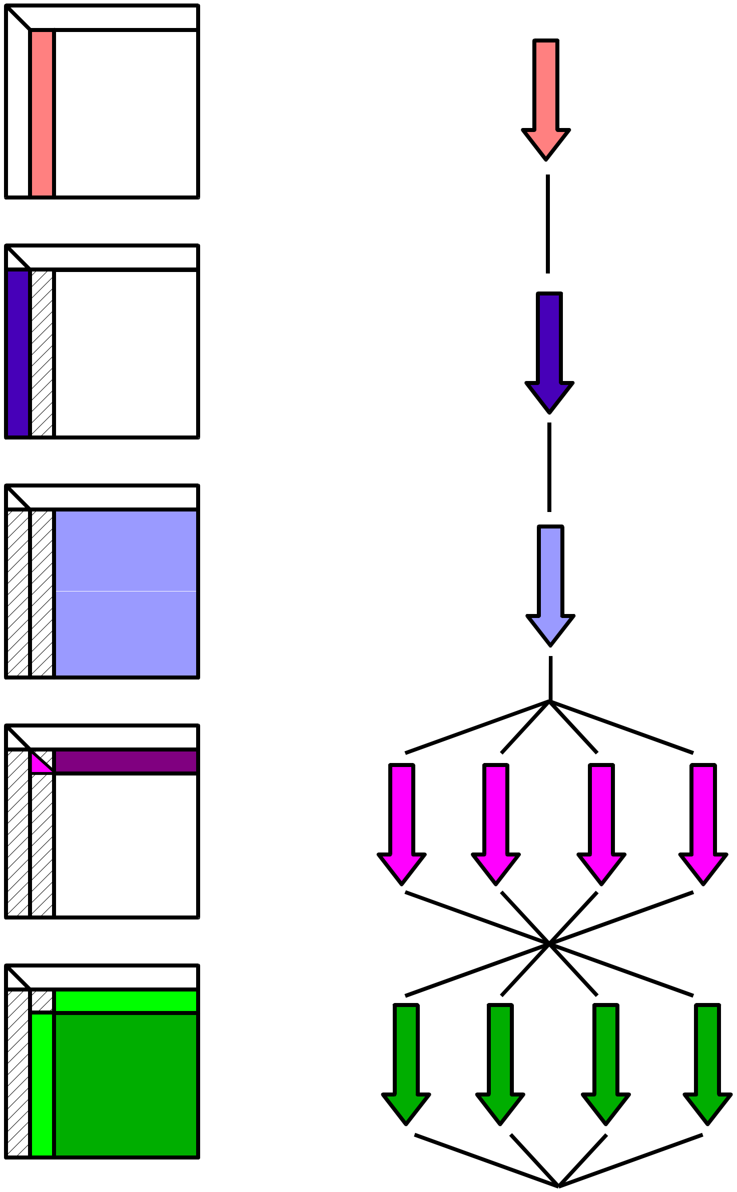
| Software/Algorithms follow hardware evolution in time | | |
|---|---|---|
| LINPACK (70's)<br>(Vector operations) |  | Rely on<br>- Level-1 BLAS operations |
| LAPACK (80's)<br>(Blocking, cache friendly) |  | Rely on<br>- Level-3 BLAS operations |
| ScaLAPACK (90's)<br>(Distributed Memory) |  | Rely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

# Steps in the LAPACK LU



DGETF2
(Factor a panel)

LAPACK

DLSWP
(Backward swap)

LAPACK

DLSWP
(Forward swap)

LAPACK

DTRSM
(Triangular solve)

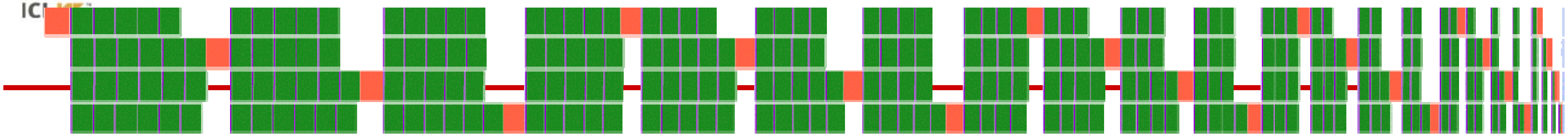
**BLAS**

DGEMM
(Matrix multiply)

