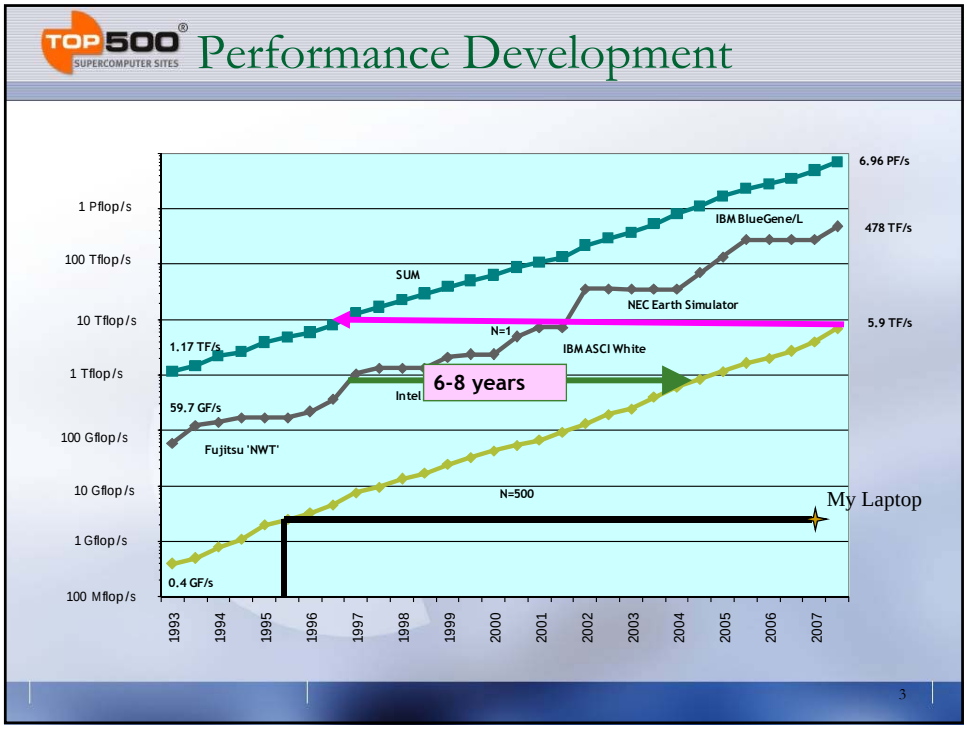
Four Important Concepts that Will Effect Math Software

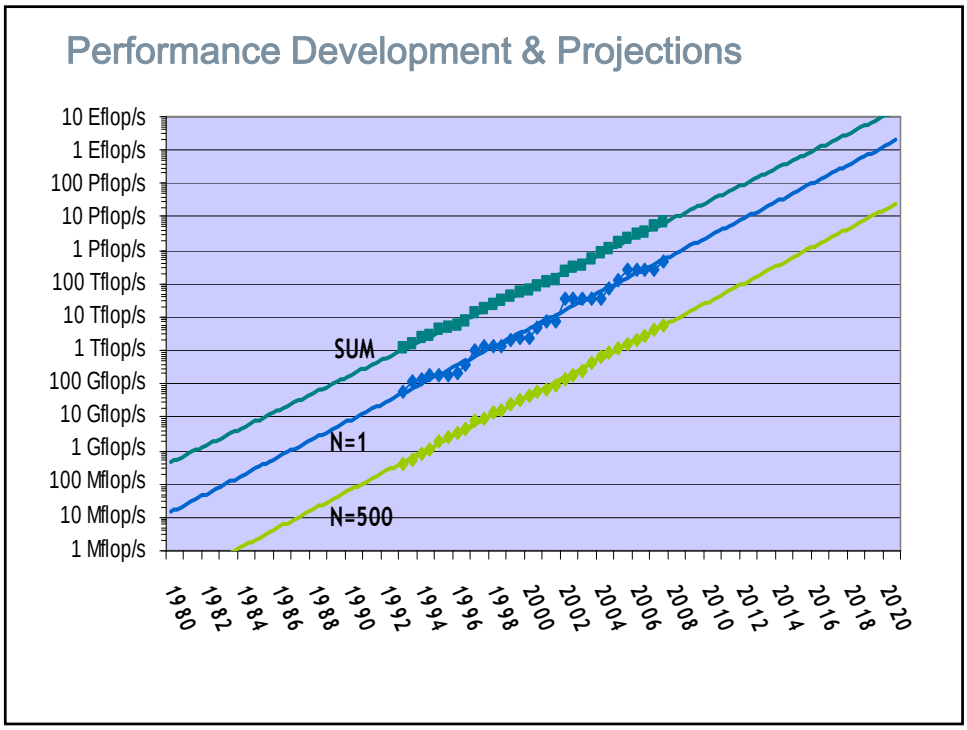
Jack Dongarra
INNOVATIVE COMPUTING LABORATORY

