Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy

(Revisiting Iterative Refinement for Linear Systems)

Julie Langou
Piotr Luszczek
Alfredo Buttari

Julien Langou
Jakub Kurzak
Jack Dongarra

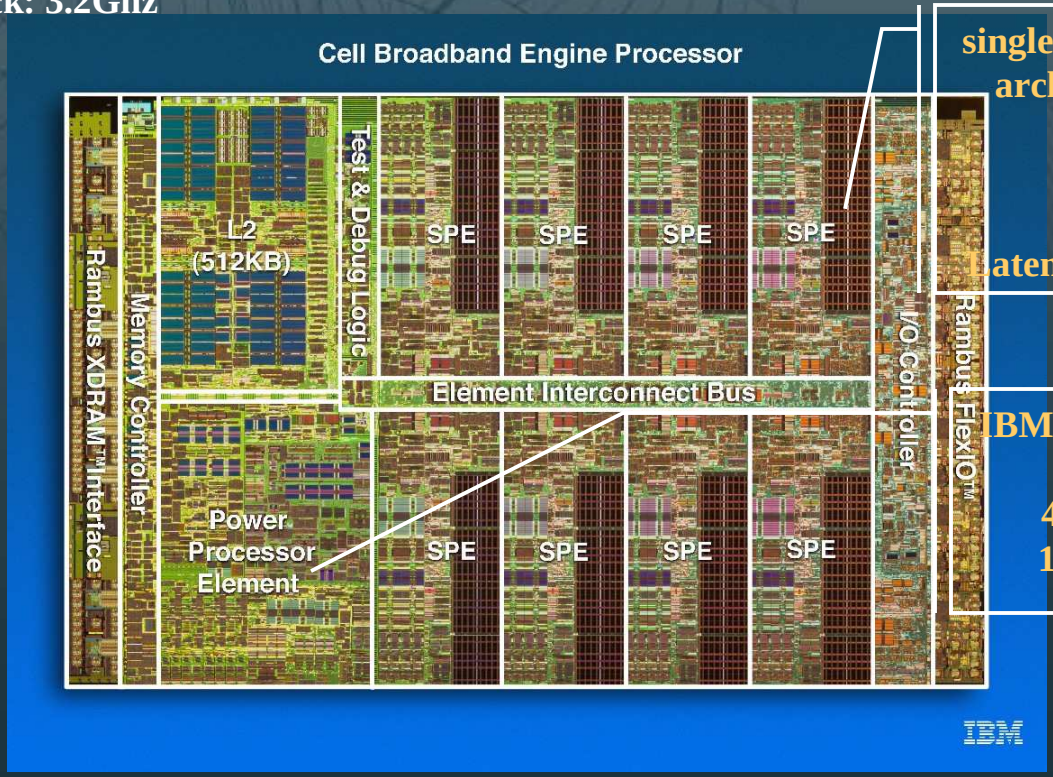
ICL UT

Innovative Computing Laboratory
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE

Friday Lunch May 19th 2006

Architecture of the cell

Cell is a heterogeneous chip multiprocessor that consists of an IBM 64-bit Power Architecture™ core, augmented with eight specialized co-processors based on a novel single-instruction multiple-data (SIMD) architecture.
Clock: 3.2Ghz



single-instruction multiple-data (SIMD) architecture : Synergistic Processor Unit (SPU)
Vector:128 bits
Fused add-multiply
Latency for double operations: 6 cycles

IBM 64-bit Power Architecture™ core
AltiVec/VMX Vector Unit
4 multiply-add /cycle for single
1 multiply-add/cycle for double

http://domino.research.ibm.com/comm/research.nsf/pages/r.arch.innovation.html
http://www.research.ibm.com/cell/cell_chip.html
http://www-128.ibm.com/developerworks/power/library/pa-cellperf/

Performance of the cell

This is how single precision performance breaks down for the CBE by "Jakub"

- The vector unit of one SPE can do one vector operation each cycle.
  A vector is 128 bits and can hold 4 floats.
  So, with fused add-multiply, you can have 8 single-precision floating point operation per cycle.
  The target clock is 3.2 GHz (our cell is 2.1), so you get 3.2 * 8 = 25.6 GFLOPS on a single SPU. You have 8 SPUs, so in total you can get 8 * 25.6 = 204.8 GFLOPS from the 8 SPUs.
  However the PPU has an Altivec/VMX vector unit which can also do 4 multiply-add operations per cycle.
  So if you add it all together, the total single-precision performance of the whole CBE is 230.4 GFLOPS.
- Double precision can also be processed in a vectorized fashion on an SPE, but a vector can store only 2 64-bit doubles, so performance is cut by 2.
  Additionally, the double precision operations are not fully pipelined, so one operation has a latency of 6 cycles, so you are 6 times slower.
  So in total you are 6*2=12 times slower for a double precision operation.
  And also the Altivec/VMX on the PPU cannot process doubles in a vectorized fashion, so on the PPU you rely on the standard floating point unit which should still do a single multiply-add in one cycle.
- To sum up, for double precision on a 3.2 GHz chip, you have 2.13 GFLOPS on a single SPU, so 2.13 * 8 = 17 in total, plus 6.4 on the PPU = 23.47 total. So, around 230 GFLOPS single precision and around 23 GFLOPS double precision

- Peak performance for single precision: ~230 GFlops
- Peak performance for double precision: ~24 GFlops

Problematic on the cell

- Single are 10 times faster than double!
- But single are 2 times less accurate than double

Can we get double accuracy results with a performance not too far from the peak of single precision?

Jack's idea: use iterative refinement!

- Factorize the matrix in single precision, $O(n^3)$ FLOP
- Then use this factorization as a preconditioner in a double precision iterative method, $O(n^2)$ FLOP

- The single LU factorization is a POOR double LU factorization but an EXCELLENT preconditioner

  Typically the iterative methods will converge in few steps

- $O(n^3)$ FLOP vs $O(n^2)$: single computations dominate the # of FLOP:
  - Speed of Single precision
- Convergence in a double precision iterative solver:
  - Accuracy in Double precision

23 years back in Jack's life:

- Improving the Accuracy of Computed Eigenvalues and Eigenvectors, J. J. Dongarra, C. B. Moler and J. H. Wilkinson, SIAM Journal on Numerical Analysis 20(1):23-45, February 1983, ISSN 0036-1429.
- Improving the Accuracy of Computed Singular Values, J. J. Dongarra, SIAM Journal on Scientific and Statistical Computing 4(4):712-719, December 1983, ISSN 0196-5204.
- Algorithm 589: SICEDR: A FORTRAN Subroutine for Improving the Accuracy of Computed Matrix Eigenvalues, J. J. Dongarra, ACM Transactions on Mathematical Software 8(4):371-375, December 1982, ISSN 0098-3500.

