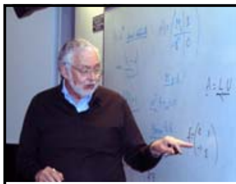


Stanford 50: State of the Art and Future Directions of Computational Mathematics and Numerical Computing

A conference celebrating the 50th anniversary of George Forsythe's arrival at Stanford and the 75th birthday of Professor Gene Golub.

The Challenge of Multicore and Specialized Accelerators for Mathematical Software

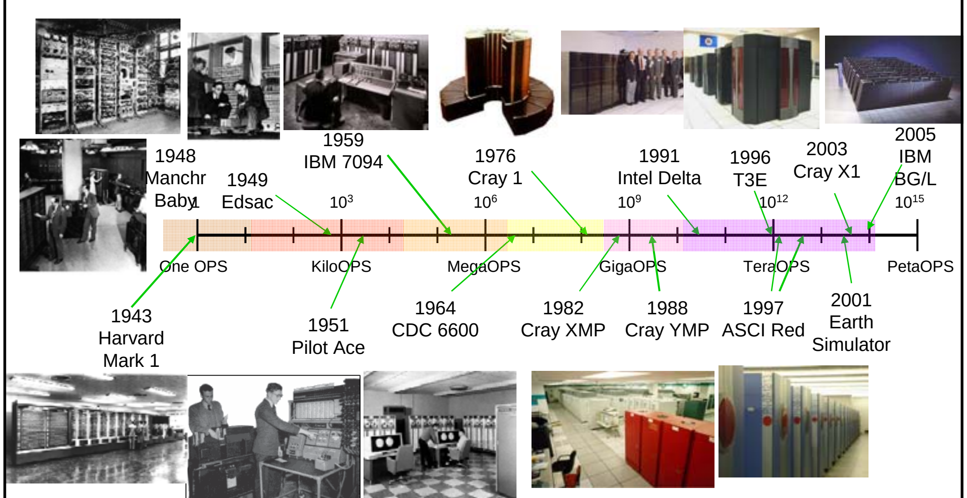
Jack Dongarra

Alfredo Buttari, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, Stan Tomov

University of Tennessee
and
Oak Ridge National Laboratory

1

A Growth-Factor of more than a Trillion in Performance in the Past 65 Years



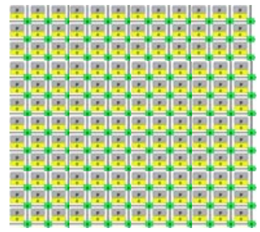
2

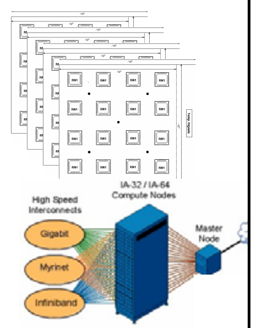
Scalar to super scalar to vector to SMP to DMP to massively parallel to many-core designs

Future Large Systems, Say in 5 Years

- 128 cores per socket
  - May be heterogeneous
- 32 sockets per node
- 128 nodes per system
- System = 128*32*128
  = 524,288 Cores!
- And by the way, its 4-8 threads of exec per core
- That's about 4M threads to manage

1 Chip =





Major Changes to Math Software

- Scalar
  - Fortran code in EISPACK
- Vector
  - Level 1 BLAS use in LINPACK
- SMP
  - Level 3 BLAS use in LAPACK
- Distributed Memory
  - Message Passing w/MPI in ScaLAPACK
- Many-Core
  - Event driven multi-threading in PLASMA
    - Parallel Linear Algebra Software for Multicore Architectures