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

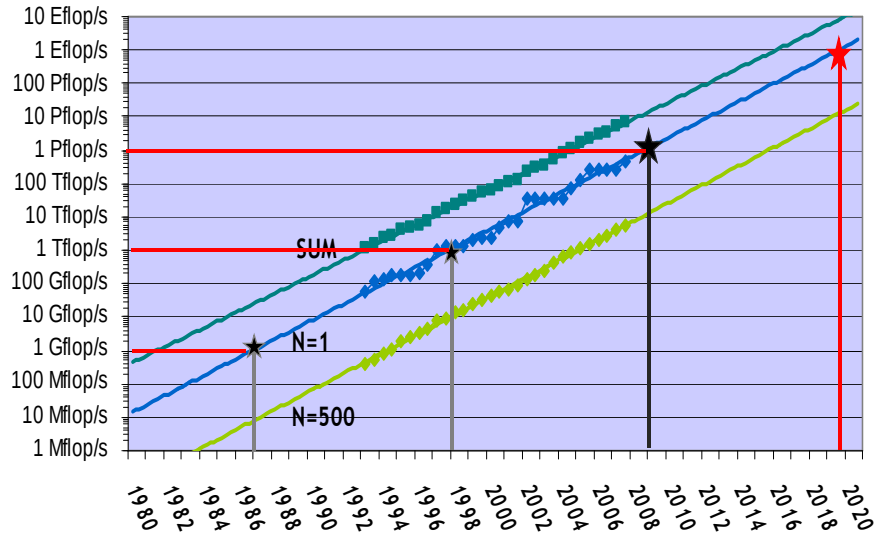
Four Important Concepts that Will Effect Math Software

- **Effective Use of Many-Core**
- **Exploiting Mixed Precision in Our Numerical Computations**
- **Self Adapting / Auto Tuning of Software**
- **Fault Tolerant Algorithms**

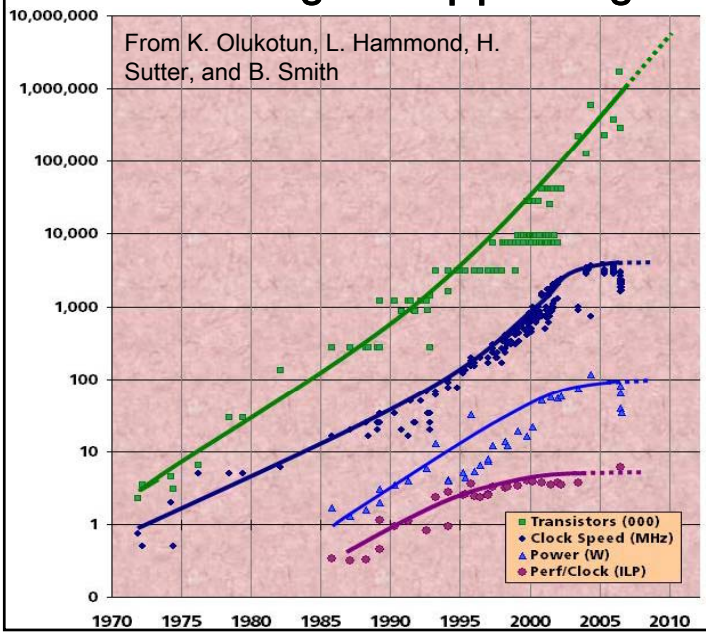

TOP500 SUPERCOMPUTER SITES
Performance Development
1 Pflop/s
100 Tflop/s
10 Tflop/s
1 Tflop/s
100 Gflop/s
10 Gflop/s
1 Gflop/s
100 Mflop/s
6.96 PF/s
478 TF/s
5.9 TF/s
IBM BlueGene/L
SUM
NEC Earth Simulator
N=1
1.17 TF/s
IBM ASCI White
6-8 years
Intel
59.7 GF/s
Fujitsu 'NWT'
N=500
My Laptop
0.4 GF/s
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
3


Performance Development & Projections
10 Eflop/s
1 Eflop/s
100 Pflop/s
10 Pflop/s
1 Pflop/s
100 Tflop/s
10 Tflop/s
1 Tflop/s
100 Gflop/s
10 Gflop/s
1 Gflop/s
100 Mflop/s
10 Mflop/s
1 Mflop/s
SUM
N=1
N=500
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020

Performance Development & Projections



Something's Happening Here…



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

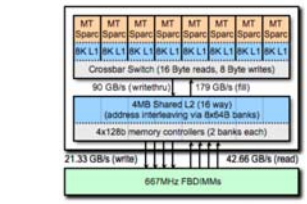
6

Moore's Law Reinterpreted

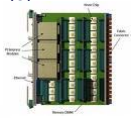
- **Number of cores per chip doubles every two year, while clock speed decreases (not increases).**
  - **Need to deal with systems with millions of concurrent threads**
    - **Future generation will have billions of threads!**
  - **Need to be able to easily replace inter-chip parallelism with intro-chip parallelism**

Today's Multicores

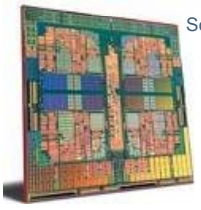
90% of Top500 Systems Are Based on Multicore



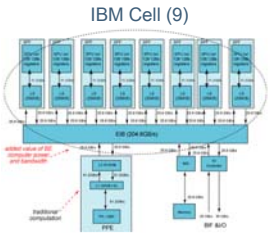
Sun Niagra2 (8)