Introduction to Iterative Refinement

Algorithm

- L U = lu(A) %LU factorization (SINGLE) $O(n^3)$
- x = L\(U\b) % Solve (SINGLE) $O(n^2)$
- r = b – Ax % Residual (DOUBLE) $O(n^2)$ (DOUBLE) $O(n^2)$
- while (|| r || not small enough), %stopping criteria
  - z = L\(U\r) %LU factorization on the residual (SINGLE) $O(n^2)$
  - x = x + z % new solution (DOUBLE) $O(n^2)$
  - r = b - Ax % new residual (DOUBLE) $O(n^2)$
- End

- COST: (SINGLE) $O(n^3)$ + #ITER * (DOUBLE) $O(n^2)$

limitation

$$c(n) \cdot \varepsilon_s \cdot \kappa(A) < 1$$

limitation

$$c(n) \cdot \varepsilon_s \cdot \kappa(A) < 1$$

$\Rightarrow$ the condition number of A cannot be too large

(typically $\kappa(A) < 10^8$ for this scheme as opposed to $\kappa(A) < 10^{16}$ for regular double LU)

Limitation also with the range of the numbers used (overflow quicker in single)

Number of steps

- Approximately …
- double precision: $\varepsilon_d = 10^{-16}$ -> $t_d = -\log_{10}(\varepsilon_d) = 16$
- single precision: $\varepsilon_s = 10^{-8}$ -> $t_s = -\log_{10}(\varepsilon_s) = 8$
- condition number: $\kappa(A) = 10^4$ -> $t_k = \log_{10}(\kappa(A)) = 4$

- Max # of steps = td/(ts-tk)
  = 16/(8-4)
  = 4

what about your laptop ...

Single vs Double

Architectures (BLAS/LAPACK)	n	DGEMM /SGEMM	DGETRF /SGETRF
Cell	-	10	10
Intel Pentium III Coppermine (Goto)	3500	2.10	2.24
Intel Pentium III Katmai (Goto)	3000	2.12	2.11
Sun UltraSPARC IIe (Sunperf)	3000	1.45	1.79
Intel Pentium IV Prescott (Goto)	4000	2.00	1.86
Intel Pentium IV-M Northwood (Goto)	4000	2.02	1.98
AMD Opteron (Goto)	4000	1.98	1.93
Cray X1 (libsci)	4000	1.68	1.54
IBM Power PC G5 (2.7 GHz) (VecLib)	5000	2.29	2.05
Compaq Alpha EV6 (CXML)	3000	0.99	1.08
IBM SP Power3 (ESSL)	3000	1.03	1.13
SGI Octane (ATLAS)	2000	1.08	1.13
Intel Itanium 2 (Goto and ATLAS)	1500	0.71	

OPERATION PER CYCLE TIME

Processor	Single			Double	
	x87	SSE	3DNOW!	x87	SSE2
Pentium	1	0	0	1	0
Pentium II	1	0	0	1	0
Pentium III	1	4	0	1	0
Pentium 4	1	4	0	1	2
Athlon	2	0	4	2	0
Enhanced Athlon	2	0	4	2	0
Athlon4	2	4	4	2	0
AthlonMP	2	4	4	2	0

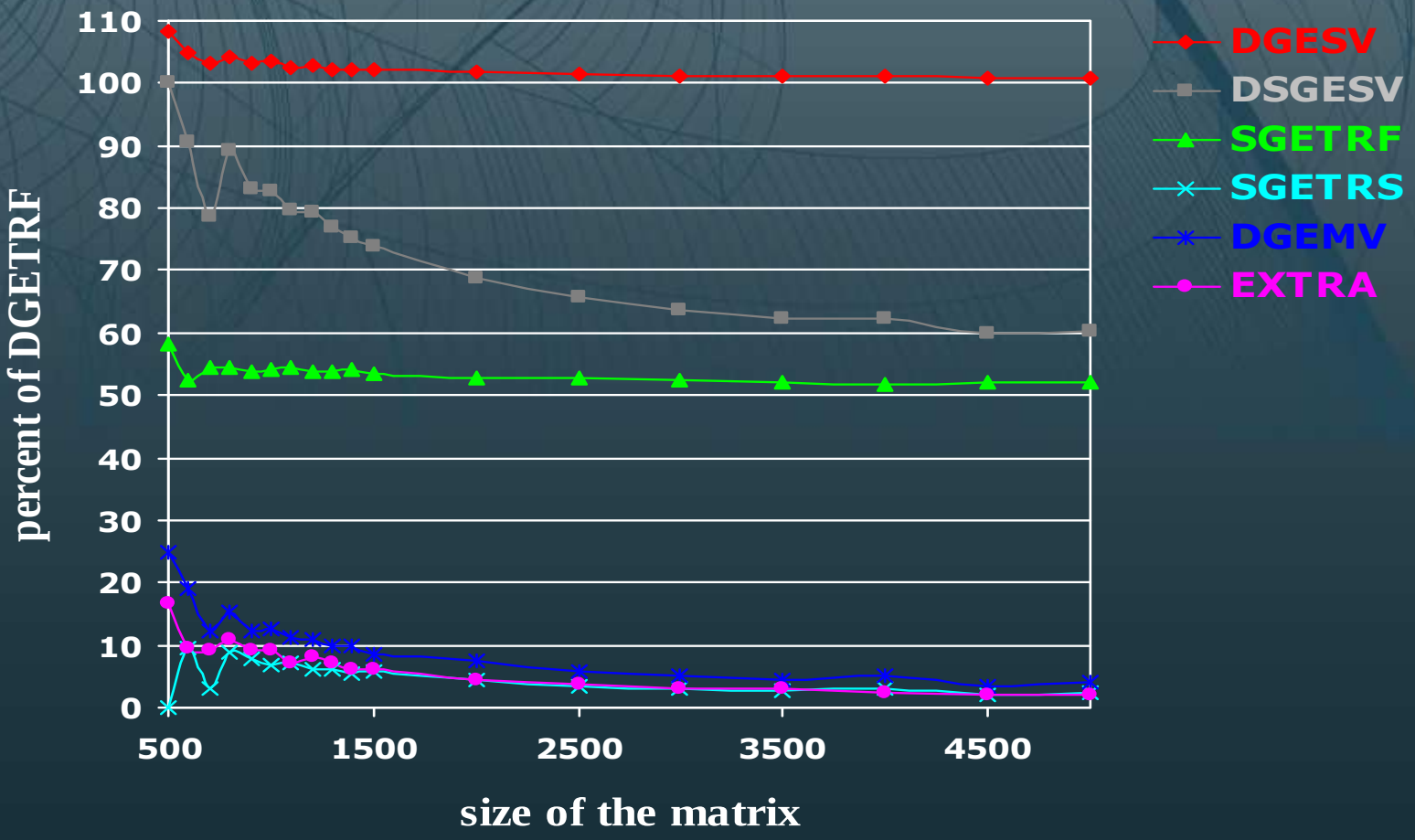
MMX : Set of "MultiMedia eXtensions" to the x86 ISA. Mainly new instructions for integer performance, and maybe some prefetch. For Intel, all chips starting with the PentiumMMX processor possess these extensions. For AMD, all chips starting with the K6 possess these extensions.

