New Developments in LAPACK and ScaLAPACK

Sven Hammarling
NAG Ltd, Oxford
&
University of Manchester

With thanks to Jim Demmel, Jack Dongarra and the LAPACK team & contributors

Plan of Talk

- Introduction and Motivation
- LAPACK 3.1
- Future LAPACK and ScaLAPACK
  - New functionality
  - Improved efficiency
  - Improved accuracy
  - Automatic tuning
  - Improved ease of use
- Community involvement

LAWN – LAPACK Working Note
www.netlib.org/lapack/lawns/downloads

Motivation

- LAPACK and ScaLAPACK are widely used
  - Adopted by AMD, Cray, Fujitsu, HP, IBM, IMSL, Intel, MathWorks, NAG, NEC, Octave, SGI, Sun, …
  - > 70M web hits on netlib (incl. CLAPACK, LAPACK95)
- Many ways to improve the packages, based on
  - Algorithmic research of the project
  - Enthusiastic participation of research community
  - User/vendor survey
  - Opportunities and demands of new architectures, programming languages
- Further releases planned (NSF support); LAWN 164

Information on the Future

- Further releases planned (NSF support)
- LAWN 164: NSF Proposal
- LAWN 181: Prospectus for the Next LAPACK and ScaLAPACK Libraries (February ’07)

Participants

- UC Berkeley:
  - Jim Demmel, Ming Gu, W. Kahan, Beresford Parlett, Xiaoye Li, Osni Marques, Christof Voemel, David Bindel, Yozo Hida, Jason Riedy, Jianlin Xia, Jiang Zhu, …
- U Tennessee, Knoxville
  - Jack Dongarra, Julien Langou, Julie Langou, Piotr Luszczek, Stan Tomov, Alfredo Buttari, Jakub Kurzak, …
- Other Academic Institutions
  - UT Austin, UC Davis, Florida IT, U Kansas, U Maryland, North Carolina SU, San Jose SU, UC Santa Barbara
  - TU Berlin, U Electrocomm. (Japan), FU Hagen, U Carlos III Madrid, U Manchester, U Umeå, U Wuppertal, U Zagreb, …
- Research Institutions
  - CERFACS, LBL, …
- Industrial Partners
  - Cray, HP, IBM, Intel, MathWorks, NAG, SGI, …

Challenges

- Challenges for all large scale computing
- Example … your laptop
  - 64 cores, 256 threads/multicore chip in a few years
- Exponentially growing gaps between processor speed and memory speed
  - Yearly speed up: processor 59%; main memory bandwidth 23%; main memory latency 5.5% (National Research Council, 2005)
- Heterogeneity
- Asynchrony
- Fault tolerance

What do users want?

- High performance, ease of use, …
- Survey results
  - Small but interesting sample
  - What matrix sizes do you care about?
    - 1000s: 34%
    - 10,000s: 26%
    - 100,000s or 1Ms: 26%
  - How many processors, on distributed memory?
    - >10: 34%, >100: 31%, >1000: 19%
  - Do you use more than double precision?
    - Sometimes or frequently: 16%
  - Would Automatic Memory Allocation help?
    - Very useful: 72%, Not useful: 14%

Goals of Forthcoming Sca/LAPACK

1. Expand contents
   - More functions, more parallel implementations
2. Better algorithms
   - Faster, more accurate
3. Improve ease of use
4. Automate performance tuning
5. Better software engineering
6. Increased community involvement

LAPACK 3.1

LAPACK Release 3.1

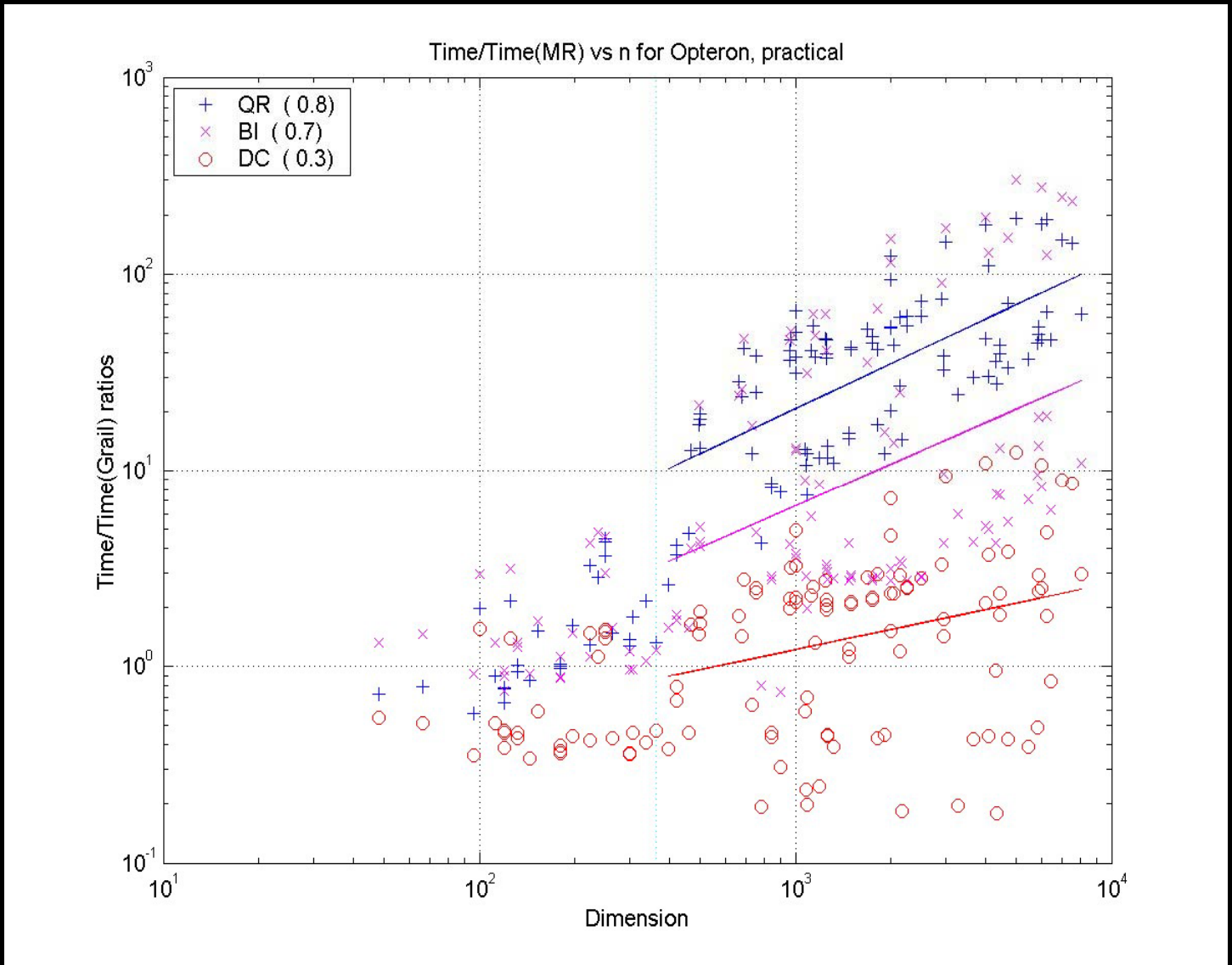
- 3.1.0 released on 12 November 2006
  - Improved numerical algorithms
  - Some further functionality
  - Thread safety
  - Bug fixes
- See: www.netlib.org/lapack/lapack-3.1.0.changes
- 3.1.1 released on 26 February 2007
  - Mainly bug fixes (principally to test software)
- See: www.netlib.org/lapack/lapack-3.1.1.changes