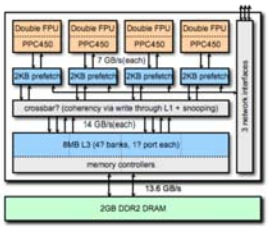
SciCortex (6)



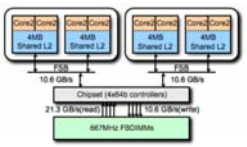
AMD Opteron (4)



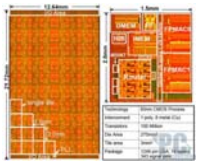
IBM Cell (9)



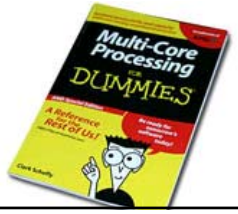
IBM BG/P (4)



Intel Clovertown (4)



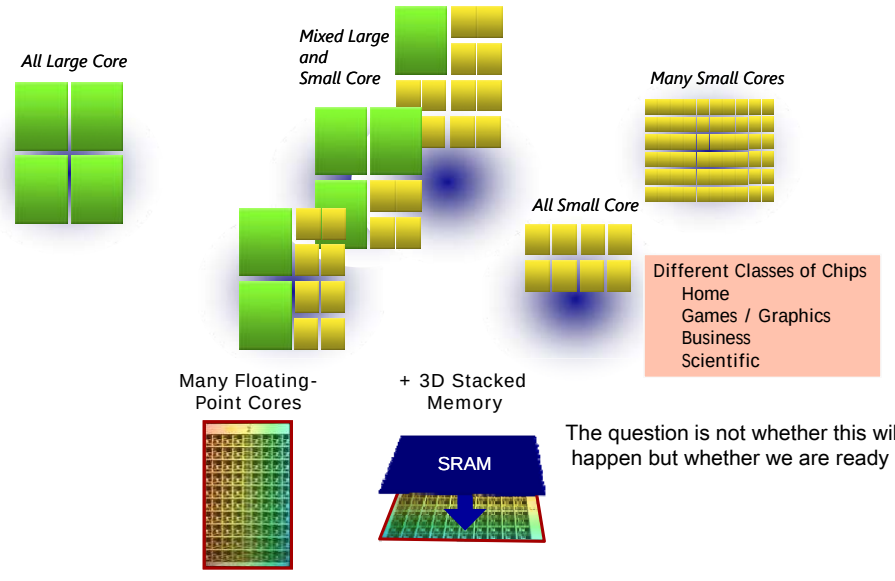
Intel Polaris (80)



Pro's and Con's for Different Computational Platforms

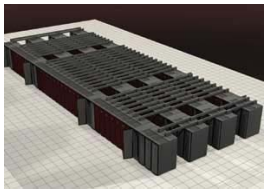
	CORE	CELL	GPU	FPGA
Speed improvement	☹	☺	☺	☺
Ease-of-programming	☺	😐	☹	☹
Availibility of libraries	☺	😐	☹	☹

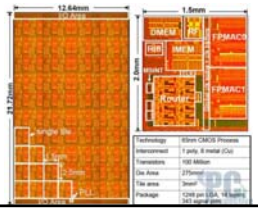
What's Next?



ASCI Red Strom Compared to Intel Polaris Chip

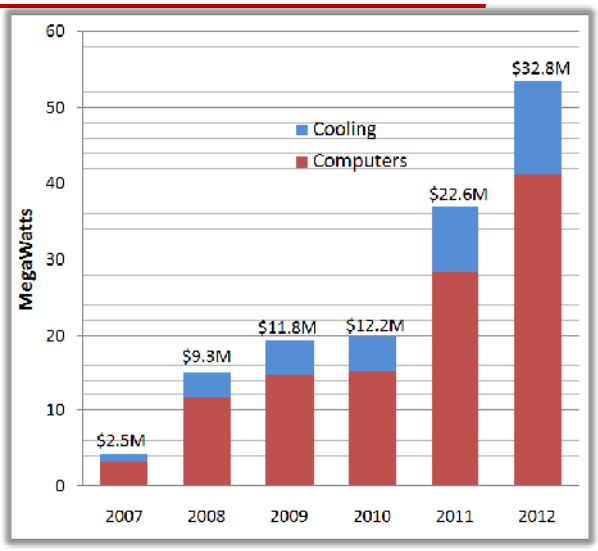
- **ASCI Red Computer**
  - 1996
  - First TFlop/s Computer
  - ~10,000 Pentium Pro
  - 1 MWatt
  - 200 MHz
  - 200 MFlop/s each proc
  - 104 cabinets
  - ~2500 sq ft (230 m²).
- **Intel Polaris Chip**
  - 2007
  - First TFlop/s chip
  - 80 floating point cores
  - 62 Watts
  - 3.16 GHz
  - 1 TFlop/s chip
  - One chip
  - 275 mm²





ORNL/UTK Power Cost Projections 2007-2011

- 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



Includes both DOE and NSF systems.

What Will a Future System Looks Like?