SSE : Streaming SIMD (Single Instruction Multiple Data) Extensions. SSE is a superset of MMX.These instructions are used to speed up single precision (32 bit) floating point arithmetic. For Intel, all chips listed starting with the Pentium III possess SSE extensions. For AMD, all chips starting from Athlon4 possess SSE.

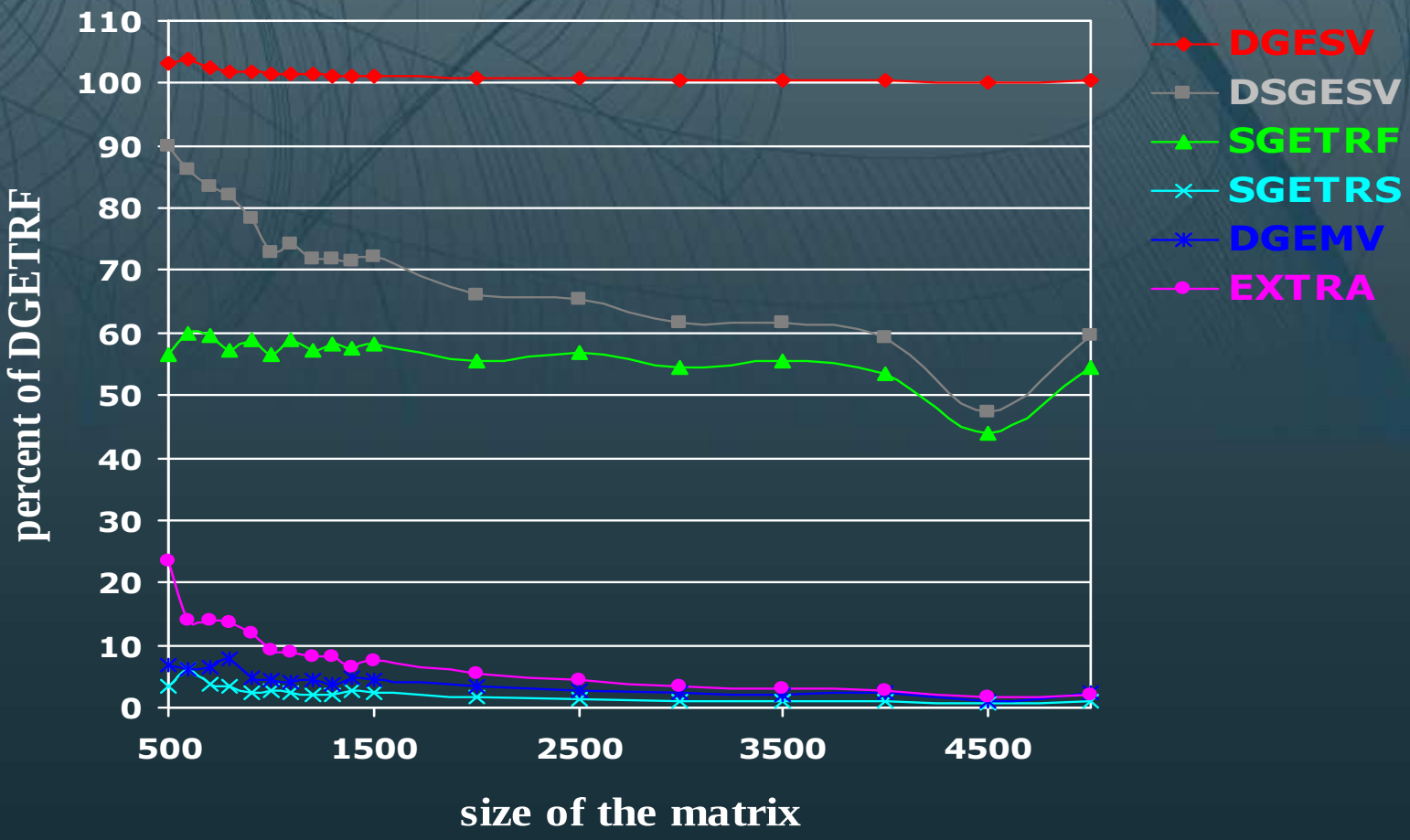
3DNow! :AMD's extension to MMX that does almost the exact same thing SSE does, except the single precision arithmetic is not IEEE compliant . It is also a superset of MMX (but not of SSE; 3DNow! was released before SSE). It is supported only on AMD, starting with the K6-2 chip.

SSE2 :Additional instructions that perform double precision floating arithmetic. Allows for 2 double precision FLOPs every cycle. For Intel, supported on the Pentium 4 and for AMD, supported on the Opteron.