Improved numerical algorithms

- More efficient nonsymmetric eigensolver
  - improved reduction to Hessenberg form
  - improved multishift with aggressive early deflation for the *QR* algorithm applied to the Hessenberg form, can be up to ten times faster
  - SIMAX 23:929-973, 2002 (SIAG LA prize)
- Improved MRRR algorithm for tridiagonal matrices; impacts symmetric eigenproblem solvers
  - faster and more accurate than LAPACK 3.0
  - support for subset computation, *O(kn)* operations
  - LAWNs 162, 167

Run Time Ratios of Eigensolvers

(2.2 GHz Opteron + ACML)



Further Functionality

- Mixed precision iterative refinement for $Ax=b$

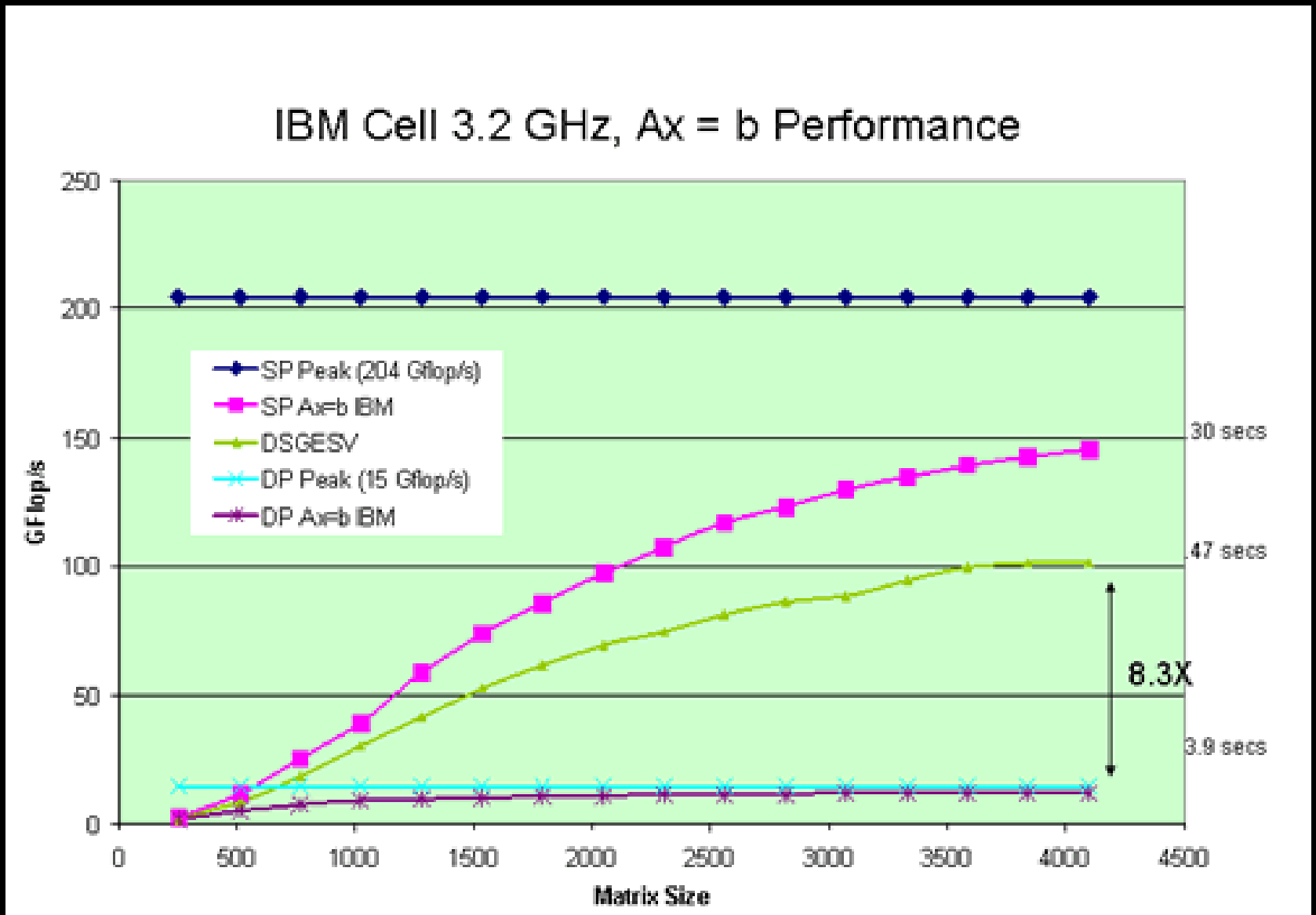
  Solve $A\tilde{x} = b$ via $A = PLU$ in single precision

  Compute $r = b - A\tilde{x}$ in double precision

  Solve $Ay = r$ for y in single precision,

  Update solution $x = \tilde{x} + y$ in double precision.

- Typically two times speed up on modern PCs
- Need $\kappa(A) \leq 1/\varepsilon_s$, use double precision otherwise
- LAWN 175 (LAWN 177 – cell processor, LAWN 180 – sparse)


IBM Cell 3.2 GHz, Ax = b Performance
250
200
150
100
50
0
GFlop/s
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Matrix Size
SP Peak (204 Gflop/s)
SP Ax=b IBM
DSGESV
DP Peak (15 Gflop/s)
DP Ax=b IBM
30 secs
47 secs
6.3X
3.9 secs

Thread Safety

- All driver and computational routines now thread safe
- xLAMCH supplied with LAPACK not absolutely guaranteed to be thread safe. Can be manually replaced by thread safe version
- Some obsolete auxiliaries not thread safe

Bug Fixes

- Subtle bug dating back to LINPACK fixed in *QR* factorization with pivoting.
  LAWN 176
- More support for NaNs and subnormal numbers
- Many other lesser bug fixes

Future LAPACK and ScaLAPACK

Future Developments: Functionality

- More LAPACK functionality into ScaLAPACK
- Updating matrix factorizations
- Polynomial eigenvalue problems, particularly quadratic
- Matrix functions such as $A^{1/2}$ and e^A
- CS decomposition and Product SVD
- Pivoted Cholesky
- More generalized Sylvester and Lyapunov equations
- Improved algorithms
- Enhanced accuracy algorithms
- …