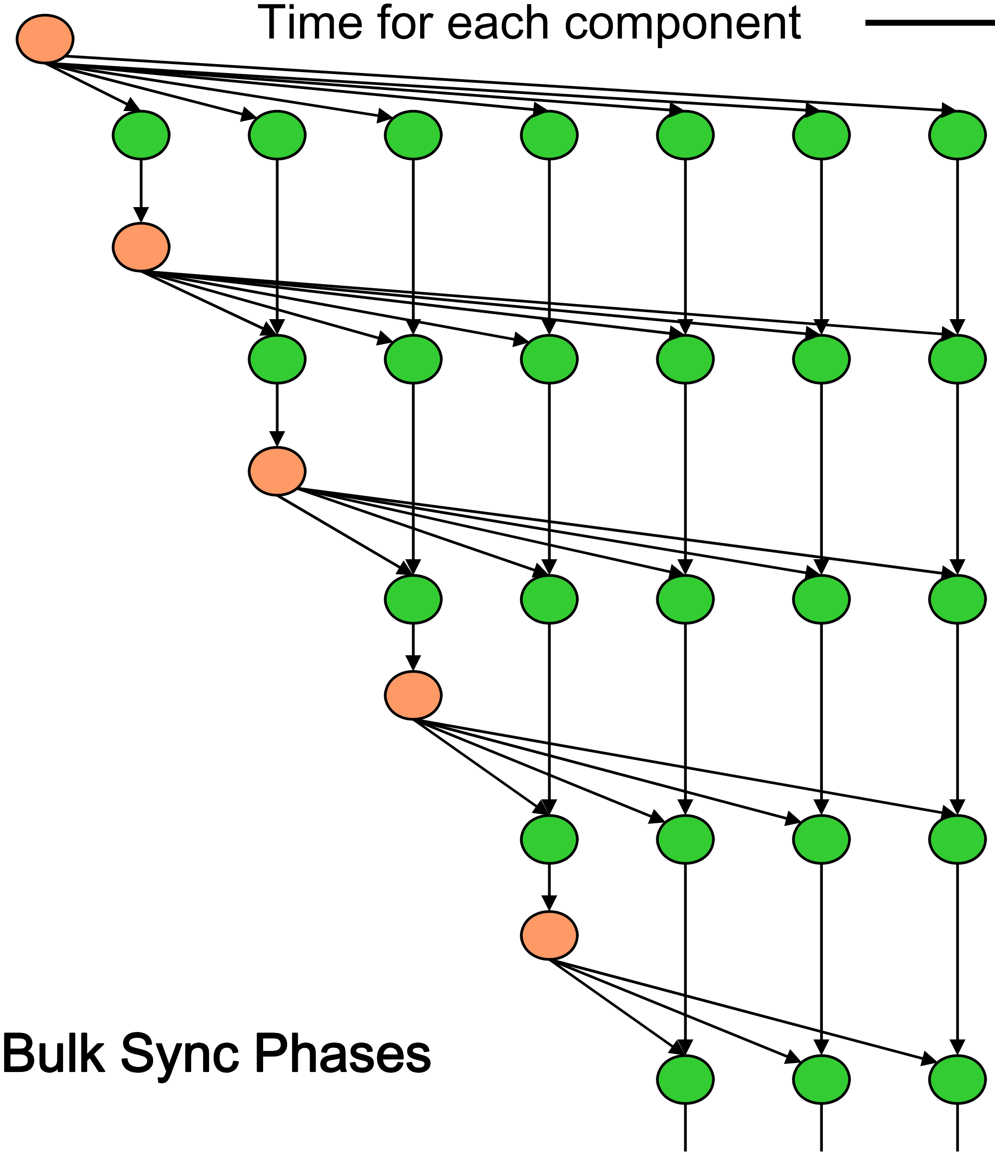
**BLAS**

# LU Timing Profile (4 processor system)

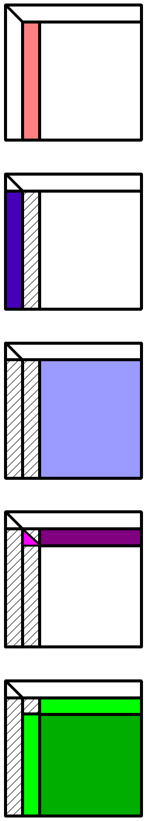
Threads – no lookahead



Time for each component →



- 🟧 DGETF2
- 🟦 DLASWP(L)
- 🟦 DLASWP(R)
- 🟪 DTRSM
- 🟩 DGEMM



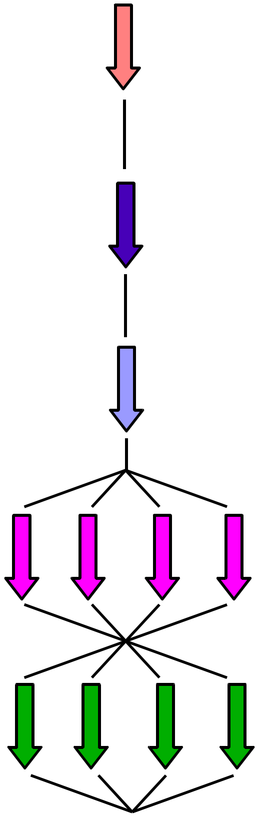
DGETF2

DLSWP

DLSWP

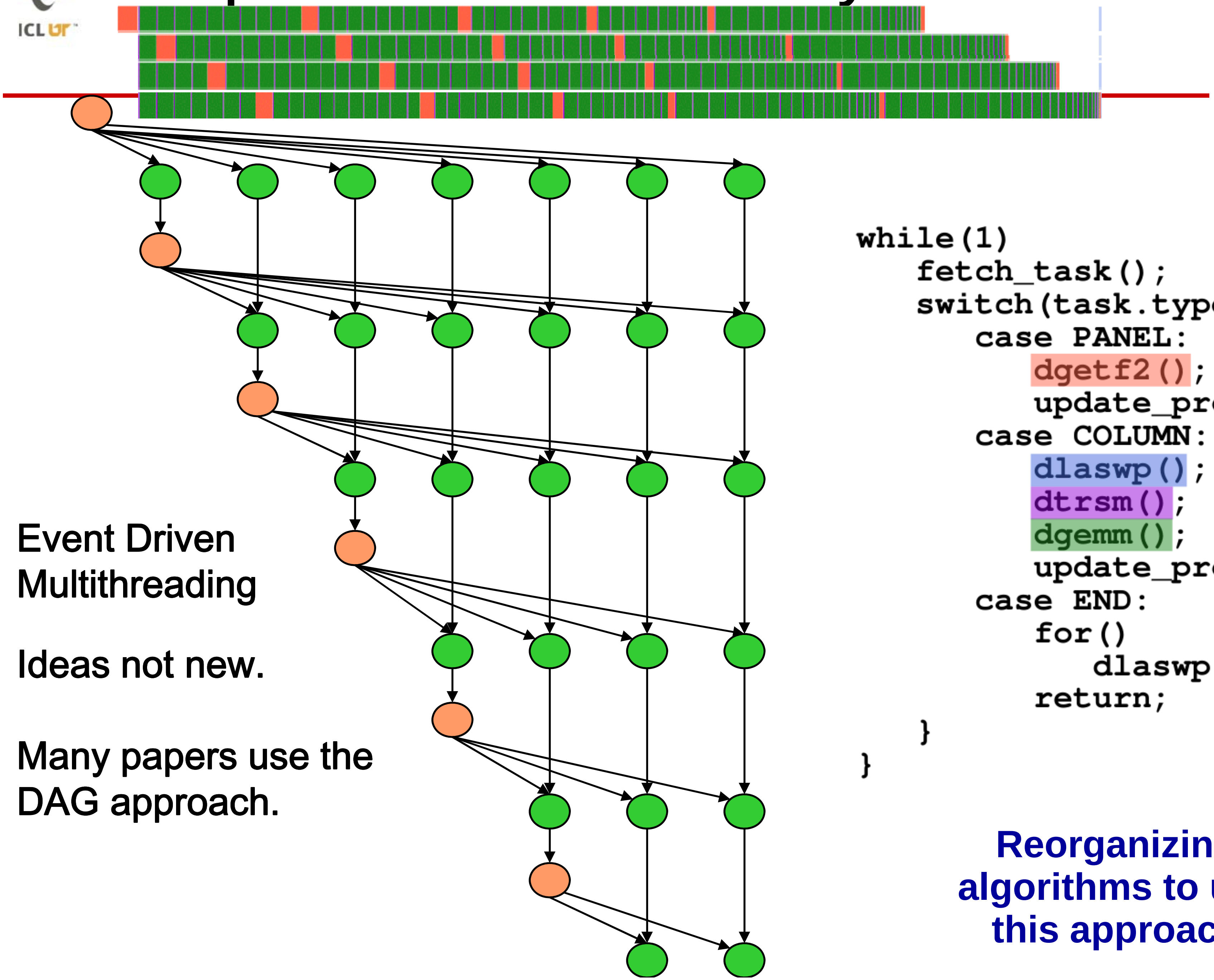
DTRSM

DGEMM



**Bulk Sync Phases**

# Adaptive Lookahead - Dynamic



Event Driven Multithreading

Ideas not new.

Many papers use the DAG approach.