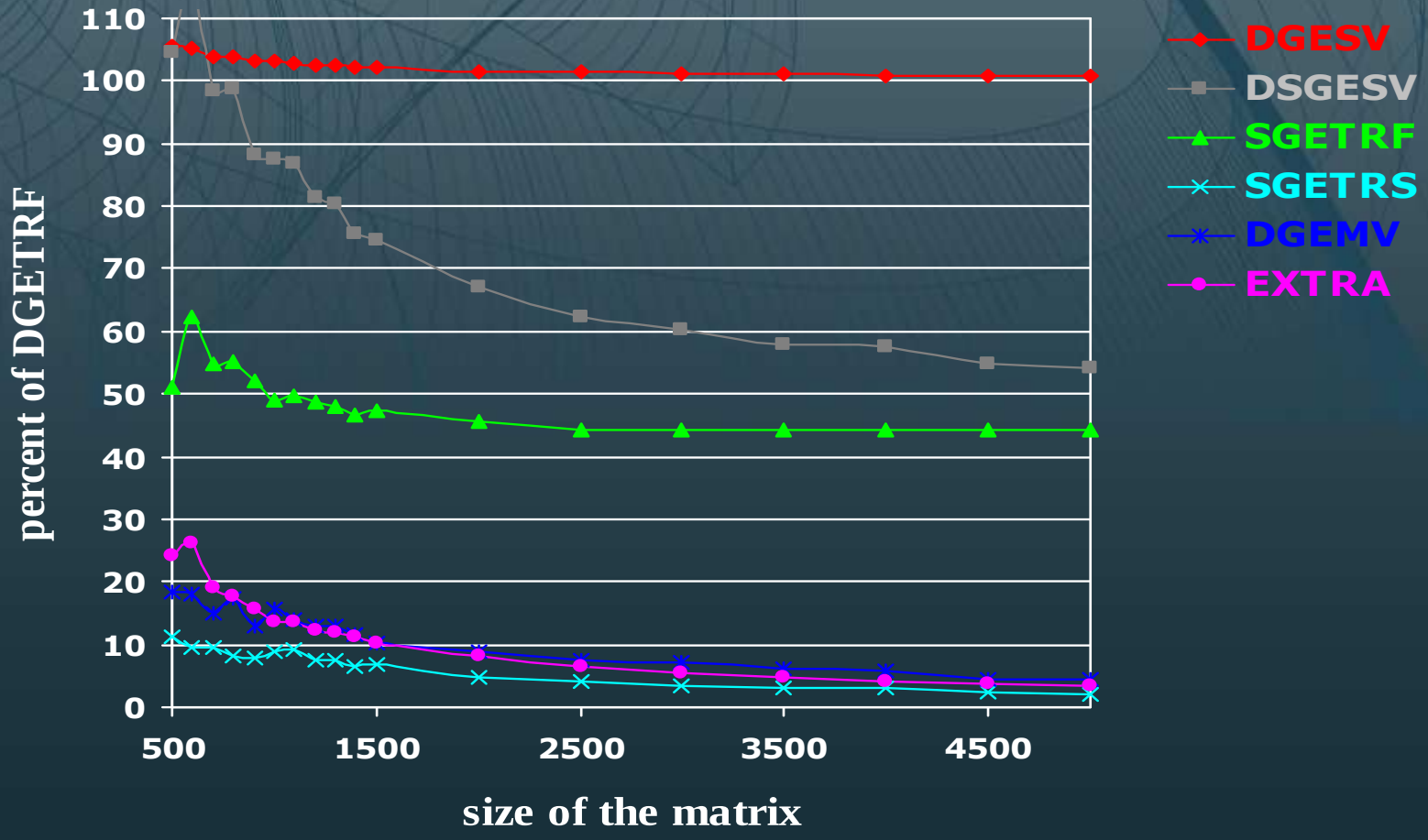
AMD Opteron(tm) Processor 240 (1.4GHz), Goto BLAS (1 thread)