Improved Ease of Use

- Core version likely to remain Fortran 77, with some Fortran 95 features (dynamic memory allocation, recursion, modules, environmental enquiries, …)
- Wrappers for the drivers in C and Fortran 95
- Wrappers for higher level languages such as MATLAB and Python

- Maximum efficiency in Fortran, best ease of use in high level languages or packages
- Straightforward to interface to Fortran

Automate Performance Tuning

- Widely used in performance tuning of Kernels, e.g. ATLAS, FFTW
- 1300 calls to `ILAENV` to get block sizes, etc.
  - Never been systematically tuned
- Extend automatic tuning techniques of ATLAS, etc. to these other parameters
  - Automation important as architectures evolve
- Allow users to supply simpler data layouts to ScaLAPACK and, if necessary, convert on the fly

Examples of New Functionality

Pivoted Cholesky Factorization

If A is positive semidefinite, of rank r, then there exists a permutation matrix P such that

$$P^T AP = U^T U$$

and U has the form

$$U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & 0 \end{pmatrix},$$

where U_{11} is upper triangular with positive diagonal elements

Pivoted Cholesky and LAPACK

- In order to incorporate pivoting, we need a Cholesky algorithm that updates the trailing matrix
- The Level 2 BLAS routine in LAPACK (**xPOTF2**) does update
- The Level 3 BLAS routine in LAPACK (**xPOTRF**) uses a left looking algorithm, and so does not update
- Need a right looking block-partitioned algorithm comparable to **xPOTF2**

$$M_k A_{k-1} M_k^T = A_k = \begin{pmatrix} I & 0 \\ 0 & \tilde{A}_k \end{pmatrix}$$

Level 2 BLAS Pivoting Routine

We can readily introduce pivoting into a right looking Level 2 BLAS Cholesky routine.

At the $(k+1)$th stage:

- Find pivot index q as

$$q = \min \left\{ j : a_{jj}^{(k)} = \max_{k \le i \le n} a_{ii}^{(k)} \right\}$$

- Swap rows and columns k and q
- Stop if $\max_{k \le i \le n} a_{ii}^{(k)} \le n \cdot eps \cdot a_{11}^{(1)}$ (Higham)

  (LINPACK uses $\max_{k \le i \le n} a_{ii}^{(k)} \le 0$)

Block-Partitioned Right Looking Cholesky

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ . & A_{22} & A_{23} \\ . & . & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ U_{12}^T & U_{22}^T & 0 \\ U_{13}^T & U_{23}^T & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \tilde{A} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & I \end{pmatrix}$$

$$U_{22}^T U_{22} = A_{22} - U_{12}^T U_{12}$$

$$U_{22}^T U_{23} = A_{23} - U_{12}^T U_{13}$$

$$\tilde{A} = A_{33} - U_{13}^T U_{13} - U_{23}^T U_{23}$$

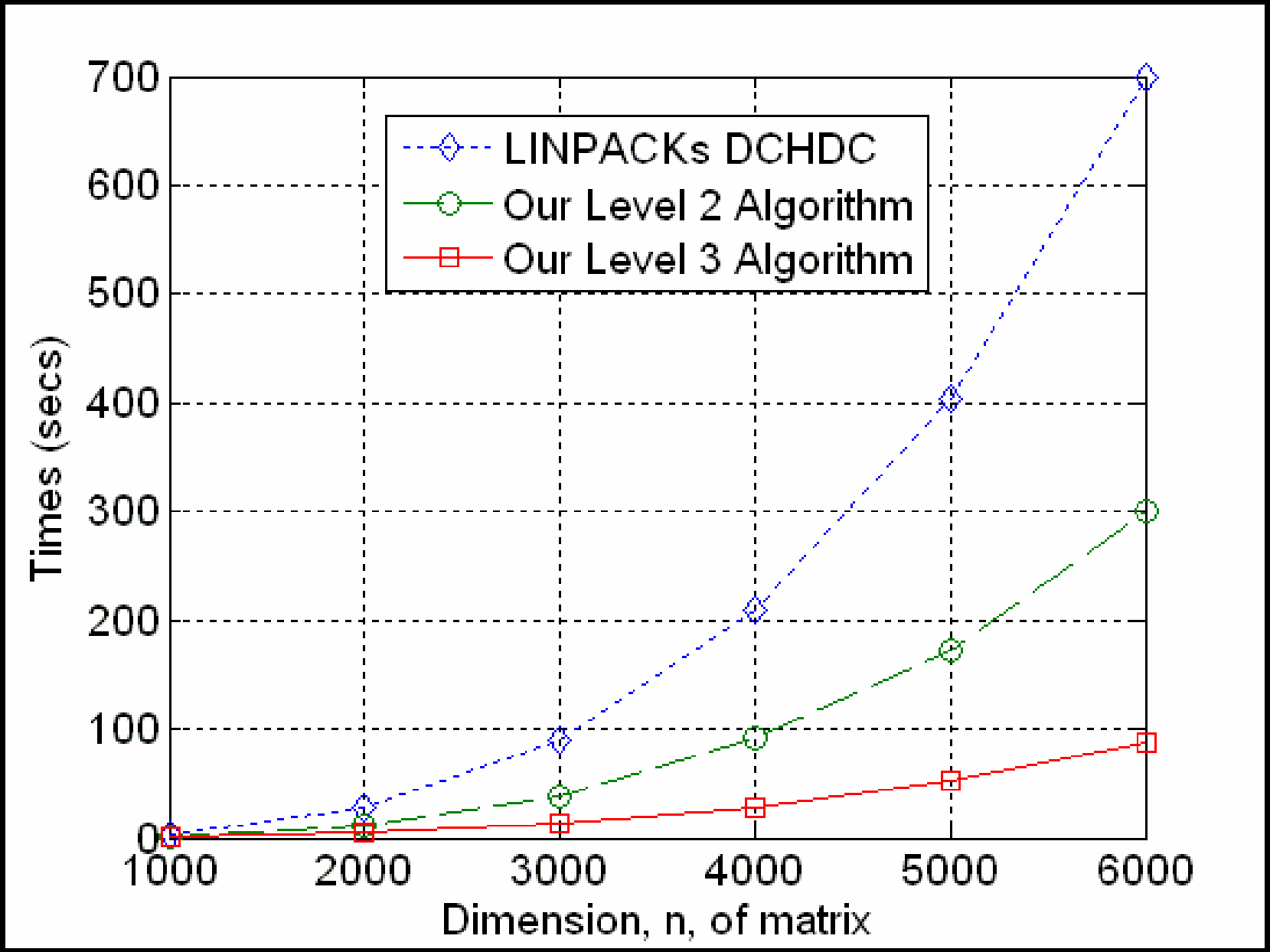
$$A_{k-1} = \begin{pmatrix} \square & \square \\ \square & \tilde{A}_{k-1} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 \\ U_{12}^T & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & A_k \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & I \end{pmatrix}$$