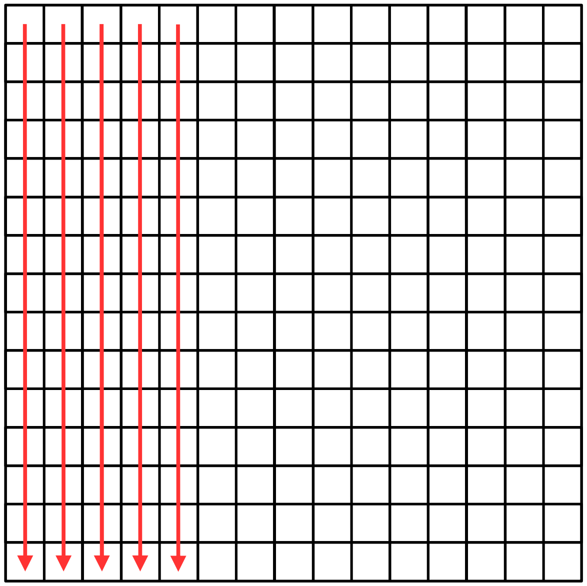
```
while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:
            dgetf2();
            update_progress();
        case COLUMN:
            dlaswp();
            dtrsm();
            dgemm();
            update_progress();
        case END:
            for()
                dlaswp();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

**Reorganizing algorithms to use this approach**

# Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

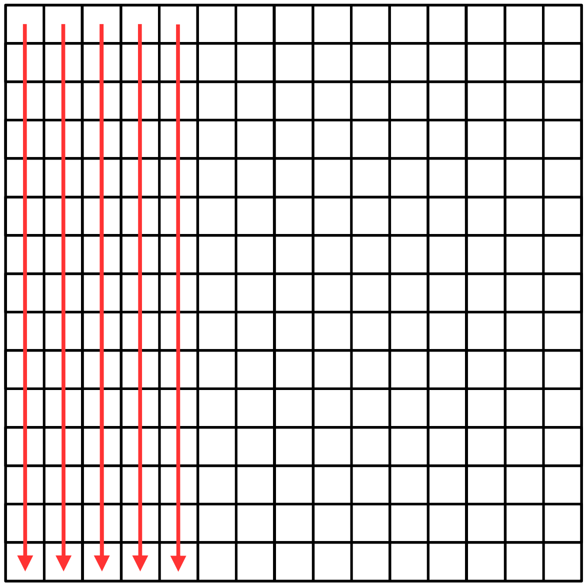
Column-Major



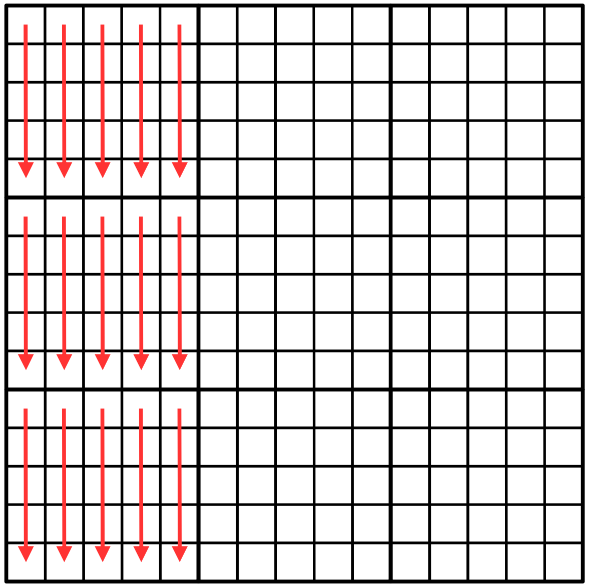
# Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

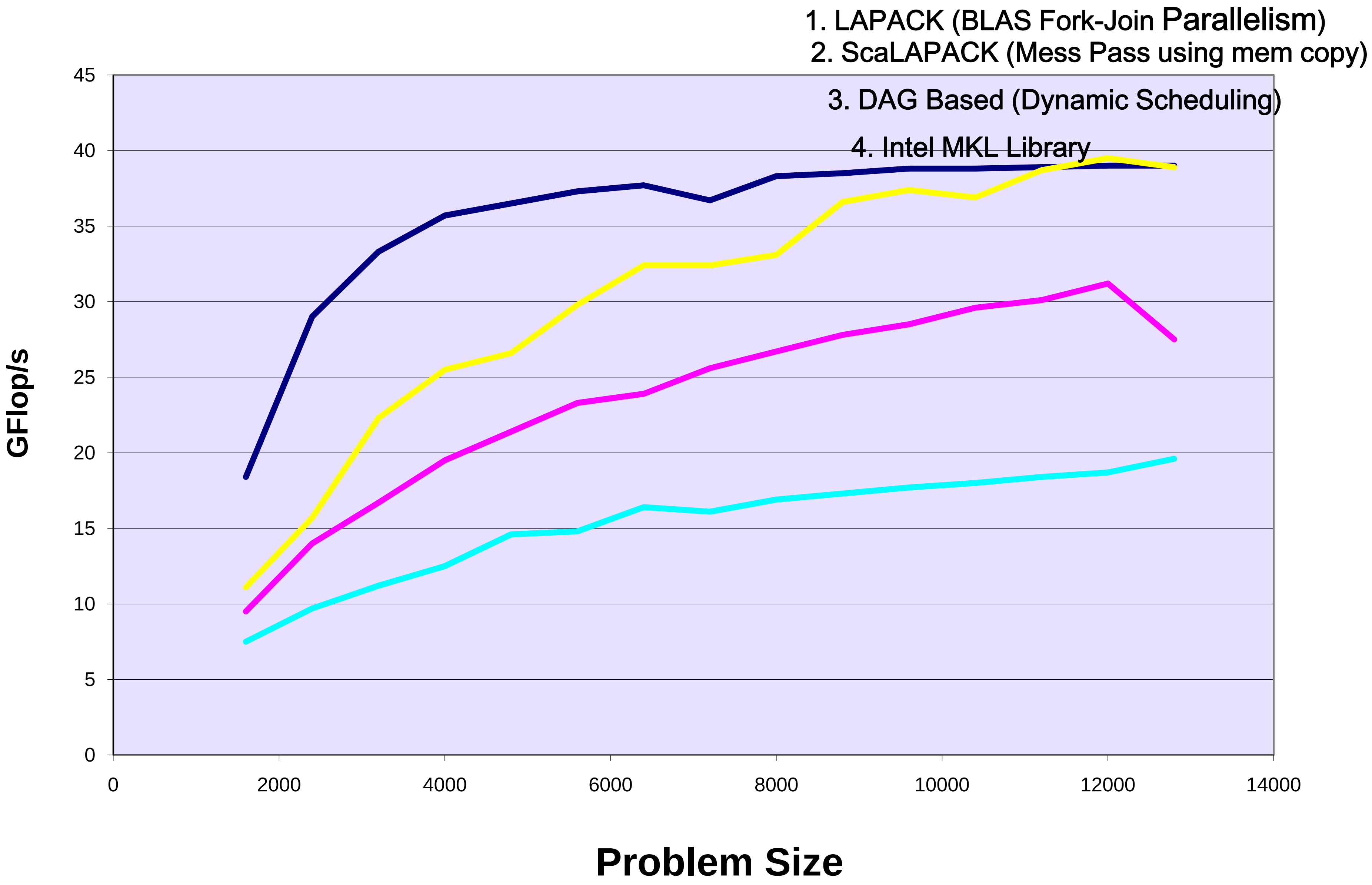
Column-Major

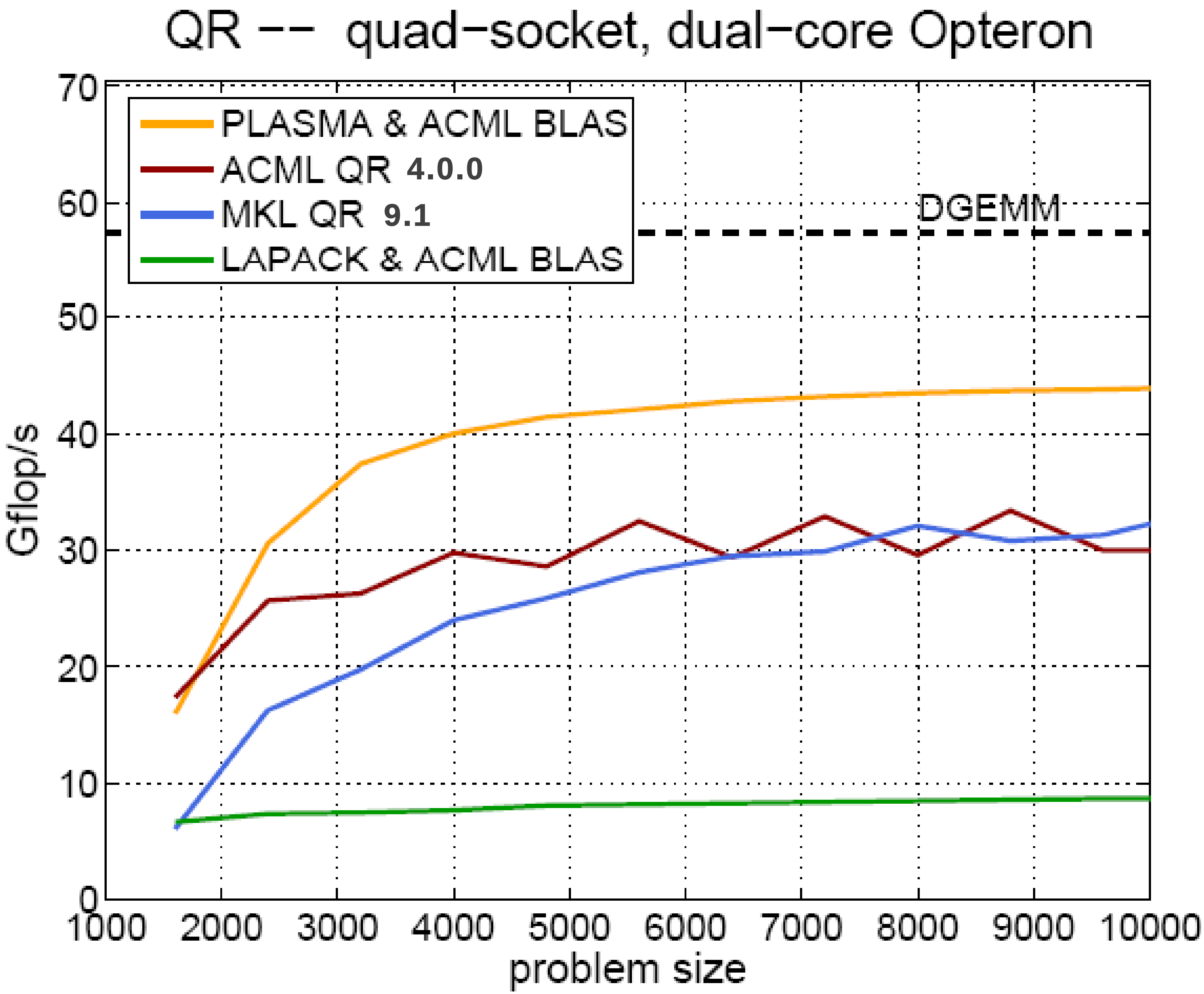


Block data layout

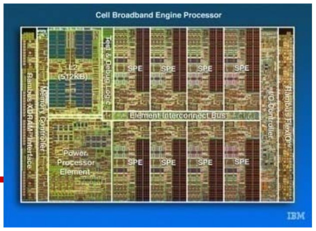


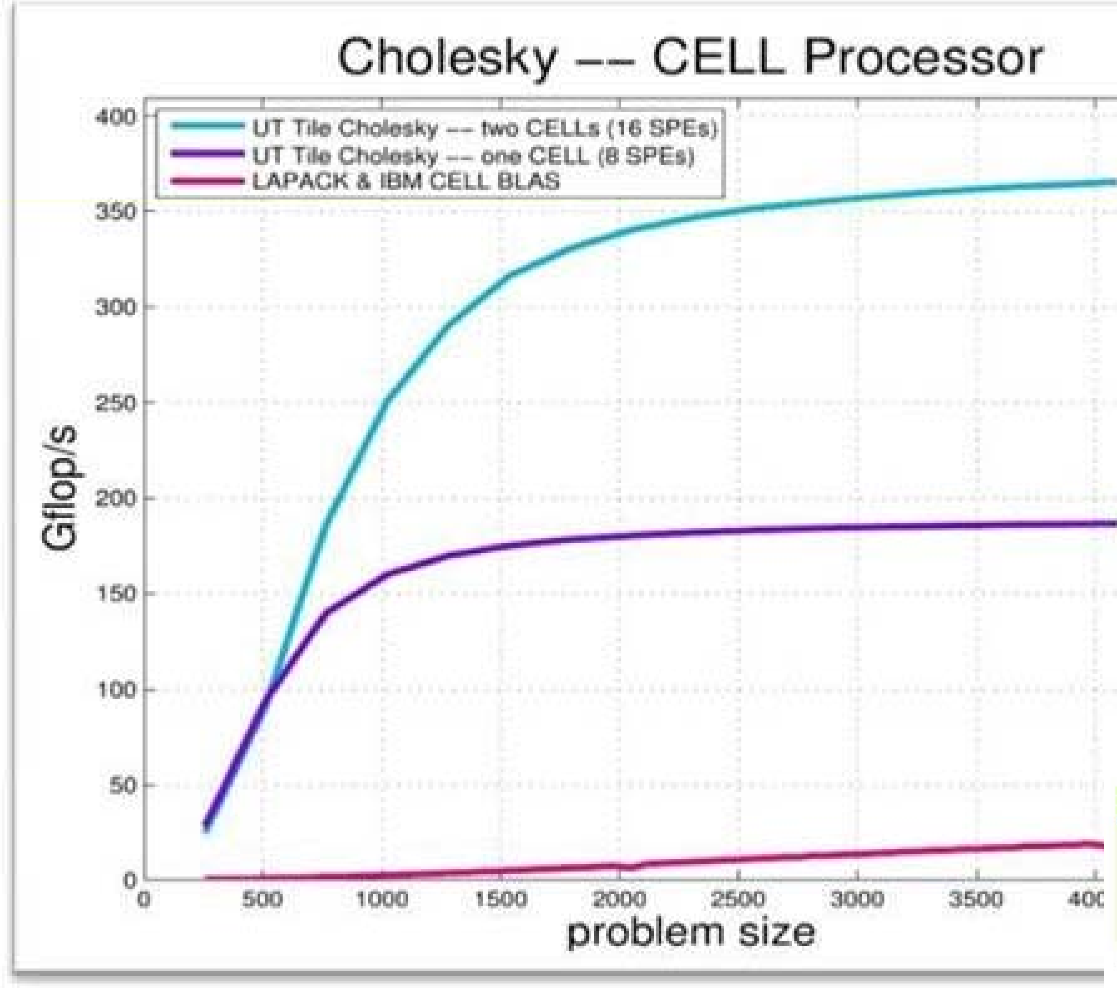
# LU – 16 Core
# (8 Socket - Dual Core Opteron 2.2 GHz)




QR -- quad-socket, dual-core Opteron
PLASMA & ACML BLAS
ACML QR 4.0.0
MKL QR 9.1
LAPACK & ACML BLAS
DGEMM
Gflop/s
0
10
20
30
40
50
60
70
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
problem size

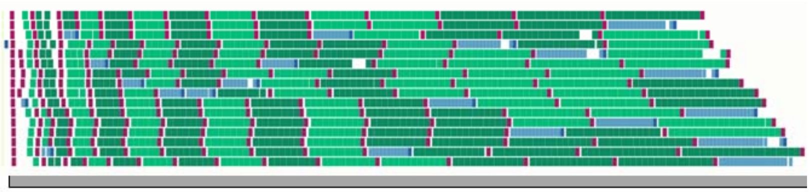
# Cholesky on the CELL



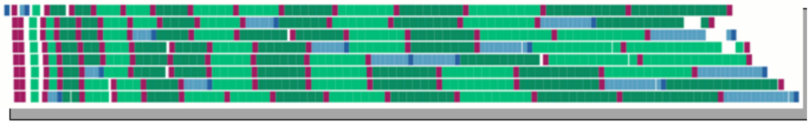


- 1 CELL (8 SPEs)
  - 186 Gflop/s
  - 91 % peak
  - 97 % SGEMM peak
- 2 CELLs (16 SPEs)
  - 365 Gflop/s
  - 89 % peak
  - 95 % SGEMM peak

◆ CELL Cholesky – 16 cores



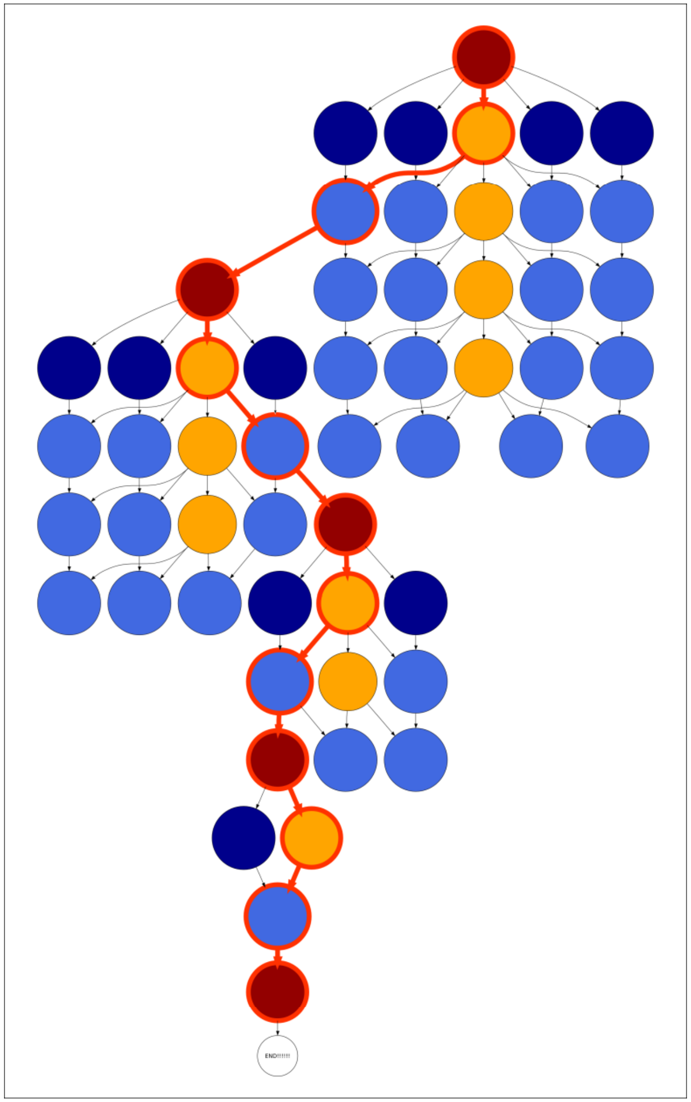
◆ CELL Cholesky – 8 cores



Single precision results on the Cell

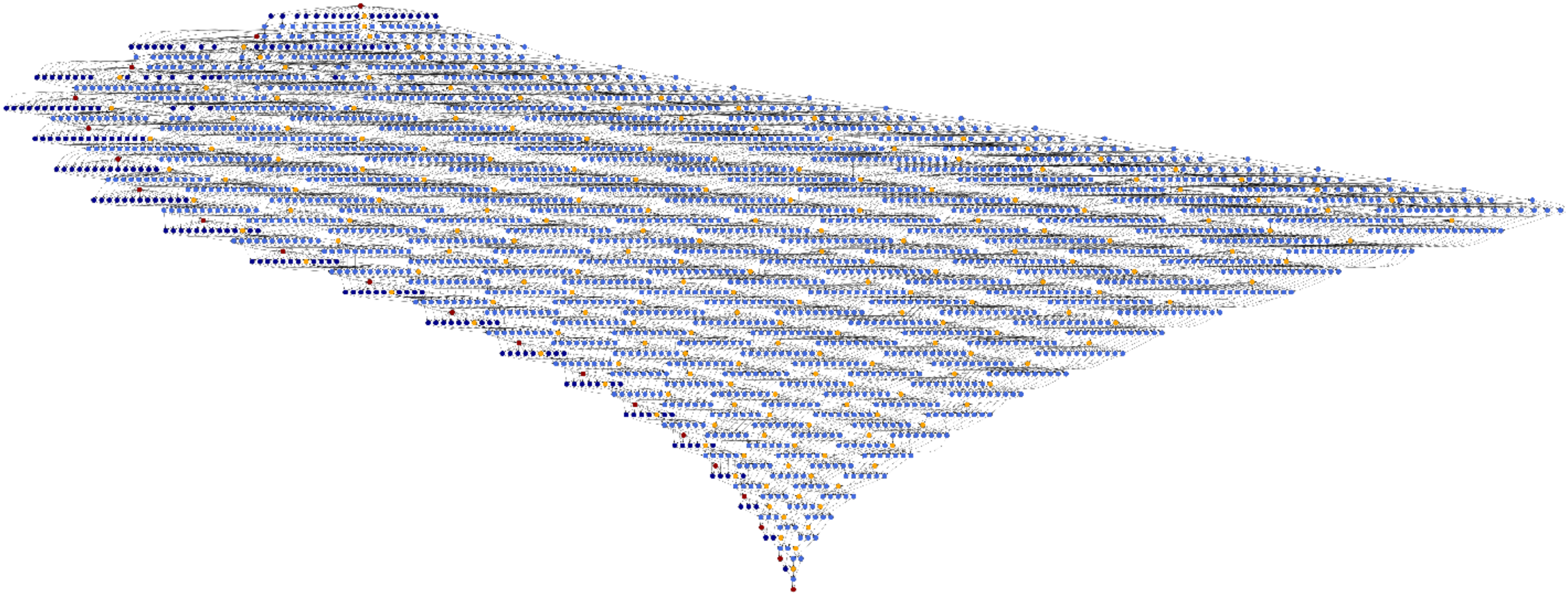
# If We Had A Small Matrix Problem

- We would generate the DAG, find the critical path and execute it.
- DAG too large to generate ahead of time
  - Not explicitly generate
  - Dynamically generate the DAG as we go
- Machines will have large number of cores in a distributed fashion
  - Will have to engage in message passing
  - Distributed management
  - Locally have a run time system