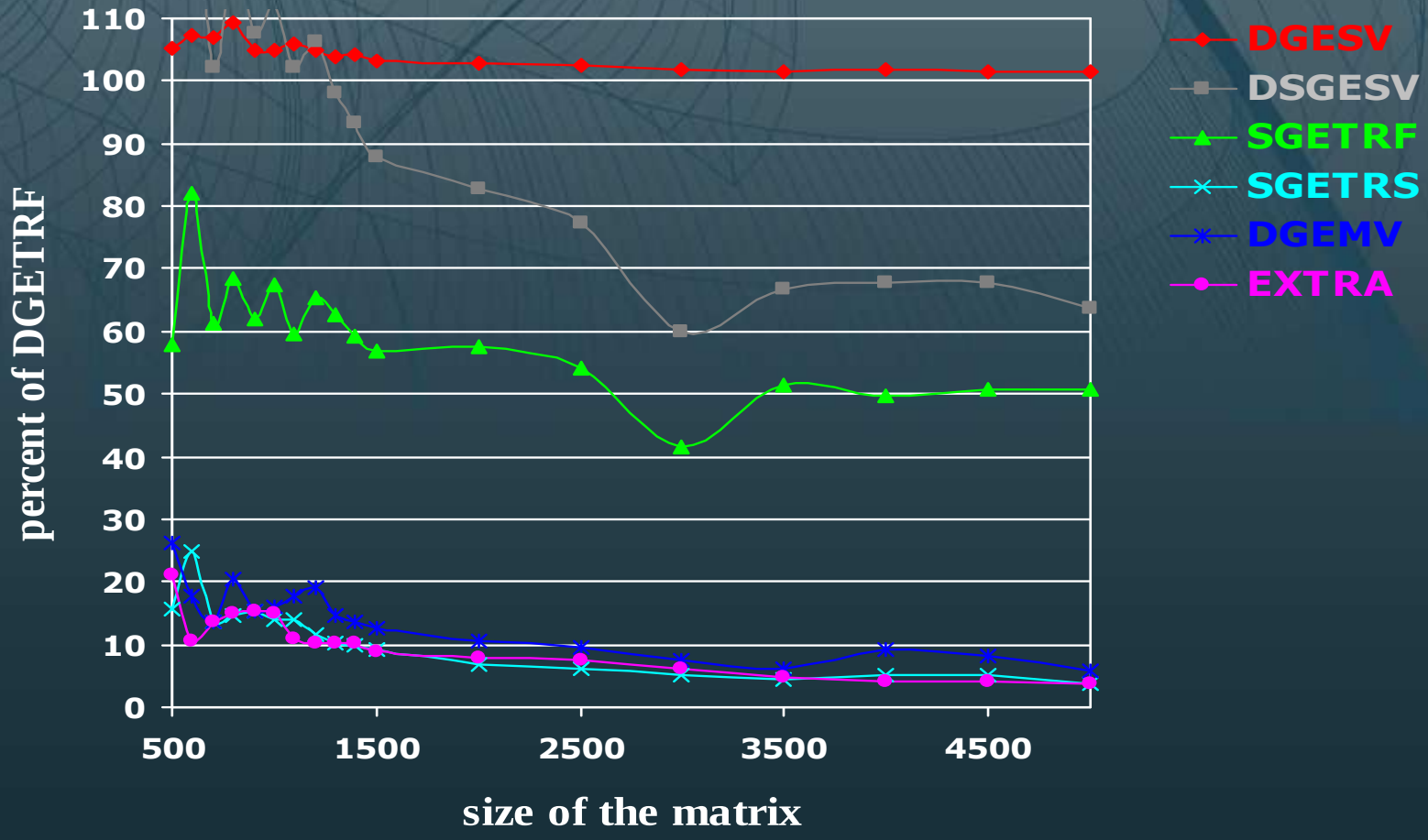
Sun UltraSPARC-IIe (502 MHz), SunPerf



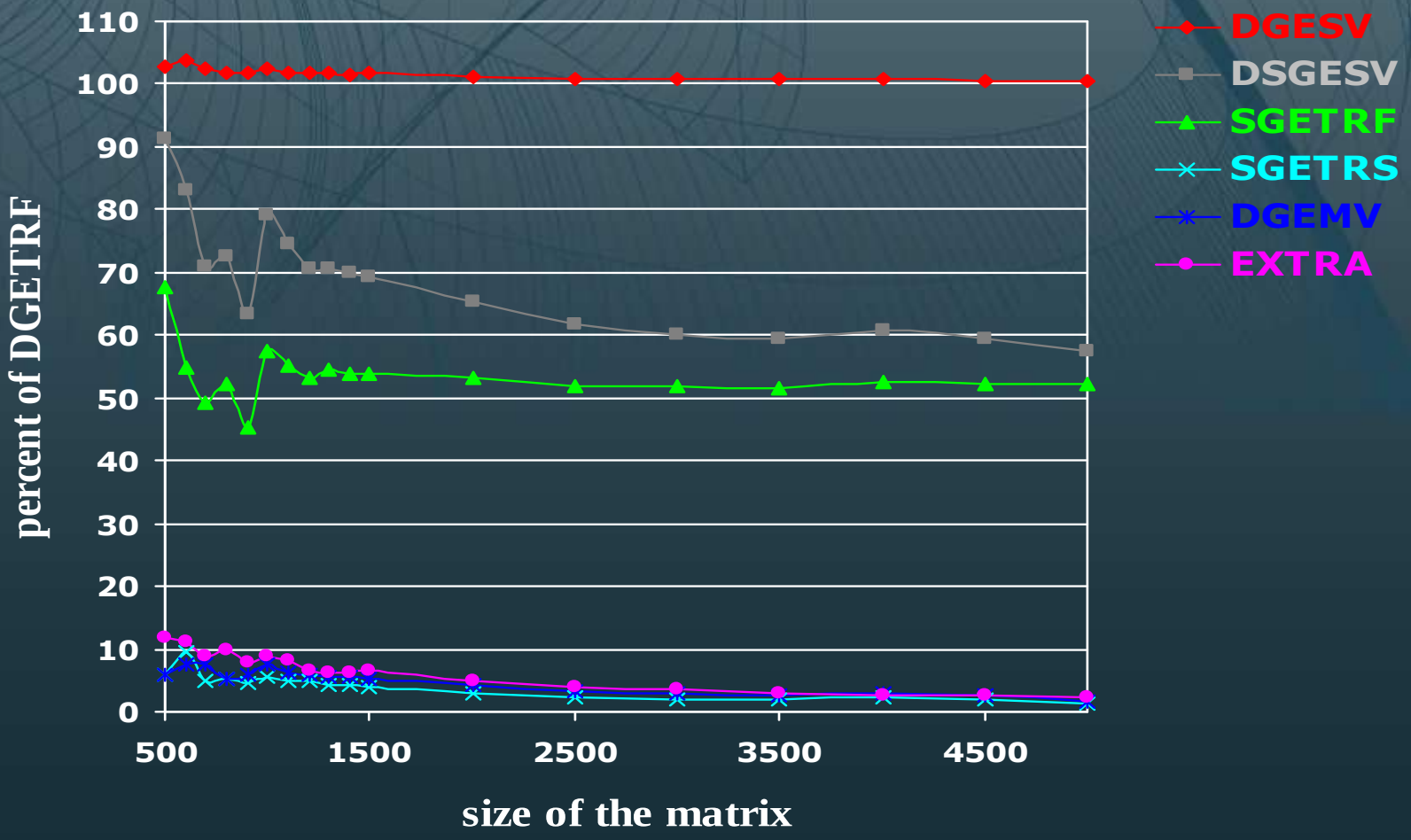
Intel Pentium III CopperMine (0.9GHz), Goto BLAS



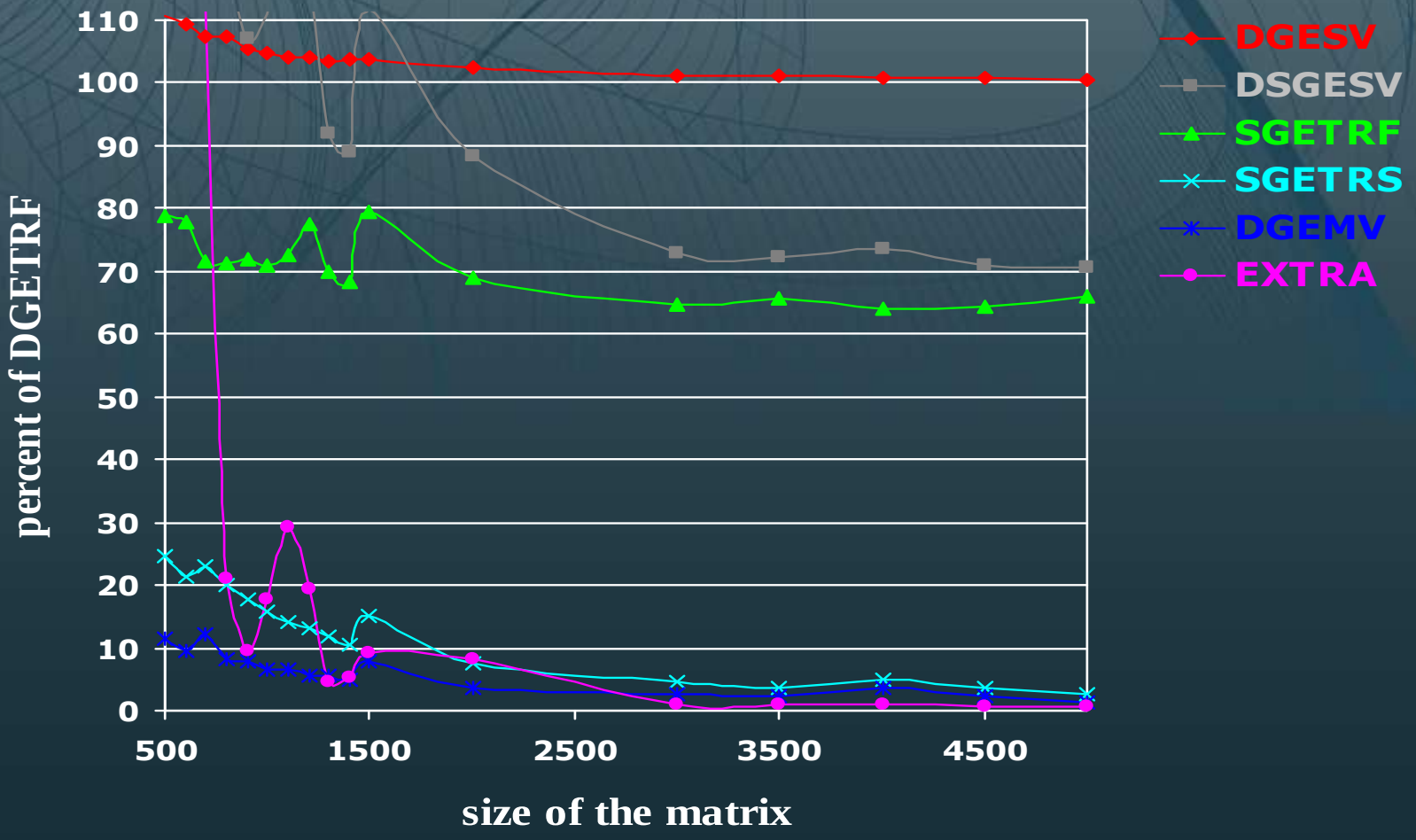
Intel Pentium Xeon Northwood (2.4GHz), Goto BLAS (1 thread)