Possible Petascale System	
1. # of cores per nodes (made up of multiple sockets)	10 - 100 cores, possibly hybrid
2. Performance per nodes	100 - 1,000 GFlop/s
3. Number of nodes	1,000 - 10,000 nodes
4. Latency inter-nodes	1 µsec
5. Bandwidth inter-nodes	10 GB/s
6. Memory per nodes	10 GB

- In general would like high…
  - 2. performance per node 5. bandwidth inter-nodes 6. memory per nodes
- Algorithms for multicore and need for latency avoiding algorithms
  - 1. Number of cores per node 2. performance per node
  - 4. Latency inter-nodes
- Issues involving fault tolerance
  - Motivation in:
    - 1. Number of cores per node 3. number of nodes

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

15

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

16

A New Generation of Software:

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

A New Generation of Software:

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:

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:

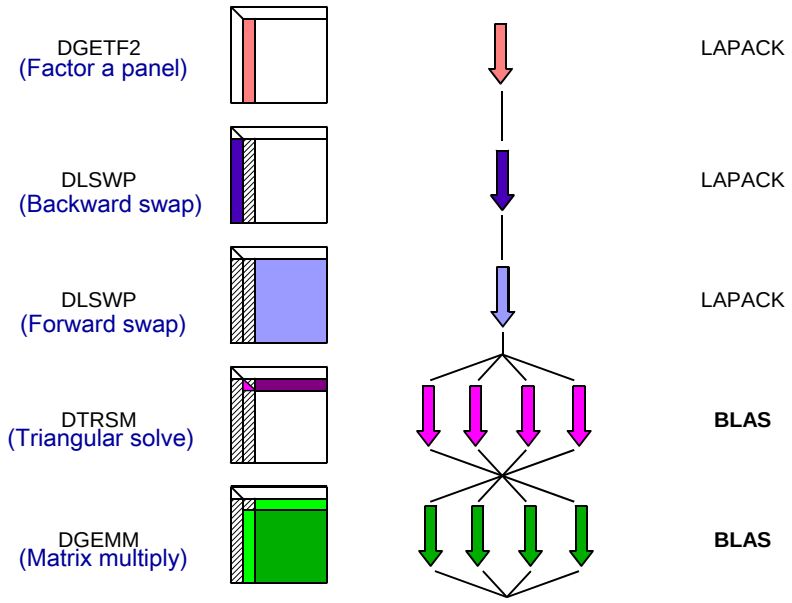
Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

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

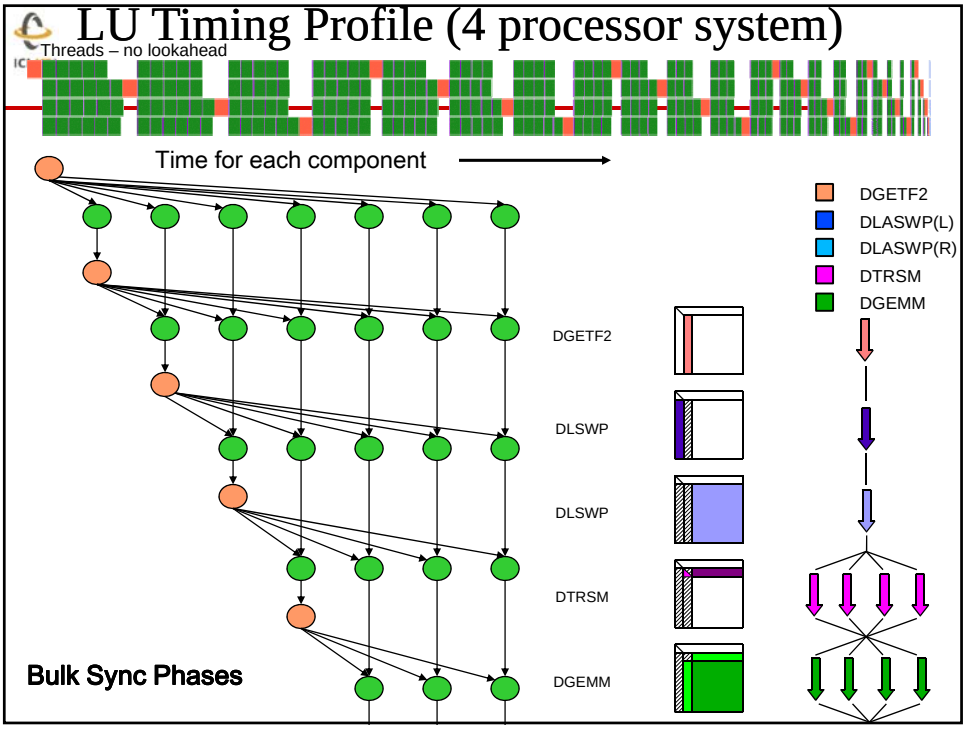
Those new algorithms need new kernels and rely on efficient scheduling algorithms.