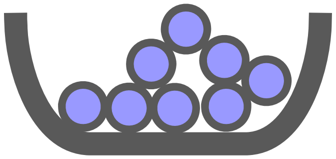
# The DAGs are Large

- **Here is the DAG for the QR factorization on a 20 x 20 matrix**



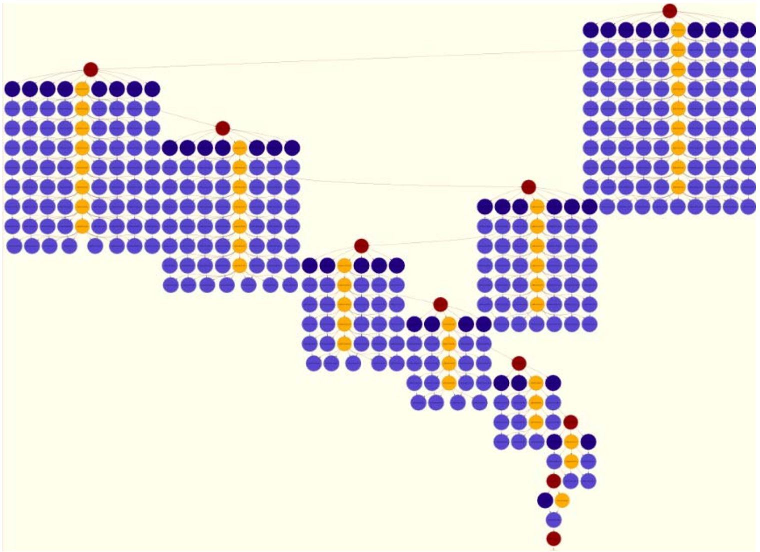
- **For a large matrix say O($10^6$) the DAG is huge**
- **Many challenges for the software**

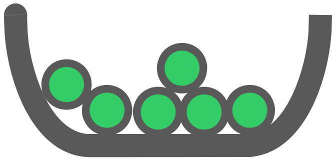
# Each Node or Core Will Have A Run Time System



BIN 1

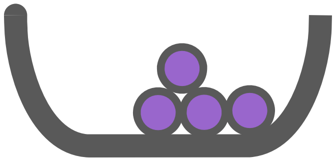
- some dependencies satisfied
- waiting for all dependencies





BIN 2

- all dependencies satisfied
- some data delivered
- waiting for all data



BIN 3

- all data delivered
- waiting for execution

# 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:

- Higher parallelism in SSE/vector units
- Reduced data motion
- Higher locality in cache

# 32 or 64 bit Floating Point Precision?

- A long time ago 32 bit floating point was used
  - Still used in scientific apps but limited
- Most apps use 64 bit floating point
  - Accumulation of round off error
    - A 10 TFlop/s computer running for 4 hours performs > 1 Exaflop ($10^{18}$) ops.
  - Ill conditioned problems
  - IEEE SP exponent bits too few (8 bits, $10^{\pm 38}$)
  - Critical sections need higher precision
    - Sometimes need extended precision (128 bit fl pt)
  - However some can get by with 32 bit fl pt in some parts
- Mixed precision a possibility
  - Approximate in lower precision and then refine or improve solution to high precision.

# 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.**

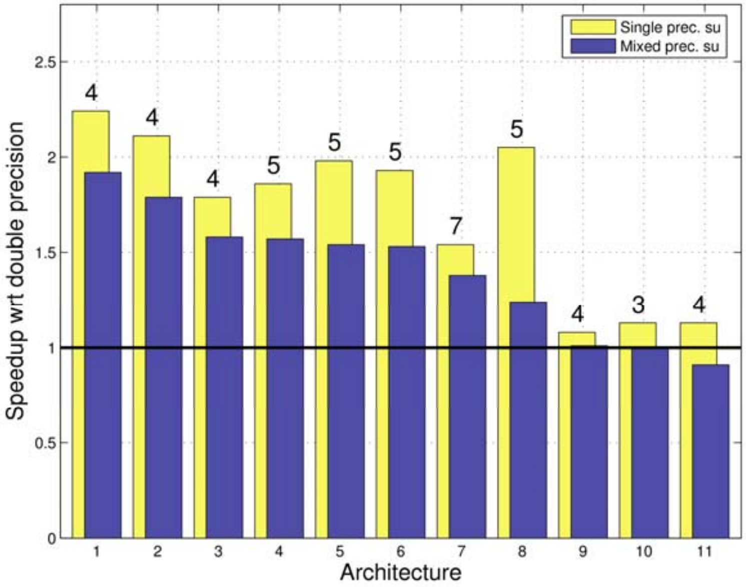
```
L U = lu(A)                          SINGLE    $O(n^3)$
x = L\(U\b)                          SINGLE    $O(n^2)$
r = b – Ax                           DOUBLE    $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)                  SINGLE    $O(n^2)$
        x = x + z                    DOUBLE    $O(n^1)$
        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; $O(10^8)$**