$$A_k = \tilde{A}_{k-1} - U_{12}^T U_{12}$$

Block-Partitioned Pivoted Cholesky

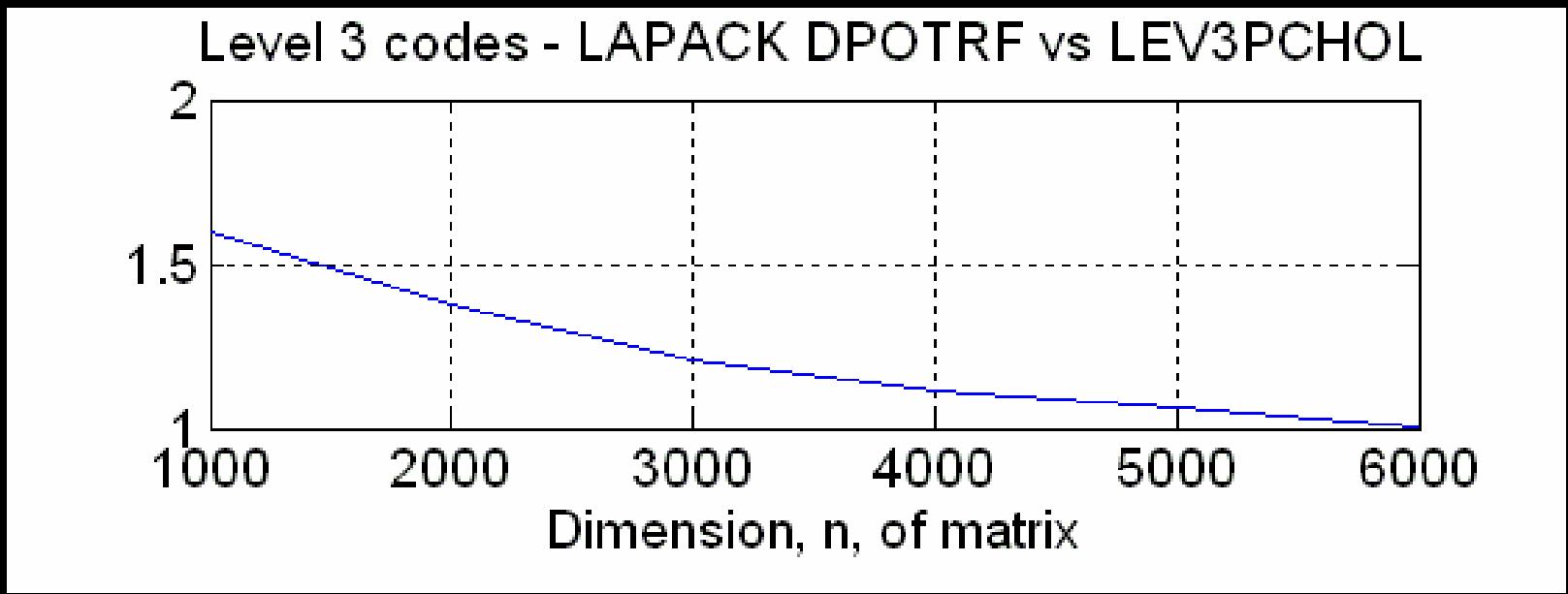
- This gives the basis of a Level 3 BLAS algorithm in which pivoting can be incorporated
  - Form the trailing matrix $\tilde{A}$
  - Use the Level 2 BLAS algorithm with pivoting to factorize the b by b matrix U_{22}, where b is the block size (but choose pivot from whole of trailing matrix)
- Routine is significantly faster than the equivalent LINPACK routine and, with Higham's stopping criterion, gives more reliable rank detection
- Craig Lucas, LAWN 161

Results - Speed



Results – Level 3 Ratios

Ratio of time for pivoted code
over LAPACK unpivoted code



Rank Revealing Properties

- Comparing computed rank vs the “expected” rank of our test matrices, the new code gets it right in 299/300 cases
- Table below shows over estimation of rank in DCHDC ($|\hat{r} - r|$)

n	70	100	200	500	1000
min	0	1	1	3	5
max	8	11	13	19	19

Rook Pivoting

- Incorporate rook pivoting into the symmetric indefinite factorization, with diagonal pivoting

$$A = P^T U^T D U P$$

  - allows U to be bounded
  - D has well-conditioned 2 by 2 diagonal blocks
- (need to investigate fast Bunch – Parlett)
- Ashcraft, Grimes and Lewis, SIMAX 20:513-561, 1998

Block-Partitioned QR Factorization

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}, \quad Q = H_1 H_2 \ldots H_n,$$

where H_r is an elementary Householder transformation matrix of the form

$$H_r = I - \tau_r v_r v_r^T.$$

Block partitioned algorithms are based on the representation

$$H_1 H_2 \ldots H_b = I - VTV^T,$$

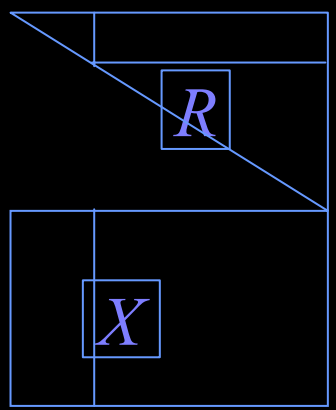
where $V = (v_1\ v_2 \ldots v_b)$, T is b by b upper triangular and b is the block size. Multiplication by $(I - VTV^T)$ involves Level 3 BLAS operations.

Block-Partitioned QR Updating

Commonest case is that of adding new rows, for example adding new observations in a least squares problem.

$$\tilde{A} = \begin{pmatrix} A \\ X \end{pmatrix} = \begin{pmatrix} Q & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} R \\ 0 \\ X \end{pmatrix} = \tilde{Q} \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}.$$

To achieve this we just need a QR factorization of $\begin{pmatrix} R \\ X \end{pmatrix}$. We can apply block partitioned Householder in the usual way



Block-Partitioned Updating

- Updating by a contiguous block of columns is also straightforward
- Downdating, as in the standard case, needs care.
- Craig Lucas, PhD thesis, 2004

- SH, Nicholas J. Higham and Craig Lucas (2006) *LAPACK-Style Codes for Pivoted Cholesky and QR Updating.* MIMS preprint 2006.385 eprints.ma.man.ac.uk

Condition number of a Tridiagonal Matrix

- Computed via the *QR* factorization in $O(n)$ operations (not estimated)
- Utilizes a result from 1974 of Gill, Golub, Murray and Saunders, on the structure of a ‘special’ matrix
- Algorithm 2.1 in “G Hargreaves. *Computing the Condition Number of Tridiagonal and Diagonal-Plus-Semiseparable Matrices in Linear Time*, SIMAX, 27, 801–820, 2006”, for which stability is proved

Lyapunov and Sylvester Equations

- Further Lyapunov and Sylvester solvers based upon
  - RECSY: High Performance Library for Sylvester-Type Matrix Equations, Isak Jonsson and Bo Kågström at the University of Umeå
  - Uses recursive blocked algorithms
  - ScaLAPACK versions also underway
  - www.cs.umu.se/~isak/recsy/
- Speeds up the generalized nonsymmetric eigenproblem