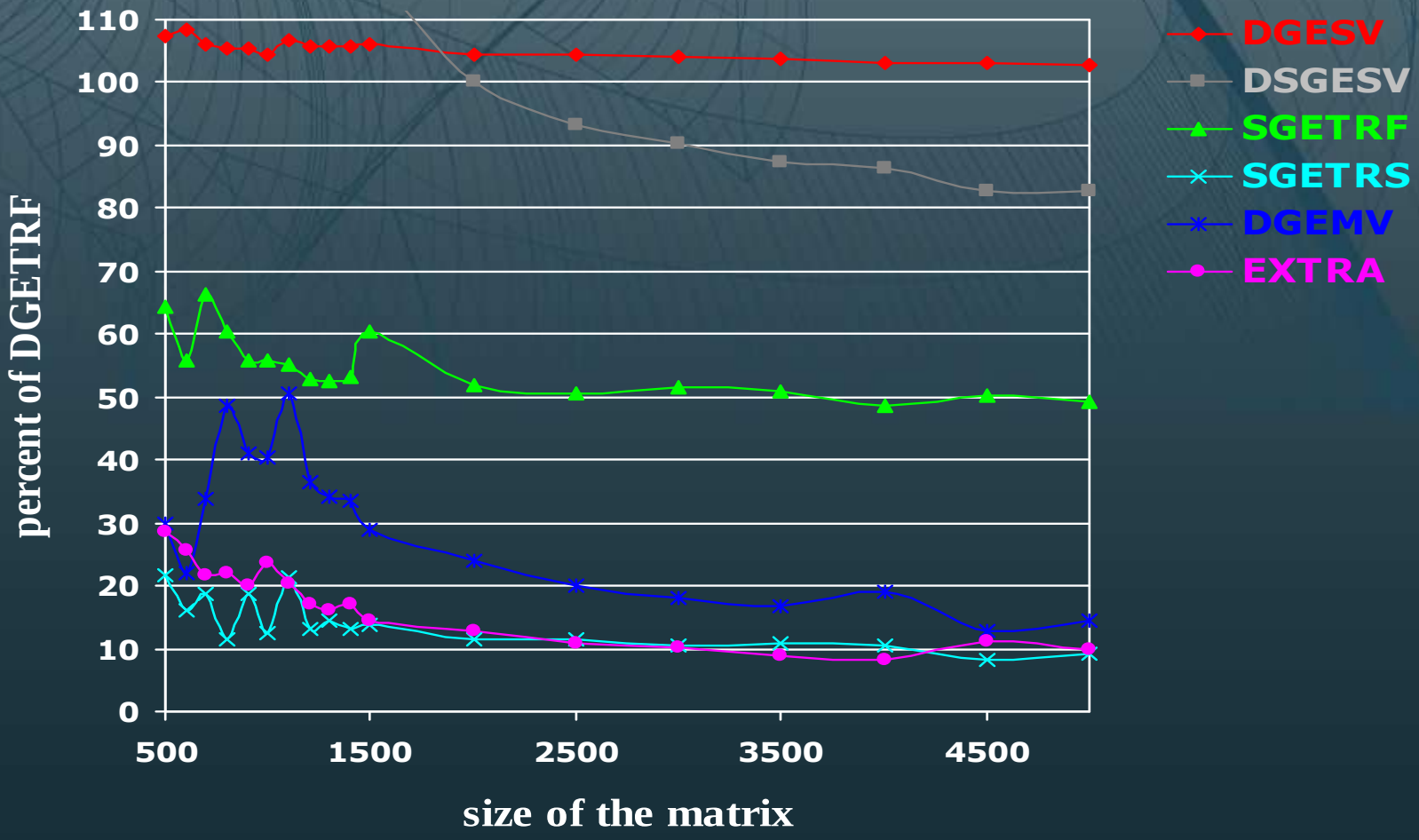
Intel Pentium IV Prescott (3.4GHz), Goto BLAS (1 thread)