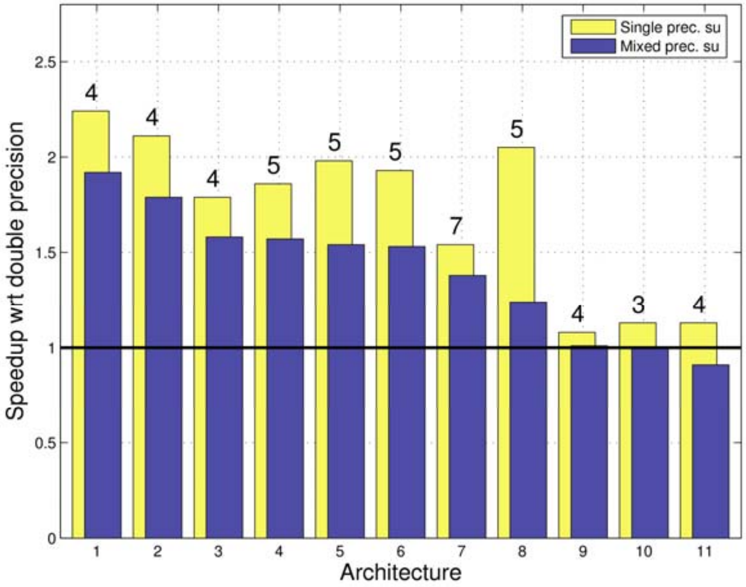
# Results for Mixed Precision Iterative Refinement for Dense *Ax* = *b*