Improved Efficiency Examples

Faster Algorithms 1

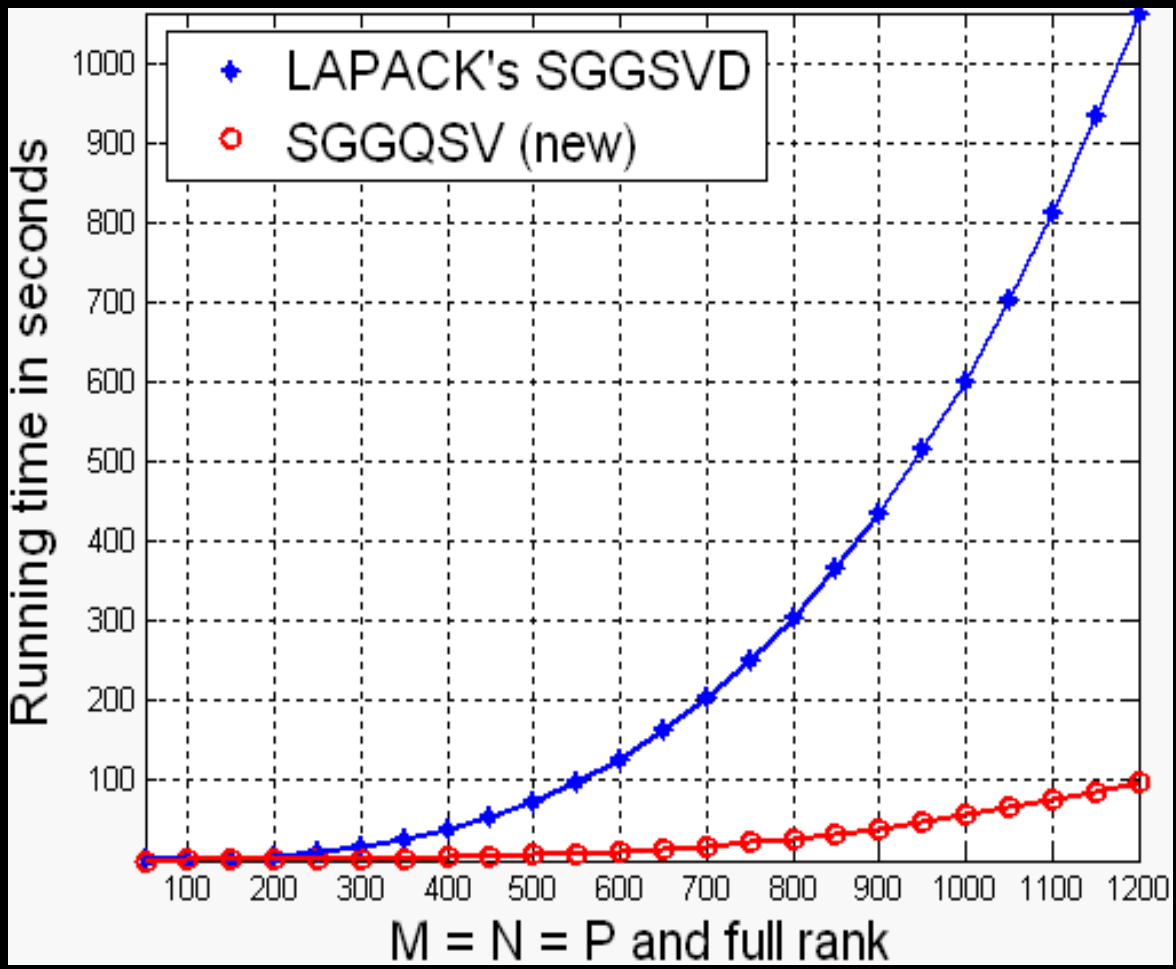
- Faster symmetric eigensolvers and SVD routines
  - MRRR algorithm for bidiagonal SVD
  - faster tridiagonal; initially reduce to band form
  - Improved bidiagonal reduction; LAWN 174
- Faster generalized eigensolvers; LAWNs 171, 173
  - On 26 real test matrices, speedups of 14.7, 4.4 on average
- Further mixed precision iterative refinement routines
  - More linear equation solvers
  - Eigenvalue problems

Faster Algorithms 1 (cont'd)

- Faster GSVD routine
  - Based on CS decomposition (C Van Loan. *Generalizing the Singular Value Decomposition*, SINUM, 13, 76–83, 1976)
  - MSc Thesis, Jenny Jian Wang, UC Davis

New GSVD Algorithm

Given m by n A and p by n B, factor $A = U \sum_a X$ and $B = V \sum_b X$



Bai and Wang, UC Davis

PSVD, CSD under development

Faster Algorithms 2

- Novel Data Layouts
  - E Elmroth, F Gustavson, I Jonsson and B Kågstrom. *Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software*, SIAM Review, 2004
  - require different data layouts, but will provide conventional interfaces
  - Some aim to be cache oblivious

Level 3 BLAS Packed Routines

- Utilize rectangular full packed (RFP) format
- Can convert in $O(n^2)$ data movement
- LAPACK packed format cannot utilise Level 3 BLAS

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} \Rightarrow AS = \begin{pmatrix} U_{11} & A_{12} \\ . & U_{22} \end{pmatrix}$$

$$AF = \begin{pmatrix} U_{22} \\ \hline U_{11}^T \\ A_{12} \end{pmatrix}$$

$$A = \left(\begin{array}{ccccc|cccc}
a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & a_{19} \\
. & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} & a_{29} \\
. & . & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} & a_{39} \\
. & . & . & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} & a_{49} \\
\hline
. & . & . & . & a_{55} & a_{56} & a_{57} & a_{58} & a_{59} \\
. & . & . & . & . & a_{66} & a_{67} & a_{68} & a_{69} \\
. & . & . & . & . & . & a_{77} & a_{78} & a_{79} \\
. & . & . & . & . & . & . & a_{88} & a_{89} \\
. & . & . & . & . & . & . & . & a_{99}
\end{array}\right)$$

$$AP = \left(a_{11} \mid a_{12}\ a_{22} \mid a_{13}\ a_{23}\ a_{33} \mid a_{14}\ a_{24}\ a_{34}\ \ldots\right)$$