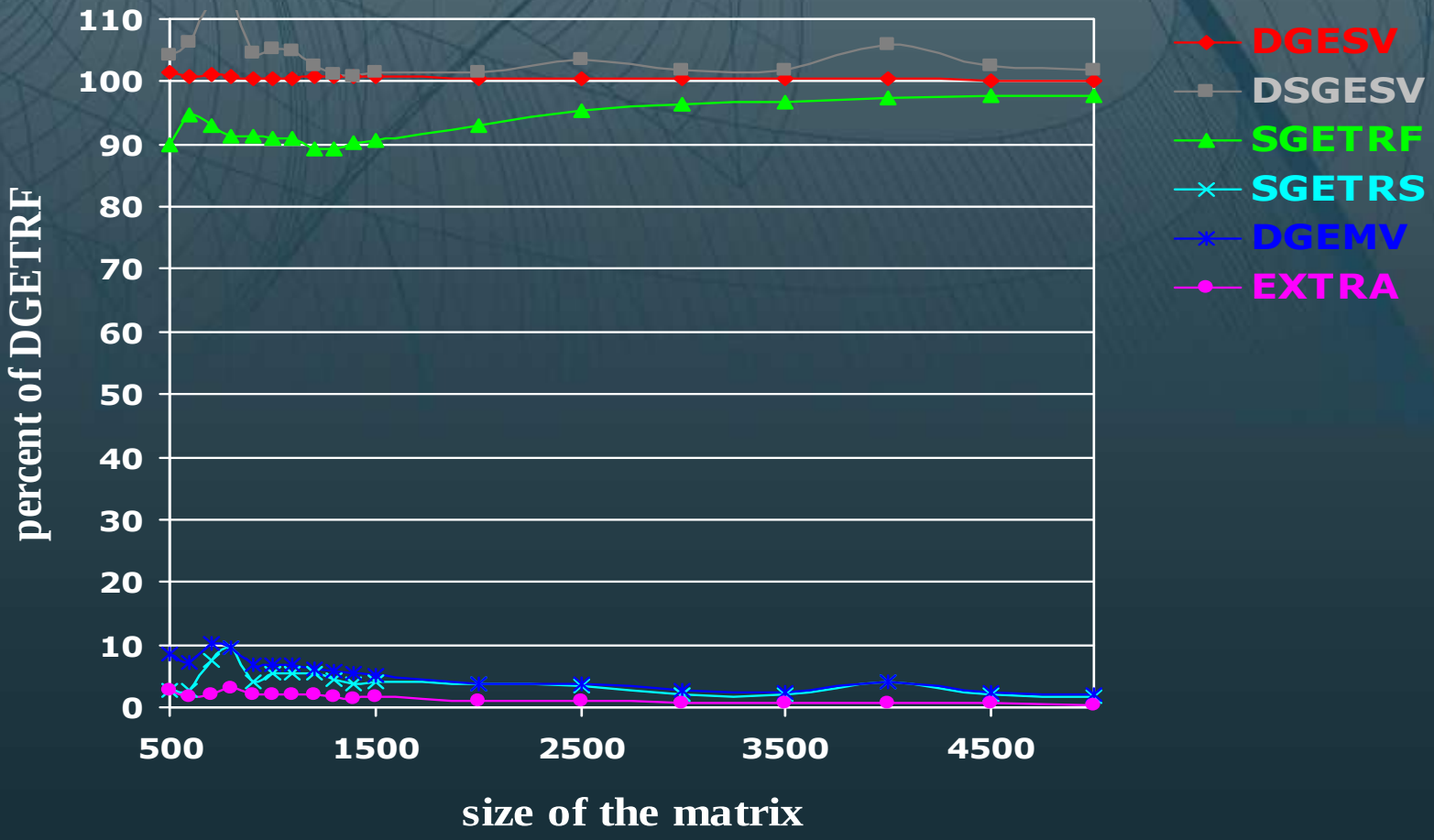
CRAY X1, libsci



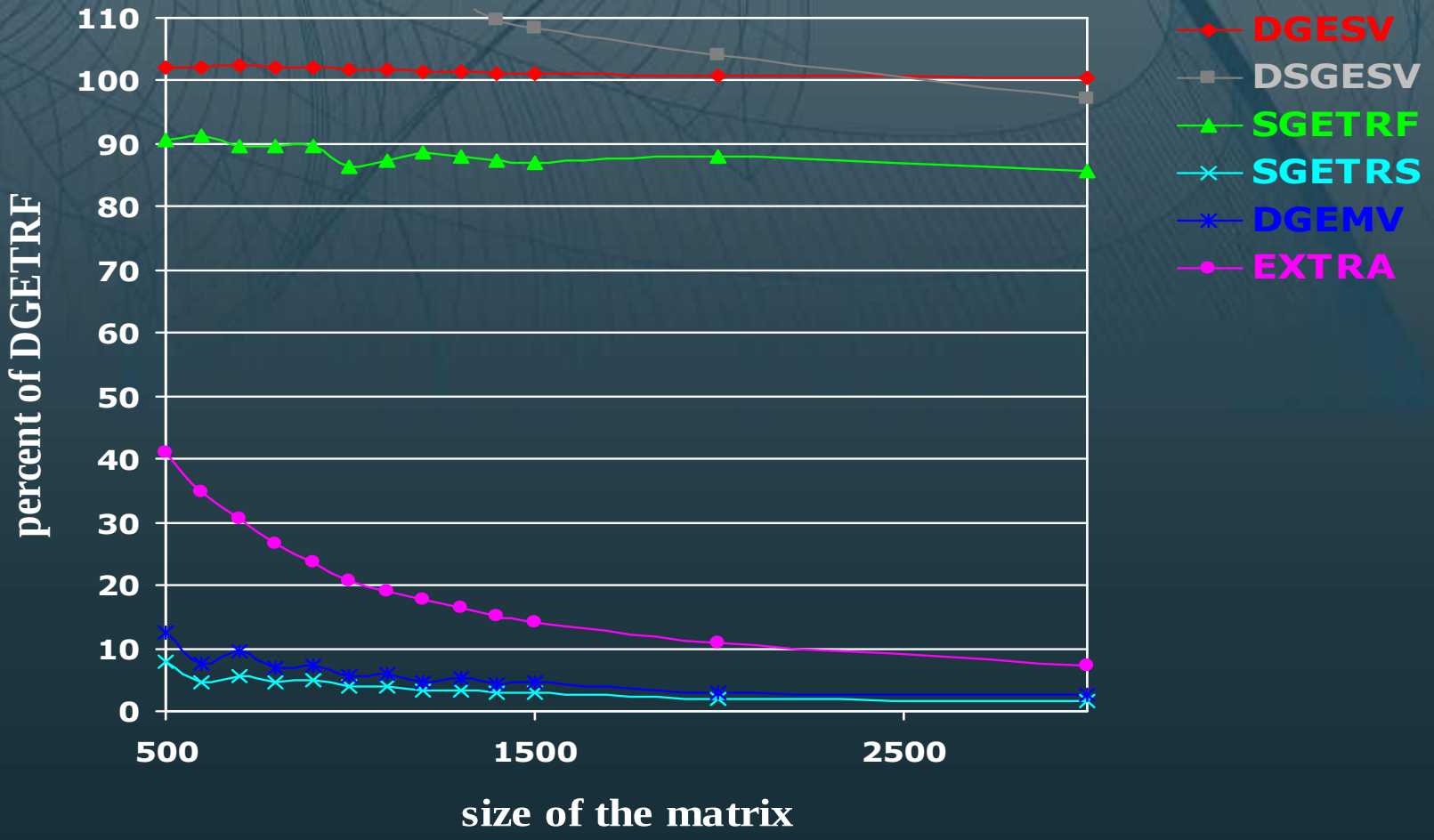
Power PC G5, vecLib (2 threads)