| | 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*



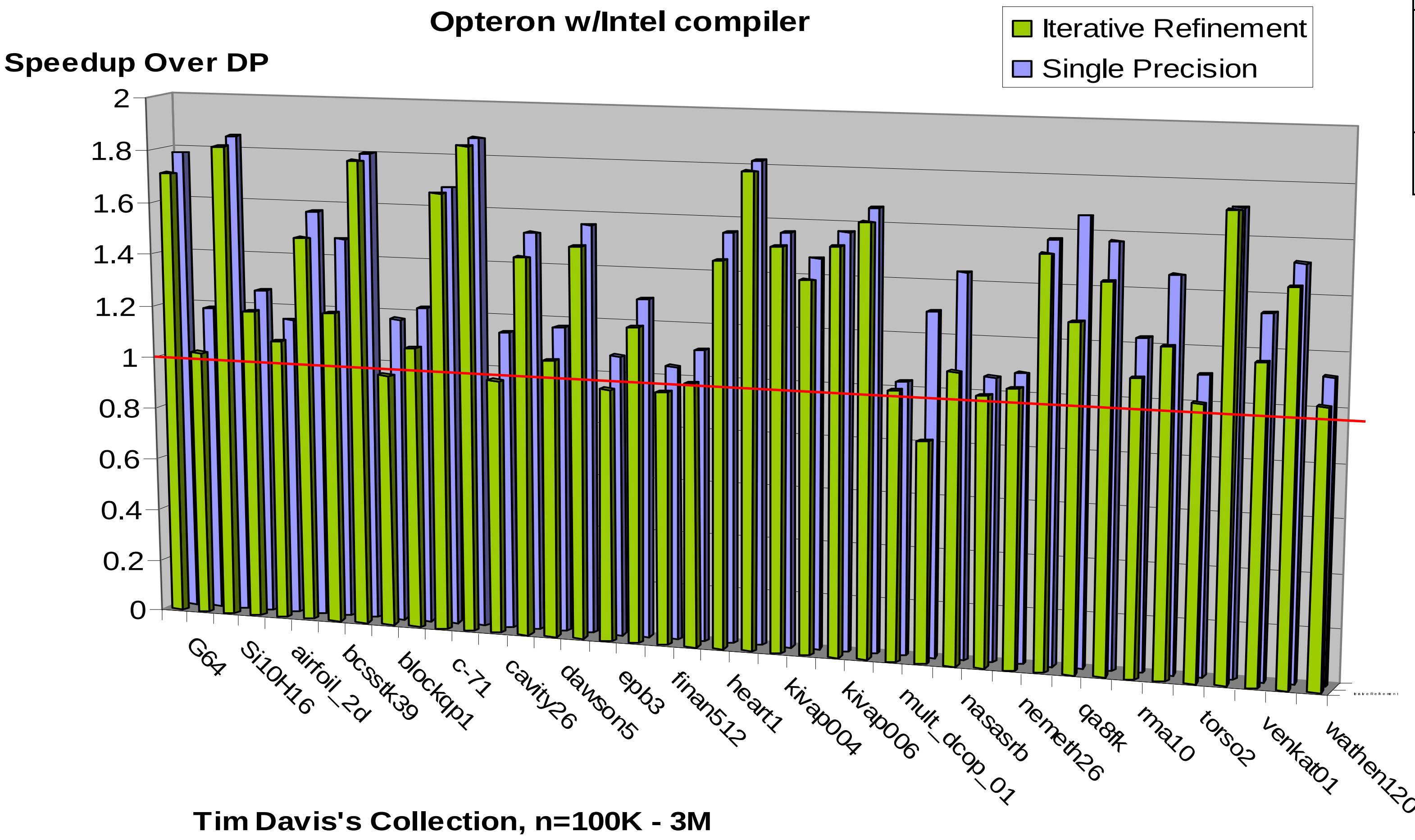
|  | Architecture (BLAS) |
|---|---|
| 1 | Intel Pentium III Coppermine (Goto) |
| 2 | Intel Pentium III Katmai (Goto) |
| 3 | Sun UltraSPARC IIe (Sunperf) |
| 4 | Intel Pentium IV Prescott (Goto) |
| 5 | Intel Pentium IV-M Northwood (Goto) |
| 6 | AMD Opteron (Goto) |
| 7 | Cray X1 (libsci) |
| 8 | 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** |

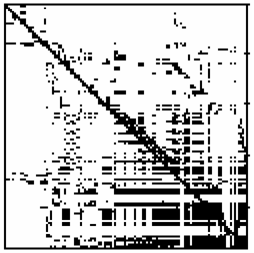
- Single precision is faster than DP because:
  - **Higher parallelism within vector units**
    - **4 ops/cycle (usually) instead of 2 ops/cycle**
  - **Reduced data motion**
    - **32 bit data instead of 64 bit data**
  - **Higher locality in cache**
    - **More data items in cache**