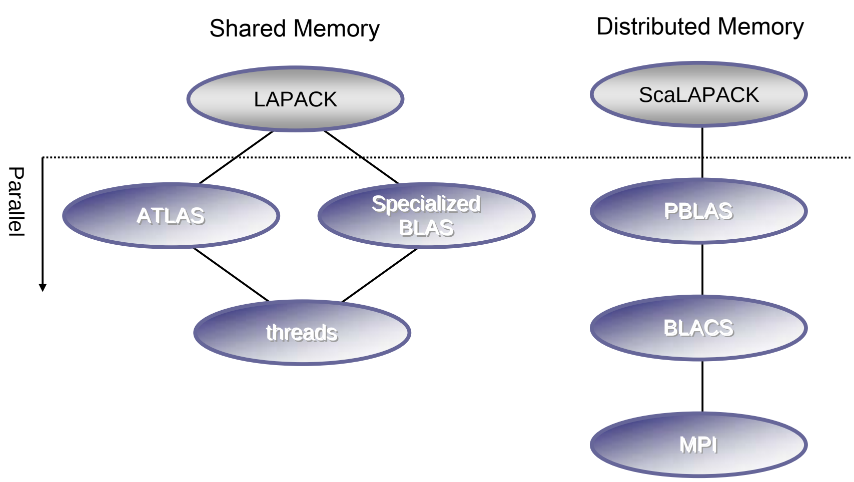
4

Time to Rethink Software Again

- 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

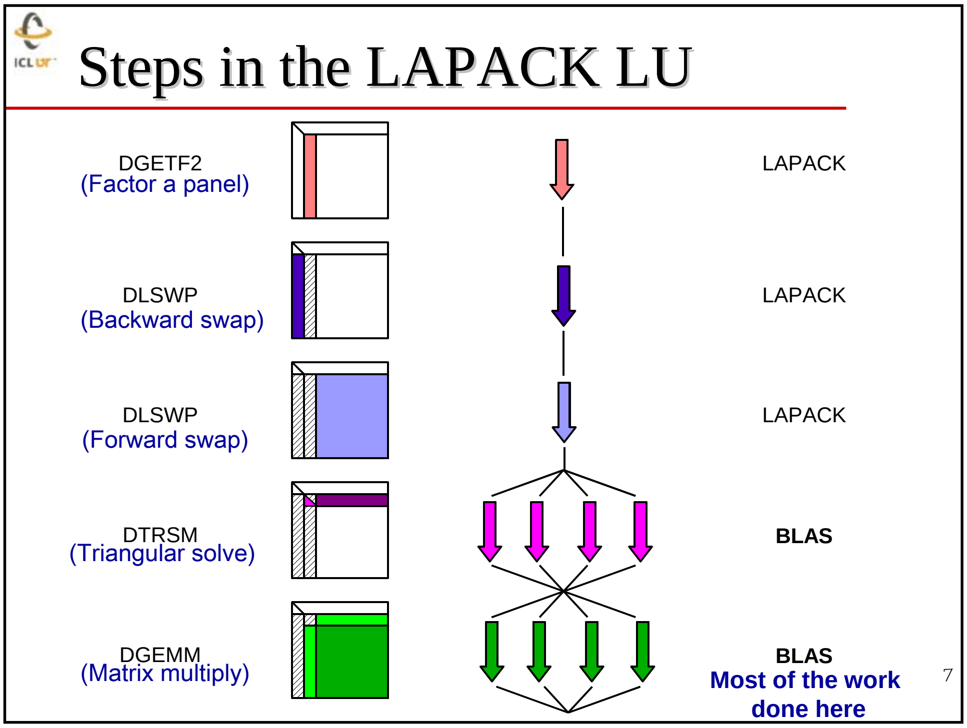
5

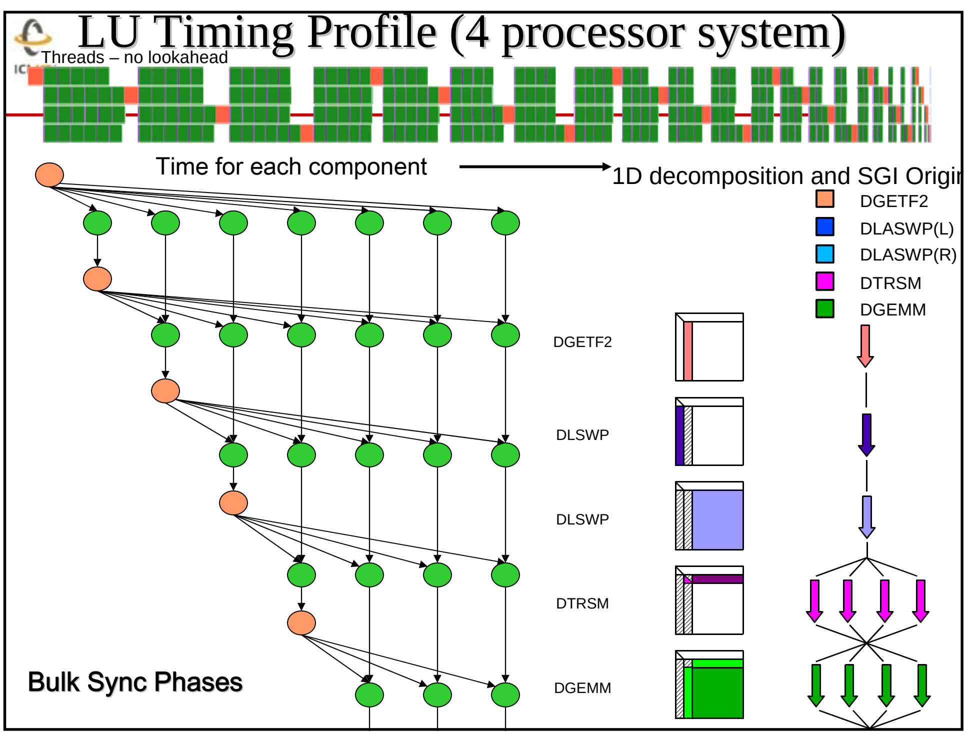
Parallelism in LAPACK / ScaLAPACK



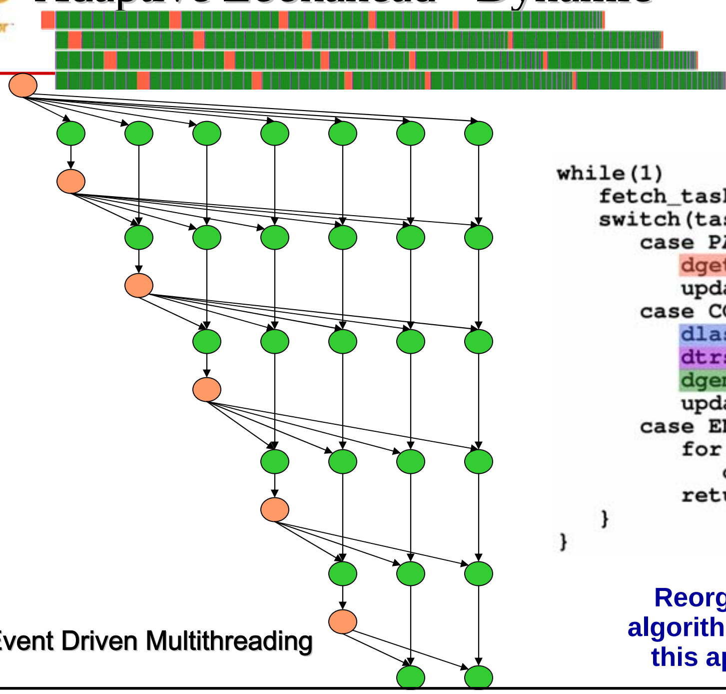
Two well known open source software efforts for dense matrix problems.

6


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
Most of the work done here
7


LU Timing Profile (4 processor system)
Threads – no lookahead
Time for each component
1D decomposition and SGI Origin
DGETF2
DLASWP(L)
DLASWP(R)
DTRSM
DGEMM
DGETF2
DLSWP
DLSWP
DTRSM
DGEMM
Bulk Sync Phases