Steps in the LAPACK LU

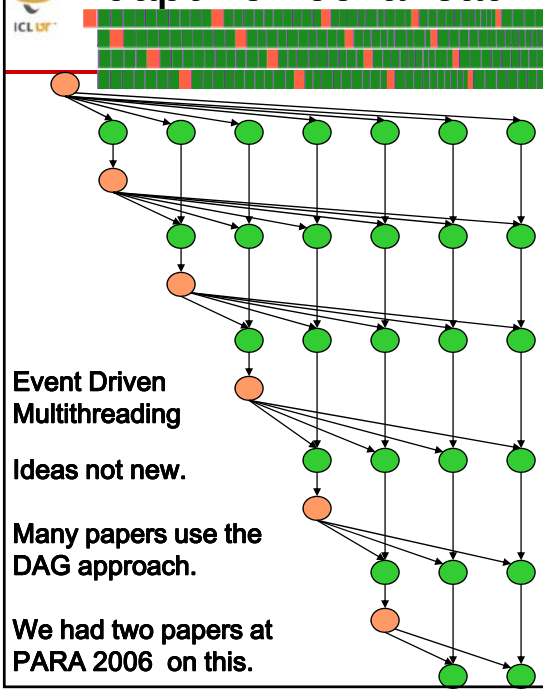


21

LU Timing Profile (4 processor system)



Adaptive Lookahead - Dynamic



Event Driven Multithreading

Ideas not new.

Many papers use the DAG approach.

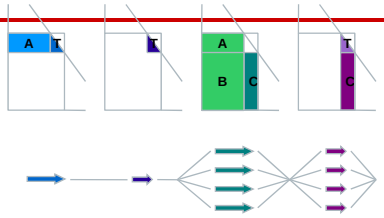
We had two papers at PARA 2006 on this.

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

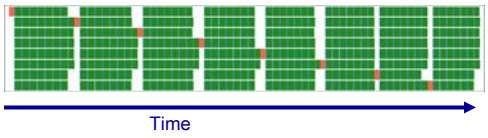
Reorganizing algorithms to use this approach

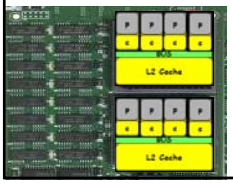
23

Fork-Join vs. Dynamic Execution



Fork-Join – parallel BLAS

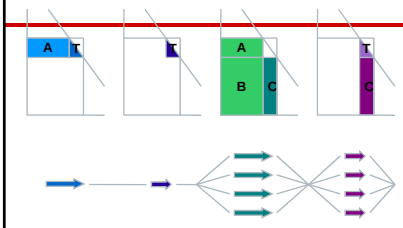


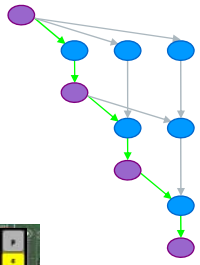


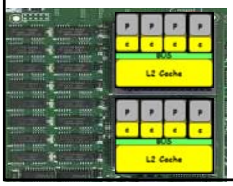
Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

24

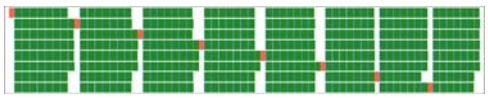
Fork-Join vs. Dynamic Execution







Fork-Join – parallel BLAS



Time

DAG-based – dynamic scheduling

Time saved

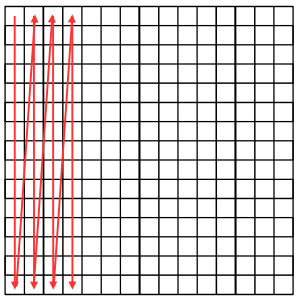
Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

25

Achieving Fine Granularity

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

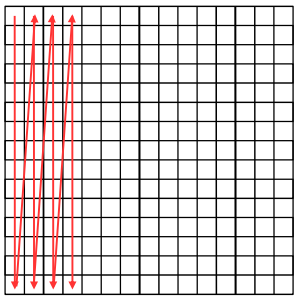
Column-Major



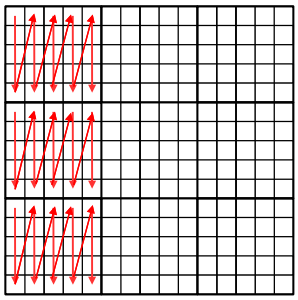
Achieving Fine Granularity

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

Column-Major



Blocked



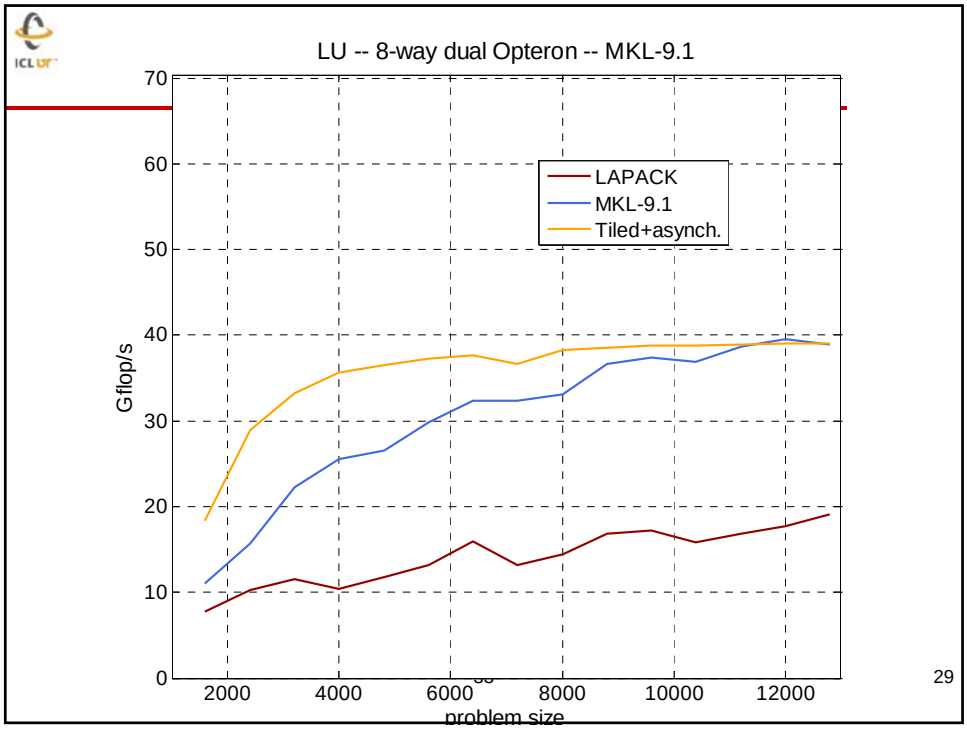
PLASMA (Redesign LAPACK/ScaLAPACK)