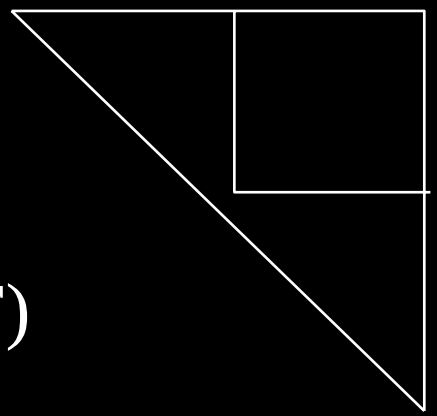
$$AF = \begin{pmatrix} a_{55} & a_{56} & a_{57} & a_{58} & a_{59} \\ a_{11} & a_{66} & a_{67} & a_{68} & a_{69} \\ a_{12} & a_{22} & a_{77} & a_{78} & a_{79} \\ a_{13} & a_{23} & a_{33} & a_{88} & a_{89} \\ a_{14} & a_{24} & a_{34} & a_{44} & a_{99} \\ a_{15} & a_{16} & a_{17} & a_{18} & a_{19} \\ a_{25} & a_{26} & a_{27} & a_{28} & a_{29} \\ a_{35} & a_{36} & a_{37} & a_{38} & a_{39} \\ a_{45} & a_{46} & a_{47} & a_{48} & a_{49} \end{pmatrix}$$

$A = U^T U$ - Block Cholesky

$$\begin{pmatrix} A_{11} & A_{12} \\ . & A_{22} \end{pmatrix} = \begin{pmatrix} R_{11}^T & 0 \\ R_{12}^T & R_{22}^T \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

$$A_{11} = R_{11}^T R_{11}, \quad A_{12} = R_{11}^T R_{12}$$

$$A_{22} = R_{12}^T R_{12} + R_{22}^T R_{22}$$



$$R_{11}^T R_{11} = A_{11} \quad \text{(DPOTRF)}$$

$$R_{11}^T R_{12} = A_{12} \quad \text{(DTRSM)}$$

$$R_{22}^T R_{22} = A_{22} - R_{12}^T R_{12} \quad \text{(DSYRK + DPOTRF)}$$

Improved Accuracy Examples

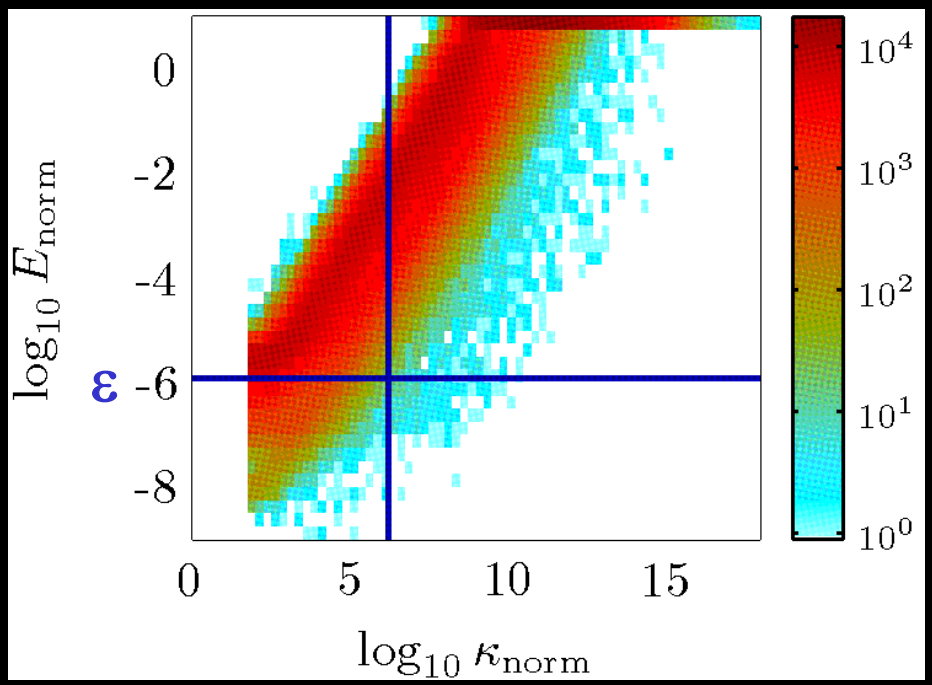
More Accurate Algorithms

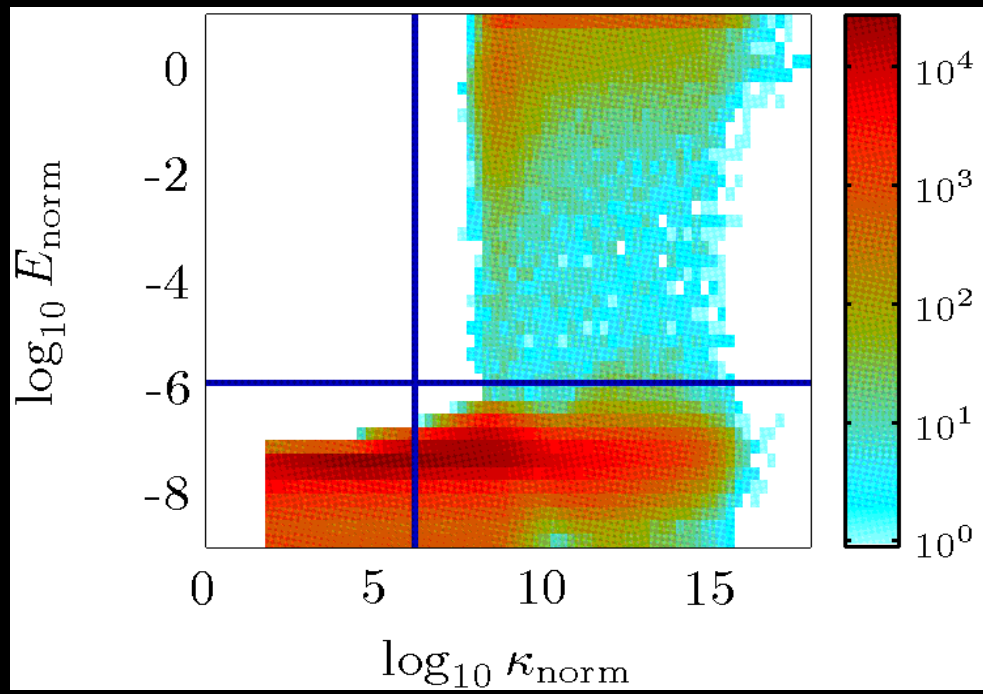
- Iterative refinement for $Ax=b$
  - "Promise" the right answer for $O(n^2)$ additional cost
  - Iterative refinement with extra-precise residuals
  - Extra-precise BLAS needed
  - "Guarantees" based on condition number estimates
    - Condition estimate < $1/(n^{1/2}\varepsilon)$ $\Rightarrow$ reliable answer and tiny error bounds
    - No bad bounds in 6.2M tests
- LAWN 165 (TOMS 32:325–351, 2006)

More Accurate: Solve $Ax=b$

Conventional Gaussian Elimination

With extra precise iterative refinement





$\varepsilon = n^{1/2}\, 2^{-24}$

More Accurate Algorithms

- Iterative refinement for least squares problems using the extra-precise BLAS
- Arbitrary precision versions of everything
  - using your favourite multiple precision package
- Jacobi-based SVD
  - can be arbitrarily more accurate on tiny singular values
  - yet faster than QR iteration, and within two times of divide and conquer
  - preprocess by QR with pivoting, block Jacobi, …
  - LAWNs 169, 170