Adaptive Lookahead - Dynamic



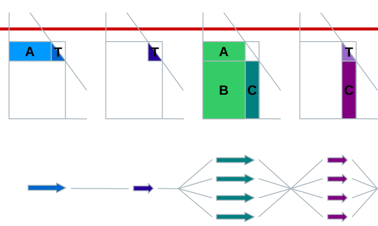
```
while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:
            dgetf2();
            update_progress()
        case COLUMN:
            dlaswp();
            dtrsm();
            dgemm();
            update_progress()
        case END:
            for()
                dlaswp();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

Event Driven Multithreading

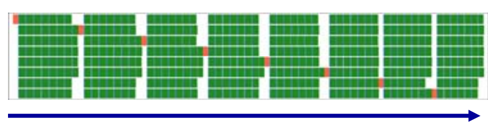
Reorganizing algorithms to use this approach

9

Fork-Join vs. Dynamic Execution



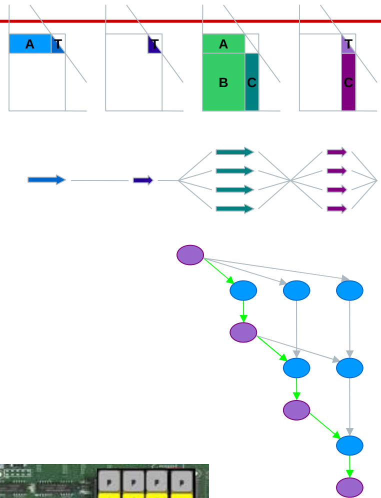
Fork-Join – parallel BLAS



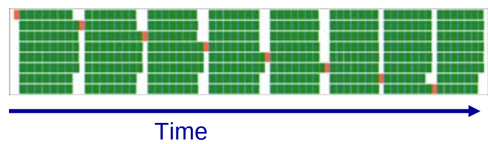
Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

10

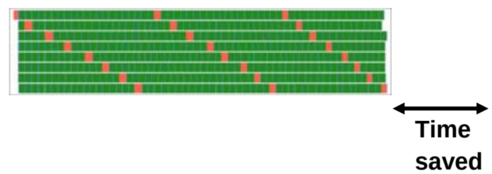
Fork-Join vs. Dynamic Execution

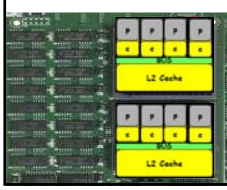


Fork-Join – parallel BLAS



DAG-based – dynamic scheduling





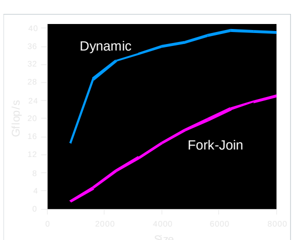
Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

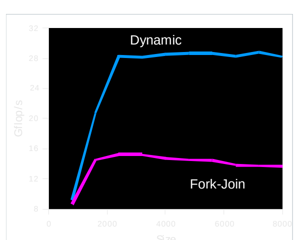
11

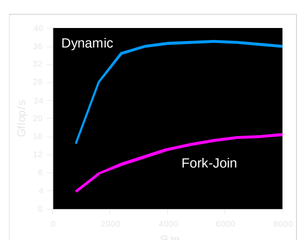
Fork-Join vs. Dynamic Execution

Breaking the "hour-glass" pattern of parallel processing

- LU Factorization
- Cholesky Factorization
- QR Factorization







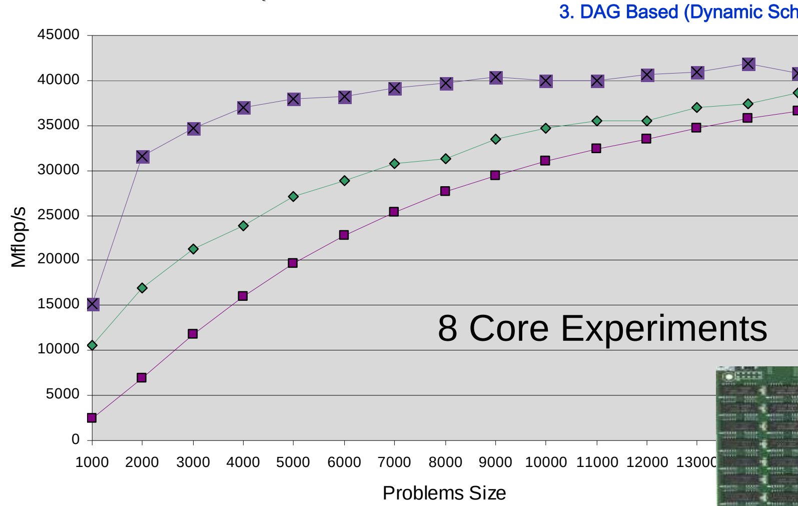
- **Intel Clovertown**
- **clock - 2.66 GHz**
- **2 sockets - quad-core**
- **8 cores total**
- **85 GFlop/s Theoretical Peak**

12

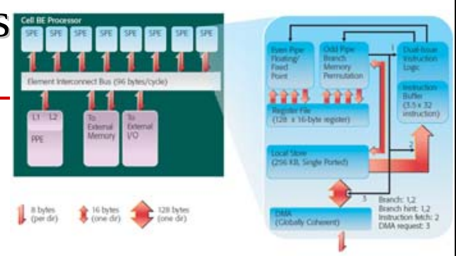