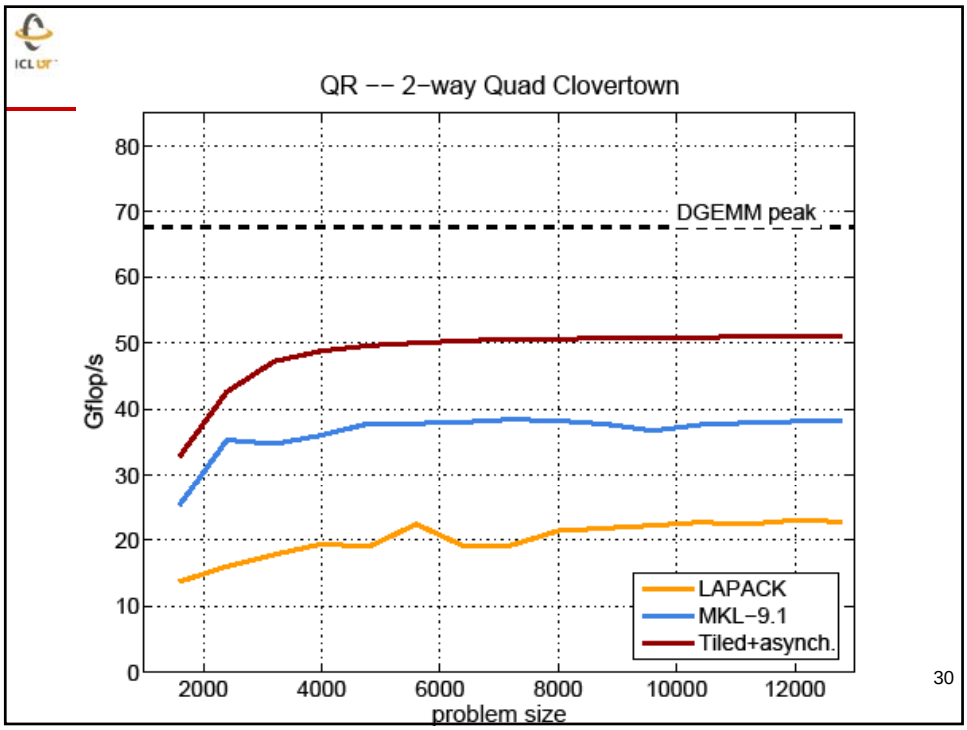
Parallel Linear Algebra Software for Multicore Architectures

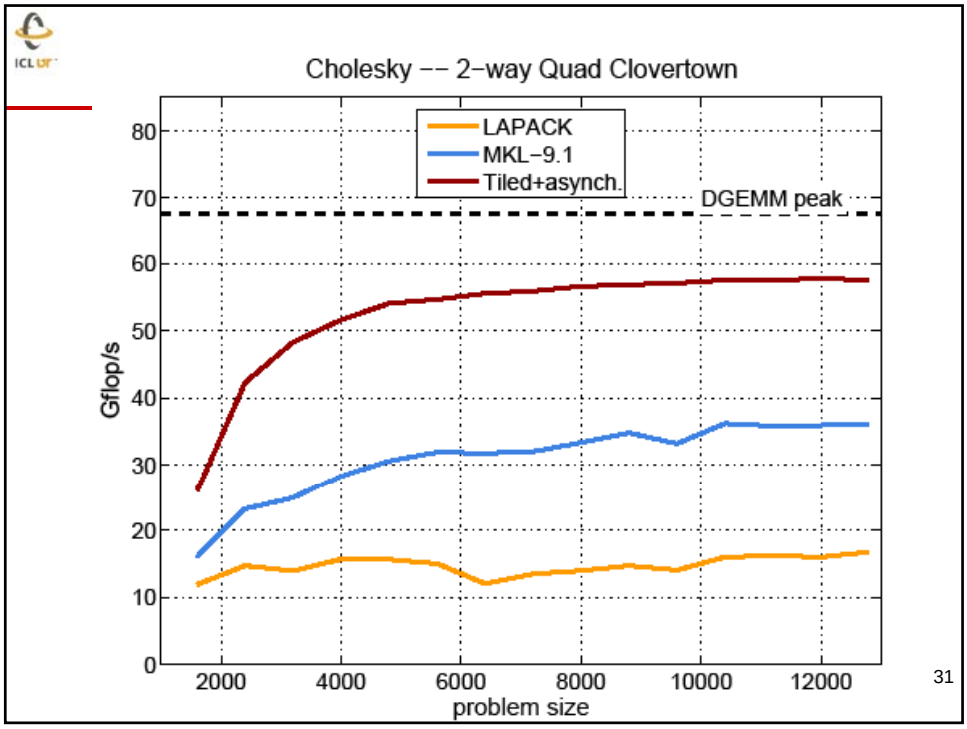
- **Asychronicity**
  - **Avoid fork-join (Bulk sync design)**
- **Dynamic Scheduling**
  - **Out of order execution**
- **Fine Granularity**
  - **Independent block operations**
- **Locality of Reference**
  - **Data storage - Block Data Layout**