Improved ScaLAPACK Coverage

Missing Drivers in Sca/LAPACK

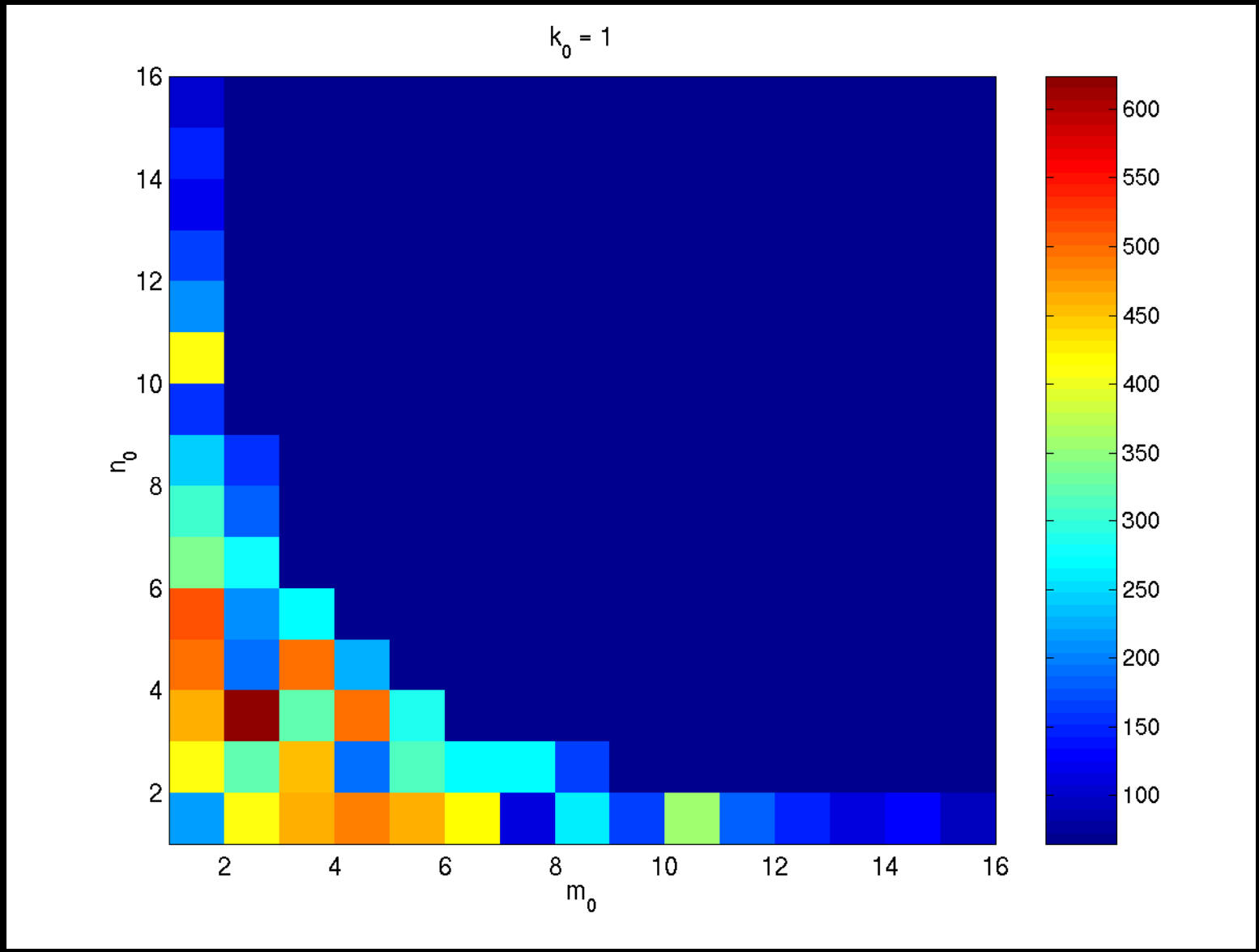
		LAPACK	ScaLAPACK
Linear Equations	*LU*	xGESV	PxGESV
	Cholesky	xPOSV	PxPOSV
	LDL^T	xSYSV	missing
Least Squares (LS)	*QR*	xGELS	PxGELS
	QR+pivot	xGELSY	missing
	SVD/*QR*	xGELSS	missing
	SVD/D&C	xGELSD	missing (ok?)
	SVD/MR^3	missing	missing
Generalized LS	LS + equality constr.	xGGLSE	missing
	Generalized LM	xGGGL	missing
	Above + Iterative ref.	missing	missing

More Missing Drivers

		LAPACK	ScaLAPACK
Symmetric EVD	*QR* / Bisection+Invit D&C MR^3	xSYEV / X xSYEVD xSYEVR	PxSYEV / X PxSYEVD missing
Nonsymmetric EVD	Schur form Vectors also	xGEES / X xGEEV / X	missing (driver) missing
SVD	*QR* D&C MR^3 Jacobi	xGESVD xGESDD missing missing	PxGESVD missing (ok?) missing missing
Generalized Symmetric EVD	*QR* / Bisection+Invit D&C MR^3	xSYGV / X xSYGVD missing	PxSYGV / X missing (ok?) missing
Generalized Nonsymmetric EVD	Schur form Vectors also	xGGES / X xGGEV / X	missing missing
Generalized SVD	Kogbetliantz MR^3	xGGSVD missing	missing (ok) missing