Intel's Clovertown Quad Core

3 Implementations of LU factorization
Quad core w/2 sockets per board, w/ 8 Treads

1. LAPACK (BLAS Fork-Join Parallelism)
2. ScaLAPACK (Mess Pass using mem copy)
3. DAG Based (Dynamic Scheduling)



What about the IBM's Cell Processor?



$600

- Power PC at 3.2 GHz
- 8 SPEs
  - 204.8 Gflop/s peak!
  - The catch is that this is for 32 bit floating point; (Single Precision SP)
  - And 64 bit floating point runs at 14.6 Gflop/s total for all 8 SPEs!!
    - Divide SP peak by 14; factor of 2 because of DP and 7 because of latency issues

The SPEs are fully IEEE-754 compliant in double precision.
In single precision, they only implement round-towards-zero.
PowerPC part is fully IEEE compliant.

14

On the Way to Understanding How to Use the Cell Something Else Happened …

- Realized have the similar situation on our commodity processors.
  - That is, SP is 2X as fast as DP on many systems
- Standard 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	Speedup SGEMM/ DGEMM	Size	Speedup SGEMV/ DGEMV
AMD Opteron 246	3000	2.00	5000	1.70
Sun 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

Two things going on:
- SP has higher execution rate and
- Less data to move.

15

Idea 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.

16

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; $O(10^8)$

17

In Matlab on My Laptop!

- Matlab has the ability to perform 32 bit floating point for some computations
  - Matlab uses LAPACK and MKL BLAS underneath.