# Sparse Direct Solver and Iterative Refinement

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



Opteron w/Intel compiler

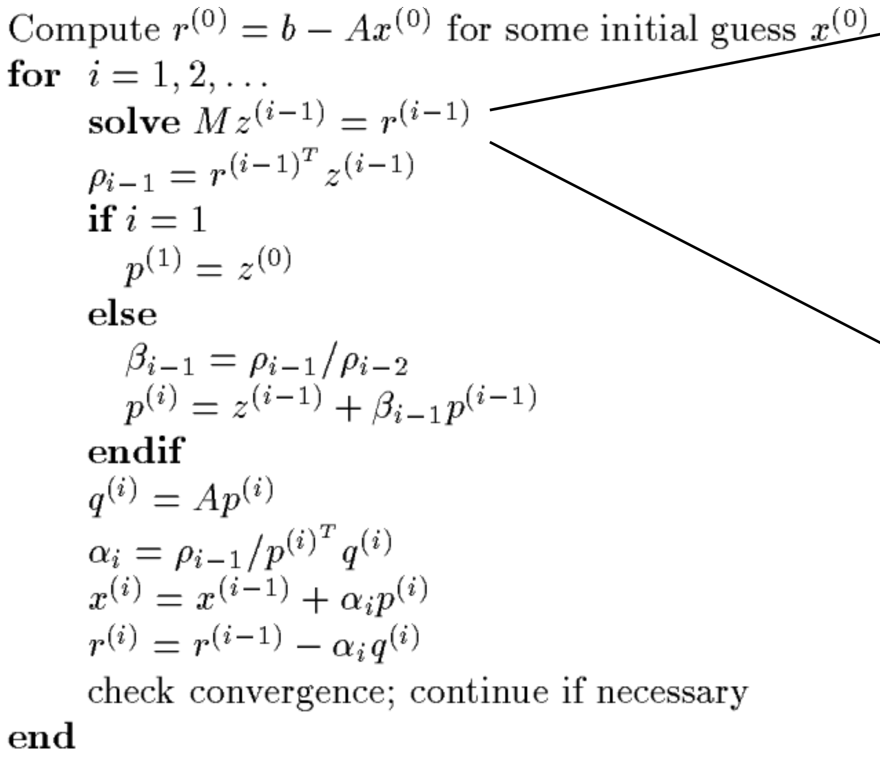


# Sparse Iterative Methods (PCG)

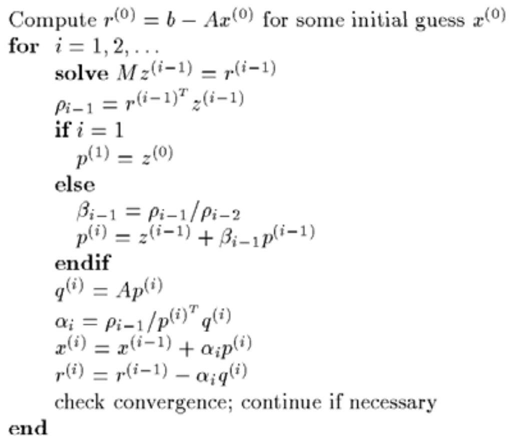
- **Outer/Inner Iteration**

Inner iteration:
In 32 bit floating point

Outer iterations using 64 bit floating point



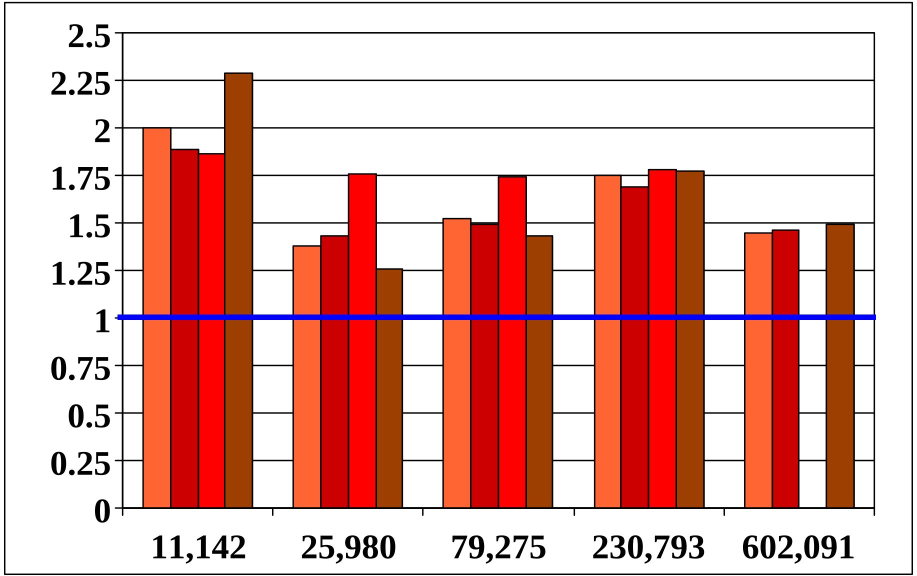
```
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for  $i = 1, 2, \ldots$
    solve $Mz^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
    if $i = 1$
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$
    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
```

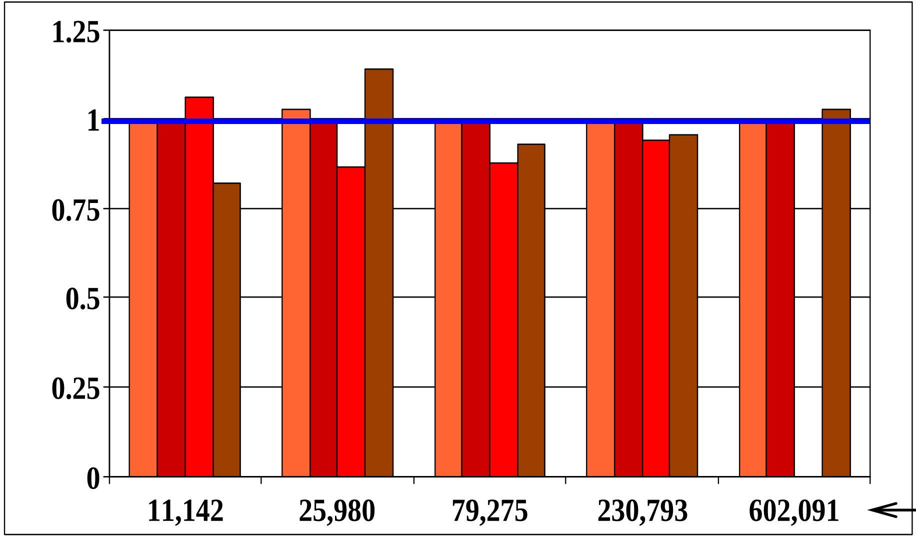
- **Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**

# 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)

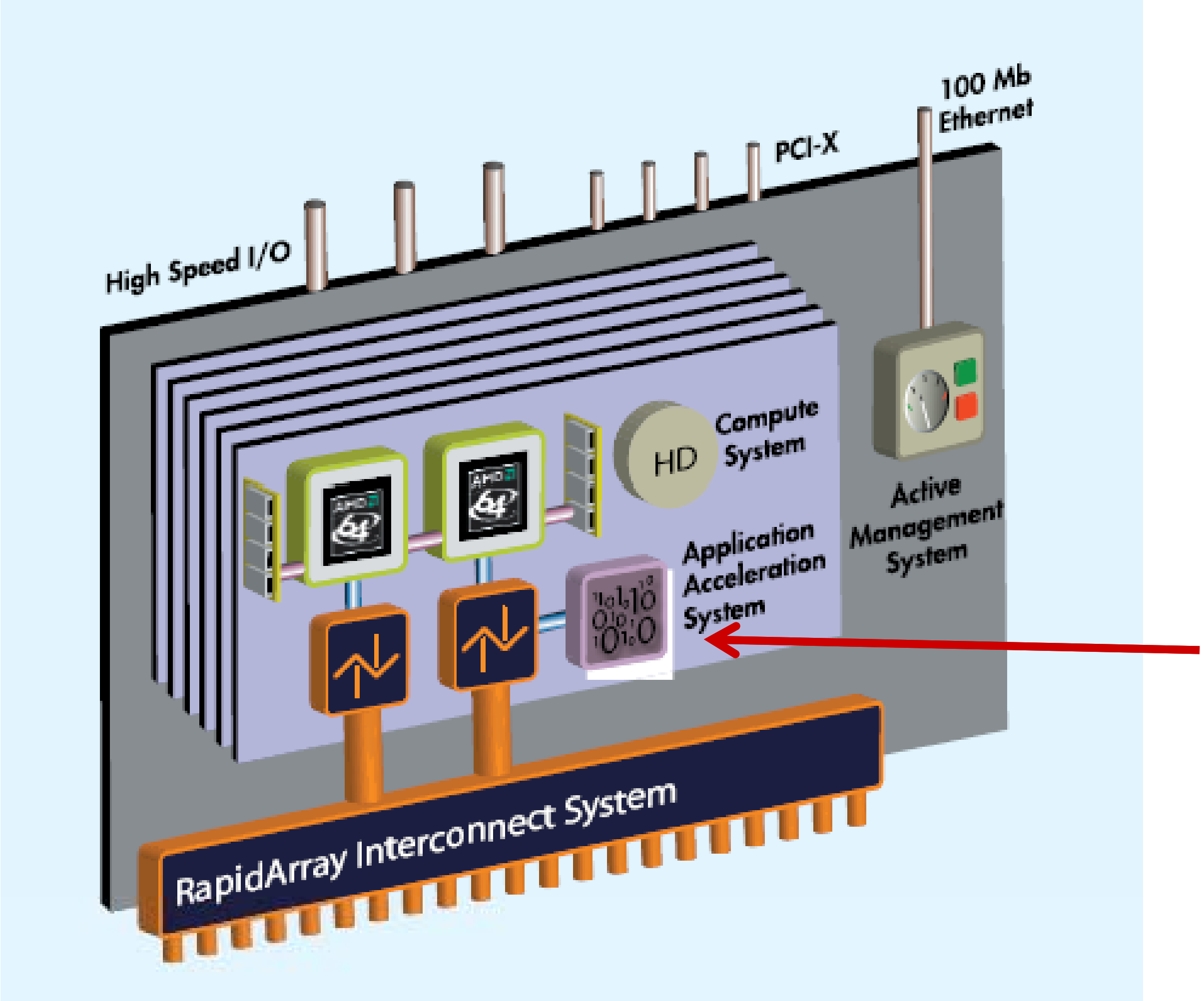
Stopping criteria:
Relative to $r_0$ residual reduction ($10^{-12}$)

← Matrix size

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

# Cray XD-1 (OctigaBay Systems)

Experiments with Field Programmable Gate Array (FPGA)
Specify arithmetic precision



**Six Xilinx Virtex-4 Field Programmable Gate Arrays (FPGAs) per chassis**

# Mixed Precision Iterative Refinement

## - FPGA Performance Test - Junqing Sun et al

Characteristics of multiplier on an FPGA* (using DSP48)

| **Data Formats** | **DSP48s** | **Frequency ( MHz)** | **GFLOPs** |
|---|---|---|---|
| s52e11 (double) | 16/96 | 237 | **1.42** |
| s51e11 | 16/96 | 238 | 1.43 |
| **s50e11** | **9/96** | 245 | 2.61 |
| s34e8 | 9/96 | 289 | 3.08 |
| **s33e8** | **4/96** | 292 | 7.01 |
| s23e8 (single) | 4/96 | 339 | **8.14** |
| s17e8 | 4/96 | 370 | 8.88 |
| **s16e8** | **1/96** | 331 | 31.78 |
| s16e7 | 1/96 | 352 | 33.79 |
| s13e7 | 1/96 | 336 | **32.26** |