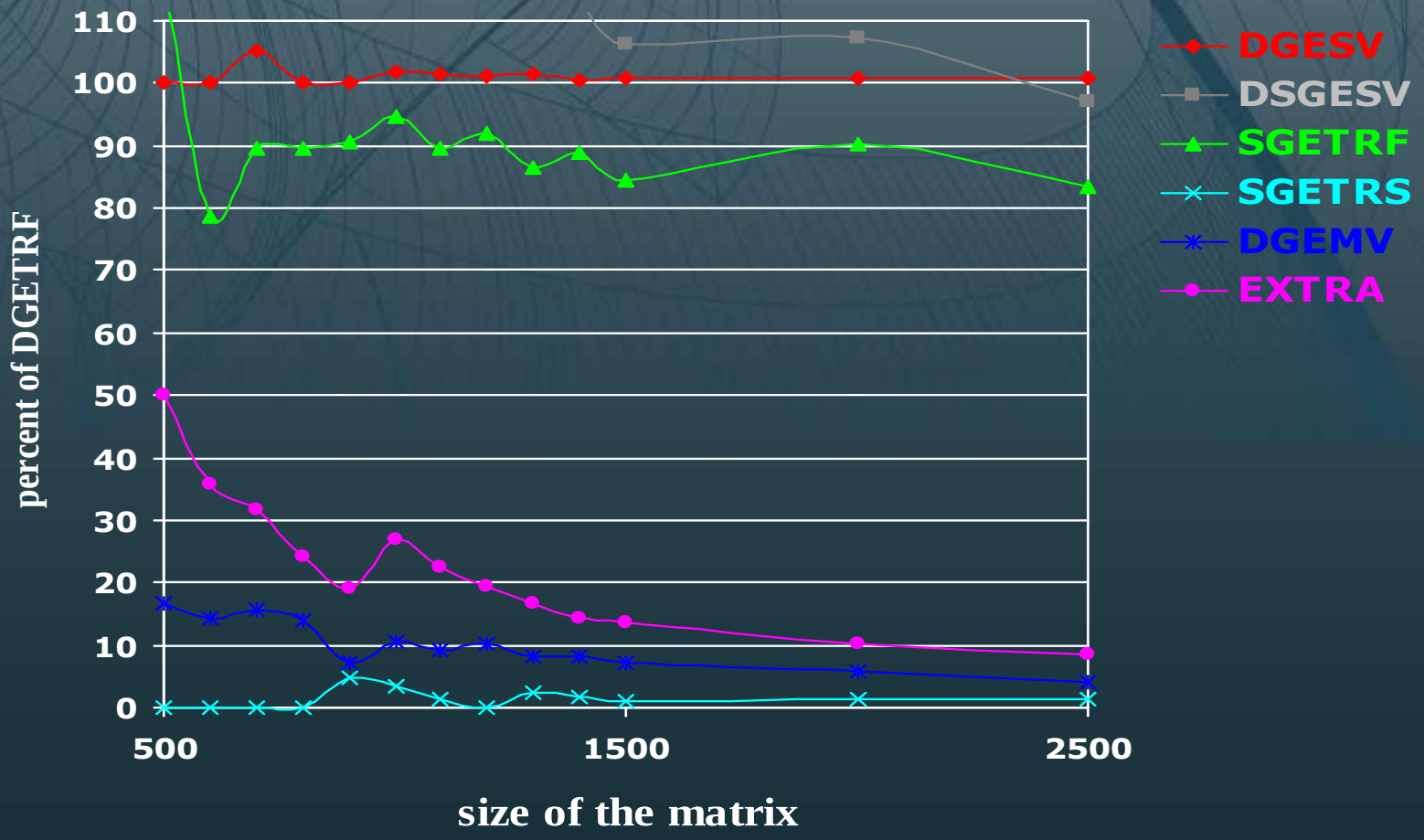
Compaq Alpha EV6, XML



SGI Octane (270 MHz), ATLAS 3.6 (1 thread)



IBM SP RS/6000, Power 3, (1.5 GHz), ESSL



Final Results

Architectures (BLAS/LAPACK)	n	DGEMM /SGEMM	DGETRF /SGETRF	DGESV /DSGESV	# iter
Intel Pentium III Coppermine (Goto)	3500	2.10	2.24	1.92	4
Intel Pentium III Katmai (Goto)	3000	2.12	2.11	1.79	4
Sun UltraSPARC IIe (Sunperf)	3000	1.45	1.79	1.58	4
Intel Pentium IV Prescott (Goto)	4000	2.00	1.86	1.57	5
Intel Pentium IV-M Northwood (Goto)	4000	2.02	1.98	1.84	5
AMD Opteron (Goto)	4000	1.98	1.93	1.53	5
Cray X1 (libsci)	4000	1.68	1.54	1.38	7
IBM Power PC G5 (2.7 GHz) (VecLib)	5000	2.29	2.05	1.24	5
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
Intel Itanium 2 (Goto and ATLAS)	1500	0.71			

very effective on a number, but not all, architectures.

Run on parallel machines

Architecture (BLAS-MPI)	# processors	*n*	PDGETRF / PSGETRF	PDGESV / PDSGESV	Number of iterations
AMD Opteron (Goto – OpenMPI MX)	32	22627	1.85	**1.79**	6
AMD Opteron (Goto – OpenMPI MX)	64	32000	1.90	**1.83**	6

- the cost of the iterative refinement $O(n^2)$ becomes negligible with respect to PDGETRF $O(n^3)$.
- Using PDSGESV is almost twice as fast (1.83) as opposed to using PDGESV for the same accuracy

Quad / double

n	QGESV time (s)	QDGESV 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	17.81	0.44	40.4
500	34.71	0.69	49.7
600	60.11	1.01	59.0
700	94.95	1.38	68.7
800	141.75	1.83	77.3
900	201.81	2.33	86.3
1000	276.94	2.92	94.8

- Intel Xeon 3.2Ghz
- ifort –O3
- Reference Blas

- Expected Accuracy: 10^{-32}
- No more than 3 steps of iterative refinement are needed.
- The speedup goes from 10 (n=100) to close to 100 (n=1000).

Implementation and Testing

DSGESV computes the solution to a real system of linear equations

A * X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

DSGESV first tries to factorize the matrix in SINGLE PRECISION and use this factorization within an iterative refinement procedure to produce a solution with DOUBLE PRECISION normwise backward error quality.

If the approach fails the method switch to a DOUBLE PRECISION factorization and solve.

The iterative refinement process is stopped if

ITER < ITERMAX = 30

or

Backward error = RNRM / (XNRM * ANRM) < MIN(4,SQRT(N/6) * EPS

where

- » ITER is the number of iteration in the iterative refinement process
- » RNRM is the 2-norm of the residual
- » XNRM is the 2-norm of the solution
- » ANRM is the Frobenius-norm of the matrix A
- » EPS is the relative machine precision returned by DLAMCH

Related work / Originality

- Iterative refinement is not a new subject. Lots of literature.
- Never been done for speed before, only for accuracy
- Iterative refinement was used to
  - Cope with not stable LU factorization (SuperLU, pivot growth)
  - Improve forward error (cf. Berkeley guys)

- Most of the theorems on mixed-precision iter-ref are:

  "what is the SINGLE accuracy I can get with iterative refinement single/double?"
- Our problem is :

  "what is the DOUBLE accuracy I can get using iterative refinement single/double?"

- See theoretical analysis at the end of technical report:
  http://www.cs.utk.edu/~library/TechReports/2006/ut-cs-06-574.pdf

extensions

- More Algorithms:
  - Cholesky / QR / eigenvalue / singular value
- Various precisions
  - Quad/double
- Change the outer iterative methods
  - Richardson -> GCR, GMRES
- Change the inner solve (SPARSE)
  - Instead of backward/forward solve in LU single, any solver (iterative methods) will do

More information

http://icl.cs.utk.edu/~julie/iter-ref/

Future work

» Implementation of Ax = b on the Cell processor