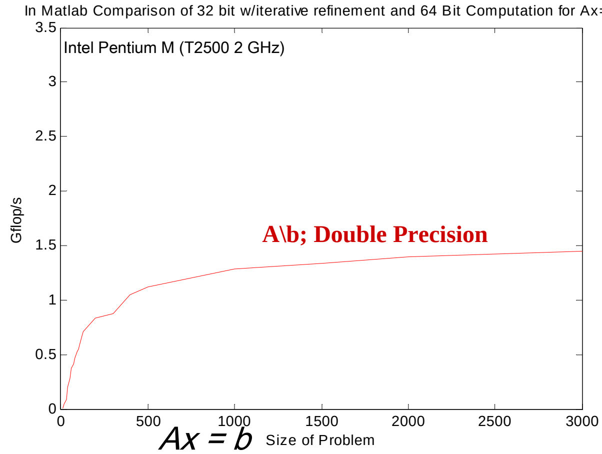
```
sa=single(a); sb=single(b);
[sl,su,sp]=lu(sa);                                   Most of the work: O(n3)
sx=su\(sl\(sp*sb)); x=double(sx); r=b-a*x;                             O(n2)
i=0;
while(norm(r)>res1),
   i=i+1;
   sr = single(r);
   sx1=su\(sl\(sp*sr)); x1=double(sx1); x=x1+x; r=b-a*x;               O(n2)
if (i==30), break; end;
```

- Bulk of work, $O(n^3)$, in “single” precision
- Refinement, $O(n^2)$, in “double” precision
  - Computing the correction to the SP results in DP and adding it to the SP results in DP.

18

Another Look at Iterative Refinement

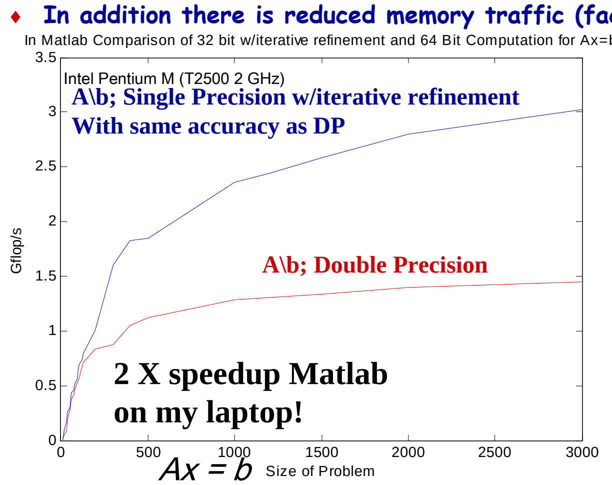
- **On a Pentium; using SSE2, single precision can perform 4 floating point operations per cycle and in double precision 2 floating point operations per cycle.**
- **In addition there is reduced memory traffic (for sp data)**



1.4 GFlop/s!
Not bad for Matlab

Another Look at Iterative Refinement

- **On a Pentium; using SSE2, single precision can perform 4 floating point operations per cycle and in double precision 2 floating point operations per cycle.**
- **In addition there is reduced memory traffic (factor on sp data)**



3 GFlop/s!!

Speedups for Ax = b (Ratio of Times)

Architecture (BLAS)	*n*	DGEMM /SGEMM	DP Solve /SP Solve	DP Solve /Iter Ref	# iter
Intel Pentium III Coppermine (Goto)	3500	2.10	2.24	1.92	4
Intel Pentium IV Prescott (Goto)	4000	2.00	1.86	1.57	5
AMD Opteron (Goto)	4000	1.98	1.93	1.53	5