Automatic Tuning

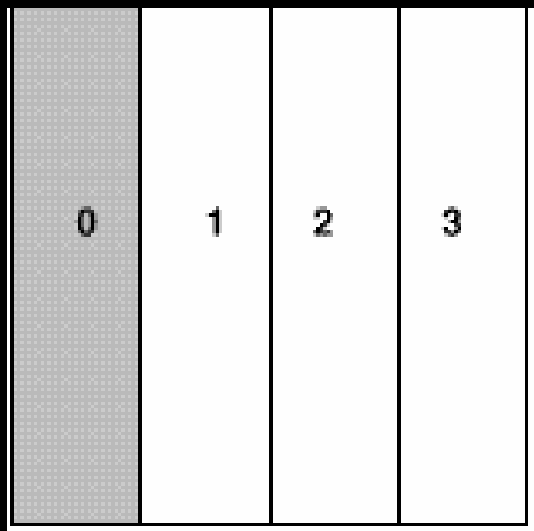
Optimizing Blocksizes for Matmul

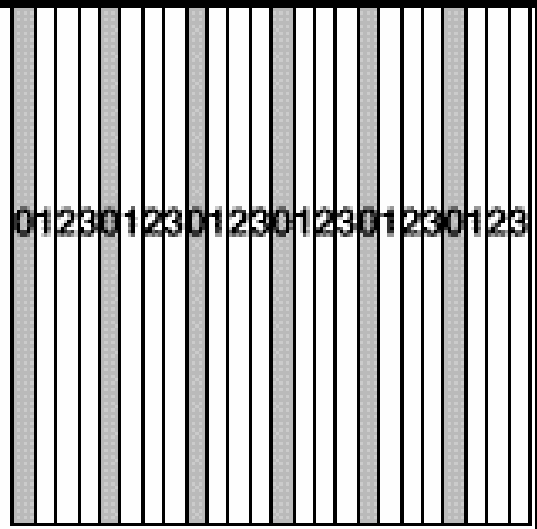


Finding a Needle in a Haystack – So Automate

ScaLAPACK Data Layouts

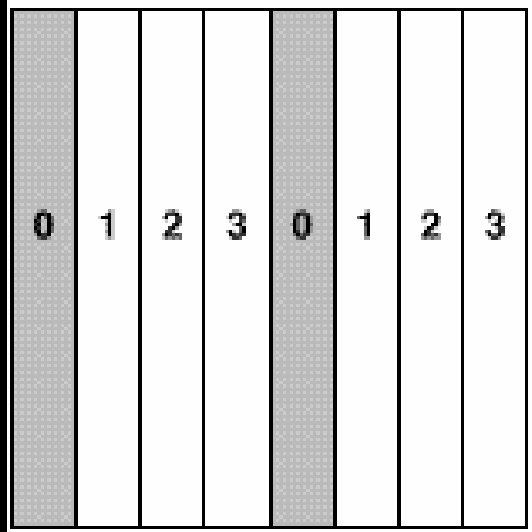
1D Block

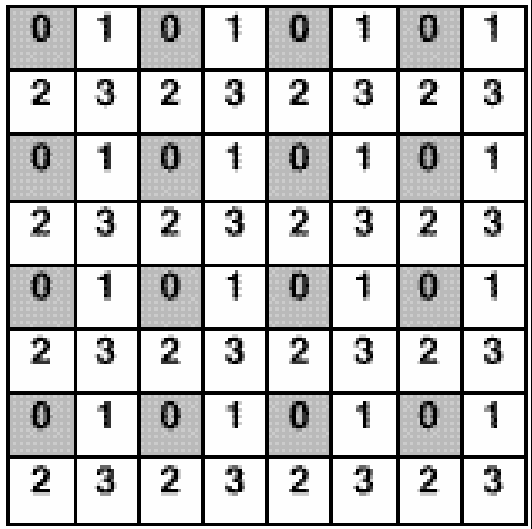




1D Cyclic

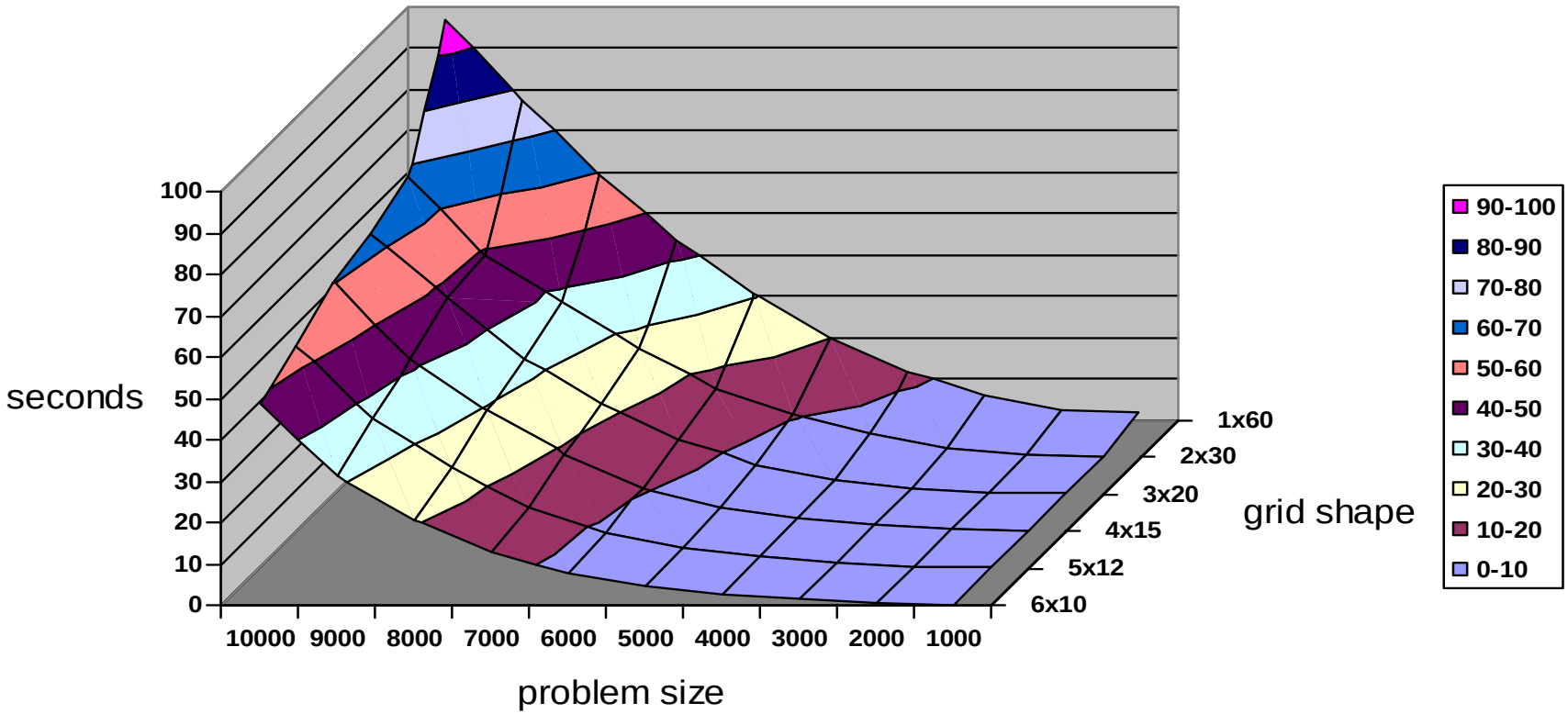
1D Block Cyclic





2D Block Cyclic

Execution time of PDGESV for various grid shapes



Speedups for using 2D processor grid range from 2x to 8x
Cost of redistributing from 1D to best 2D layout 1% - 10%

Times obtained on: 60 processors, Dual AMD Opteron 1.4GHz Cluster w/Myrinet Interconnect with 2GB Memory

Community Involvement

- To help identify priorities
  - more interesting tasks than are funded
  - see www.netlib.org/lapack-dev for list
- To help identify promising algorithms
  - what have we missed?
- To help do the work
  - bug reports, provide fixes
  - again, more tasks than we are funded to do
  - already happening: thank you!
- Contributors Guide:
  www.netlib.org/lapack-dev/lapack-coding/program-style.html

Conclusions

- Lots to do in dense linear algebra
  - new and improved numerical algorithms
  - continuing architectural challenges
    - Parallelism, performance tuning
  - ease of use, software engineering
- Some grant support, but success depends also on contributions from community
  - Contributors' guide drafted

www.netlib.org/lapack