* XC4LX160-10

# Mixed Precision Iterative Refinement

## - Random Matrix Test - Junqing Sun et al

Refinement iterations for customized formats (sXXe11).
Random matrices (average number of iterations/random matrices)

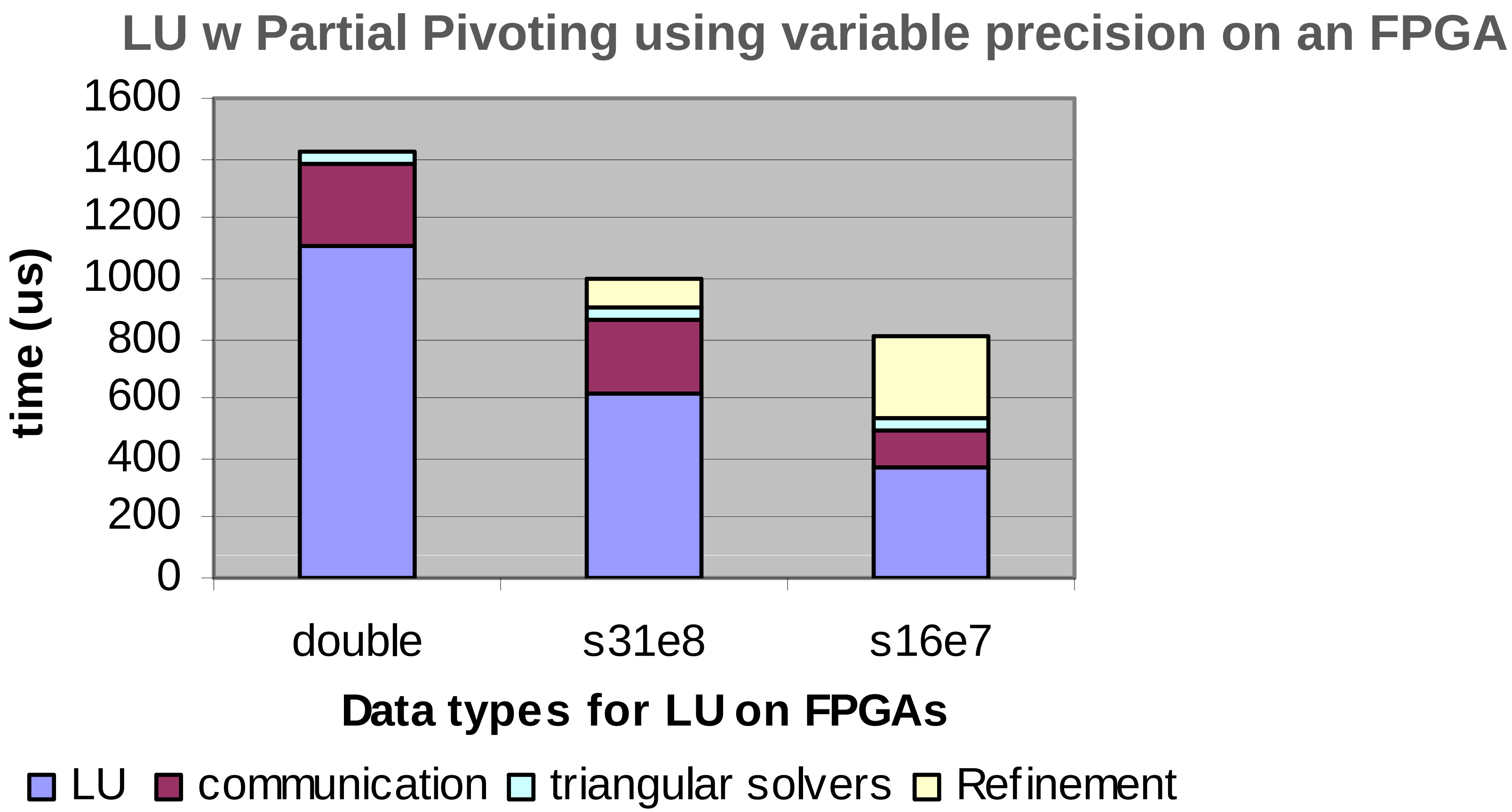
More Bits →

| Mantissa Bits<br>Problem Size | 12 | 16 | **23** | 31 | 48 | **52** |
|---|---|---|---|---|---|---|
| 128 | 8.9 | 4 | 2 | 1 | 1 | 0 |
| 256 | 11.1 | 5.1 | 2.1 | 1 | 1 | 0 |
| 512 | 19.7 | 6.1 | 2.5 | 1 | 1 | 0 |
| 1024 | 28 | 6.3 | 2.6 | 1 | 1 | 0 |
| 2048 | - | 9.3 | 3 | 1.3 | 1 | 0 |
| 4096 | - | 13.3 | 3.1 | 1.43 | 1 | 0 |

← More Iterations

# Mixed Precision Hybrid Direct Solver

## - Profiled Time* on Cray-XD1 - Junqing Sun et al



* For a 128x128 matrix

**High Performance Mixed-Precision Linear Solver for FPGAs,**
Junqing Sun, Gregory D. Peterson, Olaf Storaasli, To appear IEEE TPDC

# 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**
  - **What about 16 bit floating point?**
    - **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)

# 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**

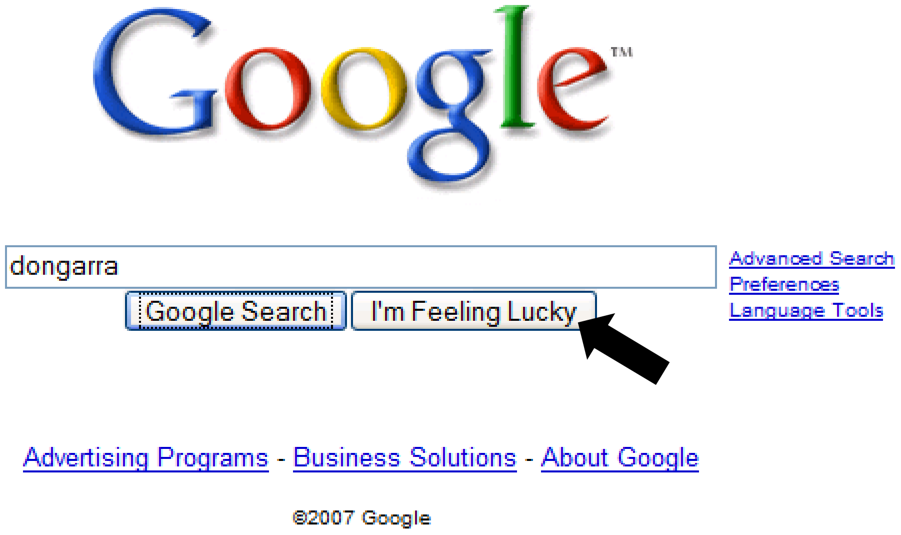
# Collaborators / Support

**Marc Baboulin UCoimbra**
**Alfredo Buttari, ENS/INRIA**
**Bilel Hadri, UTK**
**Julien Langou, UColorado**
**Julie Langou, UTK**
**Hatem Ltaief, UTK**
**Piotr Luszczek, MathWorks**
**Jakub Kurzak, UTK**
**Stan Tomov, UTK**

NSF

Microsoft

The MathWorks



dongarra

©2007 Google

# PLASMA Collaborators

- U Tennessee, Knoxville
  - Jack Dongarra, Julie Langou, Stan Tomov, Jakub Kurzak, Hatem Ltaief, Alfredo Buttari, Julien Langou, Piotr Luszczek, Marc Baboulin
- UC Berkeley
  - Jim Demmel, Ming Gu, W. Kahan, Beresford Parlett, Xiaoye Li, Osni Marques, Yozo Hida, Jason Riedy, Vasily Volkov, Christof Voemel, David Bindel
- Other Academic Institutions
  - UC Davis, CU Denver, Florida IT, Georgia Tech, U Maryland, North Carolina SU, UC Santa Barbara, UT Austin, LBNL
  - TU Berlin, ETH, U Electrocomm. (Japan), FU Hagen, U Carlos III Madrid, U Manchester, U Umeå, U Wuppertal, U Zagreb, UPC Barcelona, ENS Lyon, INRIA
- Industrial Partners
  - Cray, HP, Intel, Interactive Supercomputing, MathWorks, NAG, NVIDIA, Microsoft