Sun UltraSPARC IIe (Sunperf)	3000	1.45	1.79	1.58	4
IBM Power PC G5 (2.7 GHz) (VecLib)	5000	2.29	2.05	1.24	5
Cray X1 (libsci)	4000	1.68	1.57	1.32	7
Compaq Alpha EV6 (CXML)	3000	0.99	1.08	1.01	4
IBM SP Power3 (ESSL)	3000	1.03	1.13	1.00	3
SGI Octane (ATLAS)	2000	1.08	1.13	0.91	4

Recent addition to LAPACK 3.1 as DSGESV

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

21

Quadruple Precision

n	Quad Precision Ax = b	Iter. Refine. DP to QP	
	time (s)	time (s)	Speedup
100	0.29	0.03	9.5
200	2.27	0.10	20.9
300	7.61	0.24	30.5
400	17.8	0.44	40.4
500	34.7	0.69	49.7
600	60.1	1.01	59.0
700	94.9	1.38	68.7
800	141.	1.83	77.3
900	201.	2.33	86.3
1000	276.	2.92	94.8

Intel Xeon 3.2 GHz

Reference implementation of the quad precision BLAS

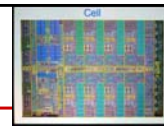
Accuracy: 10^{-32}

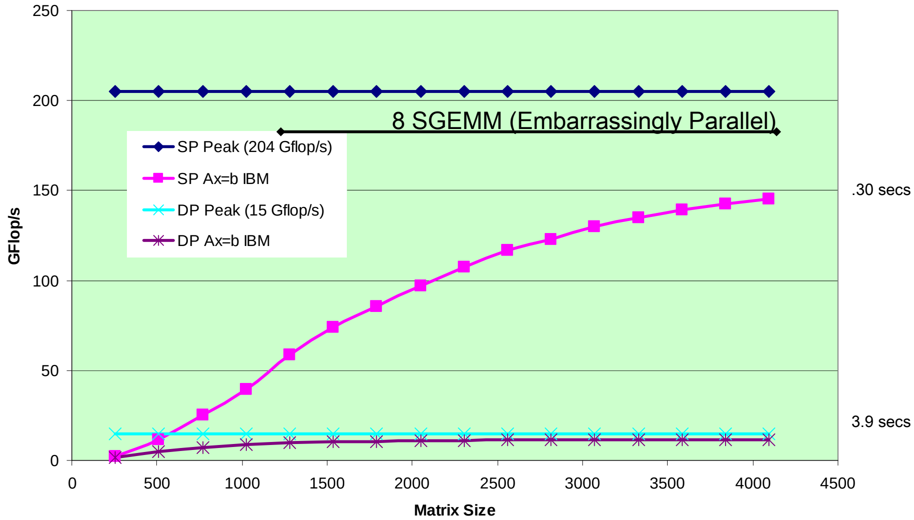
No more than 3 steps of iterative refinement are needed.

- Variable precision factorization (with say < 32 bit precision) plus 64 bit refinement produces 64 bit accuracy

22

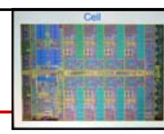
IBM Cell 3.2 GHz, Ax = b

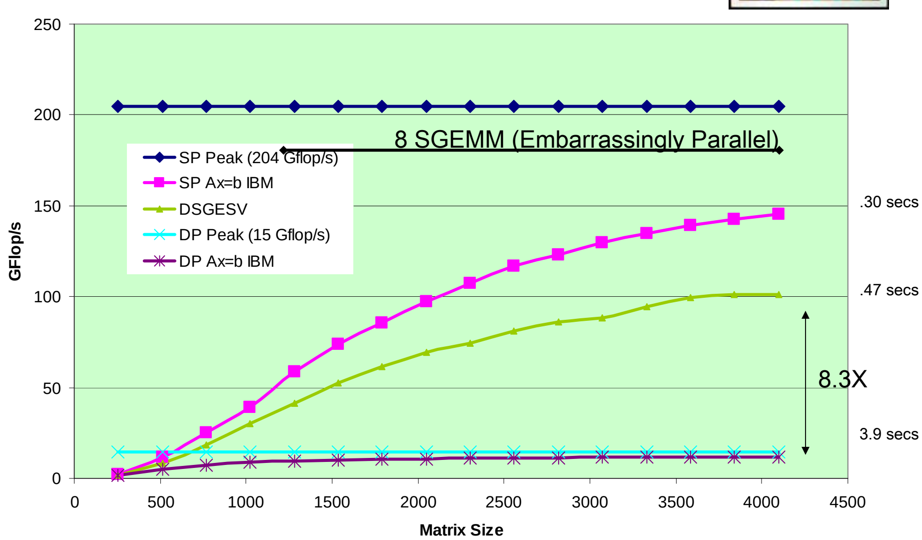




23

IBM Cell 3.2 GHz, Ax = b





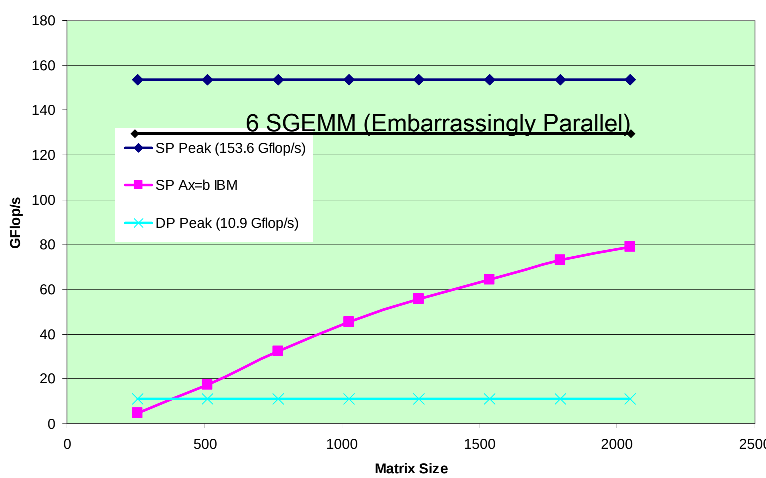
24

Sony Playstation 3 Cluster PS3-T

- From IBM or Mercury
  - 2 Cell chip
    - Each w/8 SPEs
  - 512 MB/Cell
  - ~$17K
  - Some SW
- From WAL*MART PS3
  - 1 Cell chip
    - w/6 SPEs
  - 256 MB/PS3
  - $600
  - Download SW
  - Dual boot

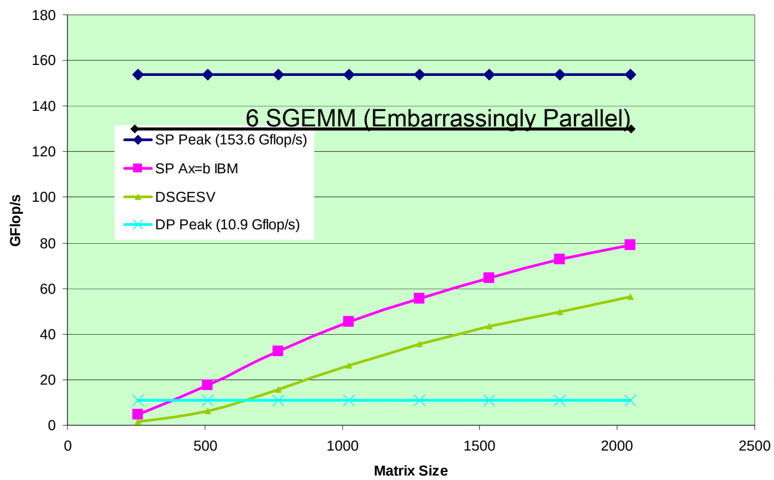