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

28

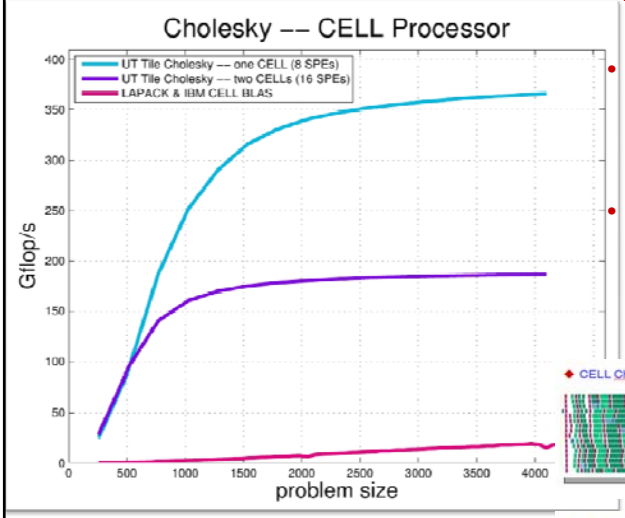

LU -- 8-way dual Opteron -- MKL-9.1
Gflop/s
problem size
LAPACK
MKL-9.1
Tiled+asynch.
0
10
20
30
40
50
60
70
2000
4000
6000
8000
10000
12000
29


QR -- 2-way Quad Clovertown
Gflop/s
problem size
DGEMM peak
LAPACK
MKL-9.1
Tiled+asynch.
0
10
20
30
40
50
60
70
80
2000
4000
6000
8000
10000
12000
30

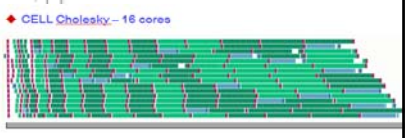


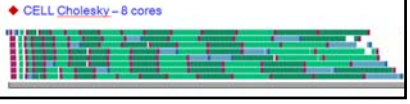
31

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**





Single precision results on the Cell

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.

34

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.

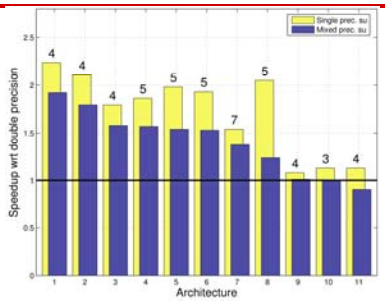
```
L U = lu(A)          SINGLE      O(n^3)
x = L\(U\b)          SINGLE      O(n^2)
r = b - Ax           DOUBLE      O(n^2)
WHILE || r || not small enough
      z = L\(U\r)    SINGLE      O(n^2)
      x = x + z      DOUBLE      O(n^1)
      r = b - Ax     DOUBLE      O(n^2)
END
```

  - 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; $O(10^8)$

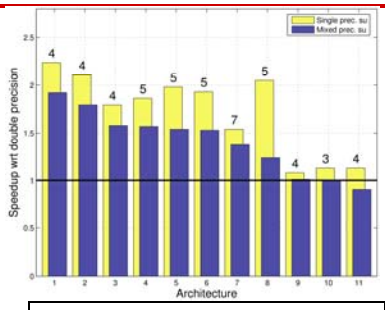
Results for Mixed Precision Iterative Refinement for Dense *Ax = b*