25

PlayStation 3 LU Codes



26

PlayStation 3 LU Codes



27

Refinement Technique Using Single/Double Precision

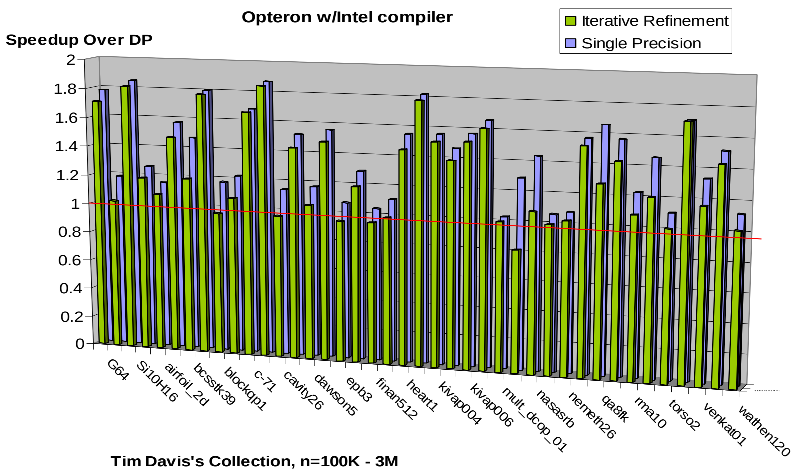
- **Dense Linear Systems**
  - LU dense (in current release of LAPACK)
  - Cholesky
  - QR Factorization
- **Sparse Direct Method**
  - When kernel matrix multiple
  - multifrontal approach - MUMPS
- **Iterative Linear System**
  - Relaxed GMRES
  - Inner/outer iteration scheme

See webpage for tech report which discusses this.

28

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

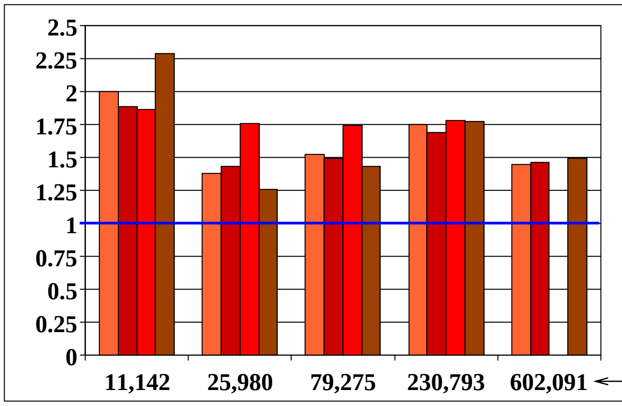
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif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

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 fixed number of inner iteration in 32 bit floating point**

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

Time speedups for mixed precision Inner SP/Outer DP (SP/DP) iter. methods *vs* DP/DP
(CG, GMRES, PCG, and PGMRES with diagonal preconditioners)



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

Reference methods

- **CG**
- **PCG**
- **GMRES**
- **PGMRES**

(*More is better*)

← Matrix size

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

Data movement the main source of improvement

31

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?
  - 128 bit floating point?
- **Linear systems and Eigenvalue, optimization problems, where Newton's method is used.**

32

Happy Birthday Gene!