	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*



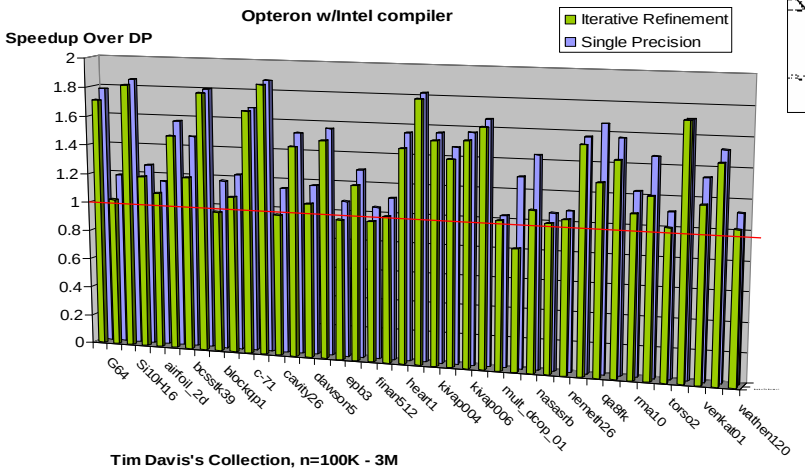
	Architecture (BLAS)
1	Intel Pentium III Coppermine (Goto)
2	Intel Pentium III Katmai (Goto)
3	Sun UltraSPARC IIe (Sunperf)
4	Intel Pentium IV Prescott (Goto)
5	Intel Pentium IV-M Northwood (Goto)
6	AMD Opteron (Goto)
7	Cray X1 (libsci)
8	IBM Power PC G5 (2.7 GHz) (VecLib)
9	Compaq Alpha EV6 (CXML)
10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

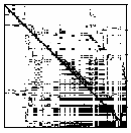
Architecture (BLAS-MPI)	# procs	*n*	DP Solve /SP Solve	DP Solve /Iter Ref	# iter
AMD Opteron (Goto – OpenMPI MX)	32	22627	1.85	1.79	6
AMD Opteron (Goto – OpenMPI MX)	64	32000	1.90	1.83	6

- 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





Sparse Iterative Methods (PCG)

- **Outer/Inner Iteration**

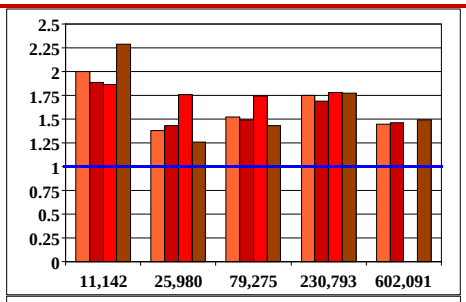
Outer iterations using 64 bit floating point

Inner iteration:
In 32 bit floating point

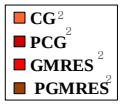
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif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

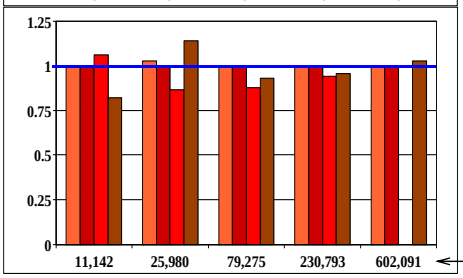
- **Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**

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
(CG², GMRES², PCG², and PGMRES² with diagonal prec.)
(*Higher is better*)





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

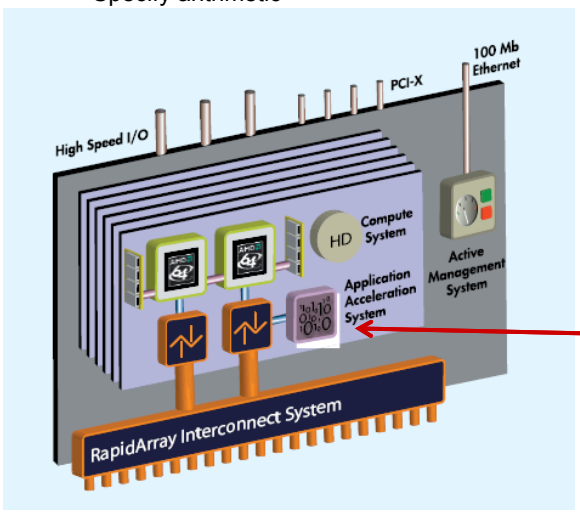
6,021 18,000 39,000 120,000 240,000

Condition number

41

Cray XD-1 (OctigaBay Systems)

Experiments with Field Programmable Gate Array
Specify arithmetic



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

42

Mixed Precision Iterative Refinement

- FPGA Performance Test - Junqing Sun

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

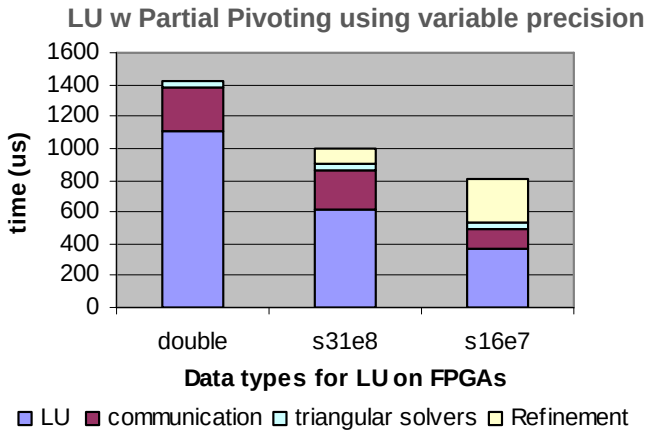
Refinement iterations for customized formats (sXXe11).
Random matrices

More Bits

Problem Size \ Mantissa Bits	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 JPDC

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.

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

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

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

Alfredo Buttari, ENS/INRIA
Julien Langou, UColorado
Julie Langou, UTK
Piotr Luszczek, MathWorks
Jakub Kurzak, UTK
Stan Tomov, UTK