A parallel tiled solver for dense symmetric indefinite systems on multicore architectures

Marc Baboulin

Inria Saclay - Île-de-France
Université Paris-Sud

Collaboration with Dulceneia Becker and Jack Dongarra (University of Tennessee)

Outline

Motivation

How to speed up numerical simulations ?

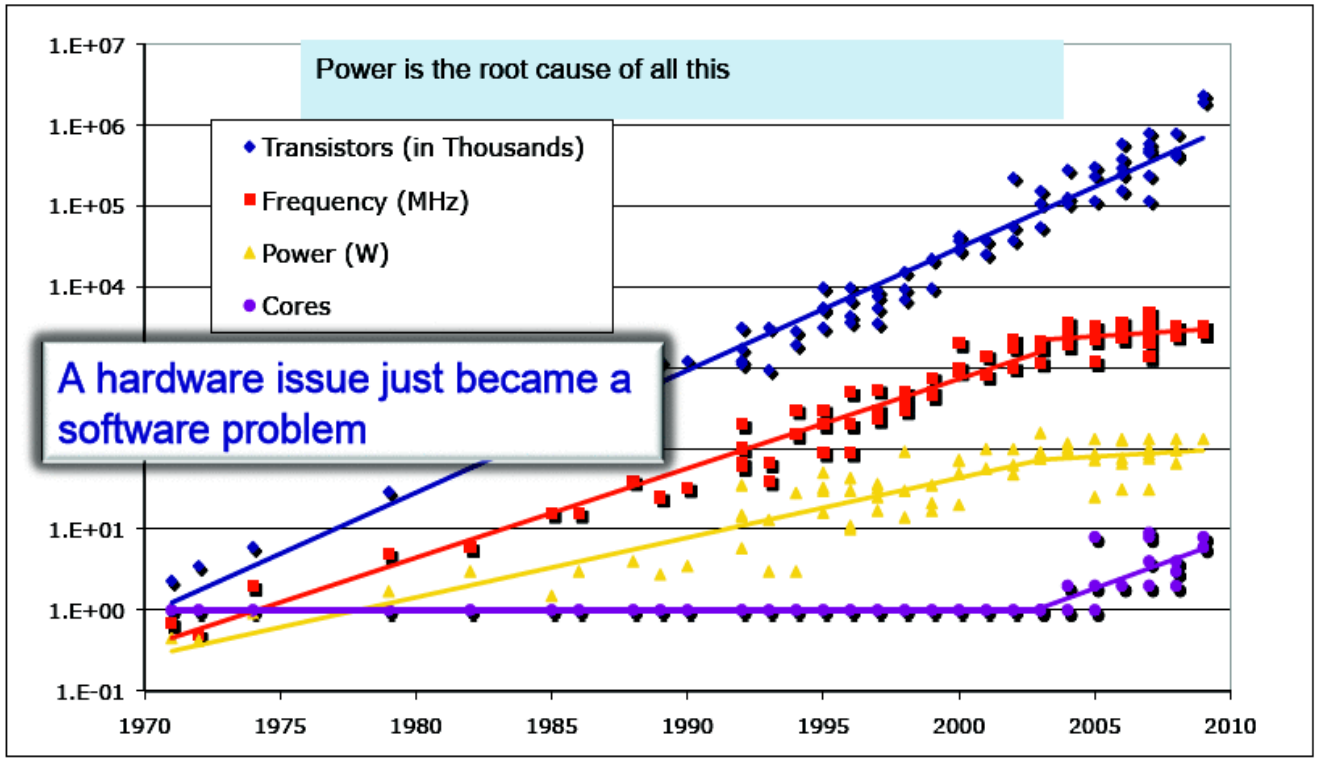
(while maintaining accuracy)

- Exploit advances in hardware (e.g multicore, GPUs, FPGAs, Cell)
- Change numerical methods (e.g. randomized algorithms)

Impact on linear algebra libraries ?

- LAPACK, ScaLAPACK, sparse solvers, iterative solvers...have to be (are being) rethought and rewritten
- Develop efficient linear algebra algorithms for multicore and multicore/GPU systems (MAGMA and PLASMA projects).

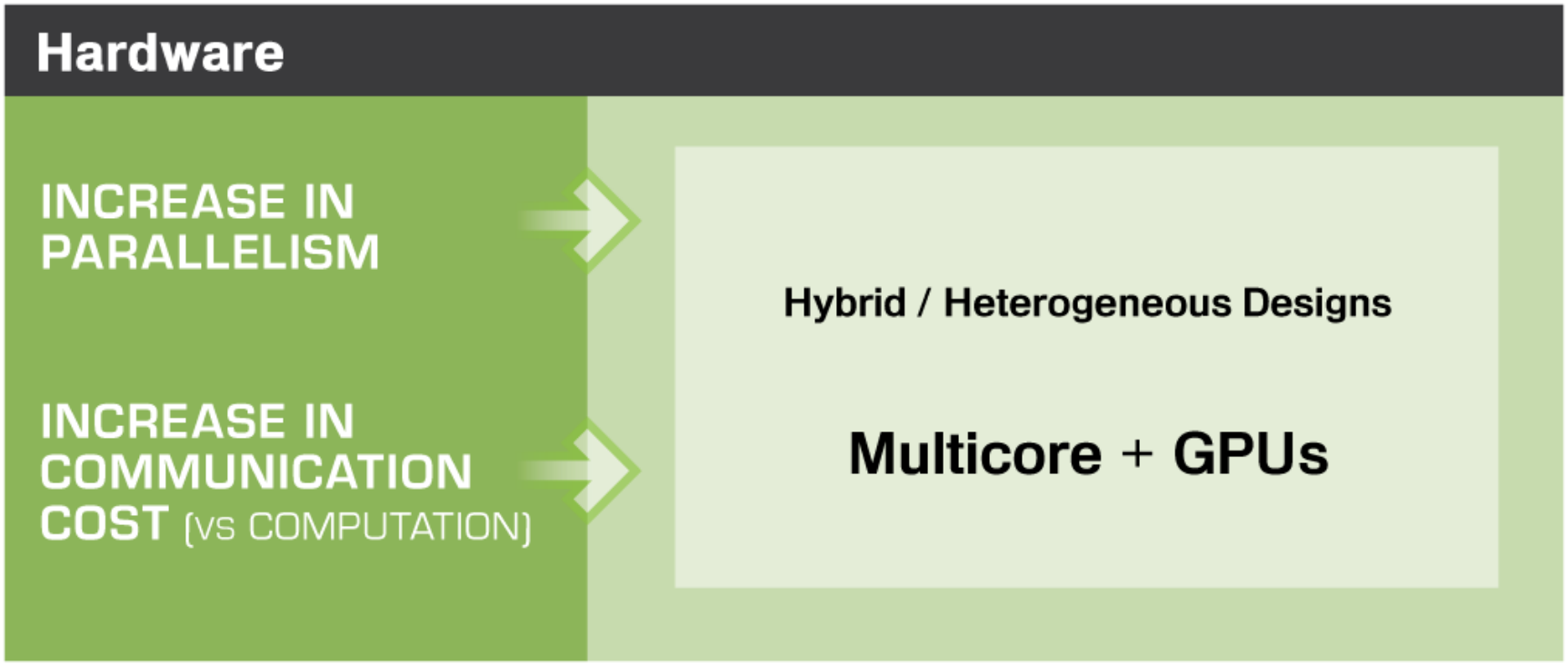
The “new deal” of high performance computing



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

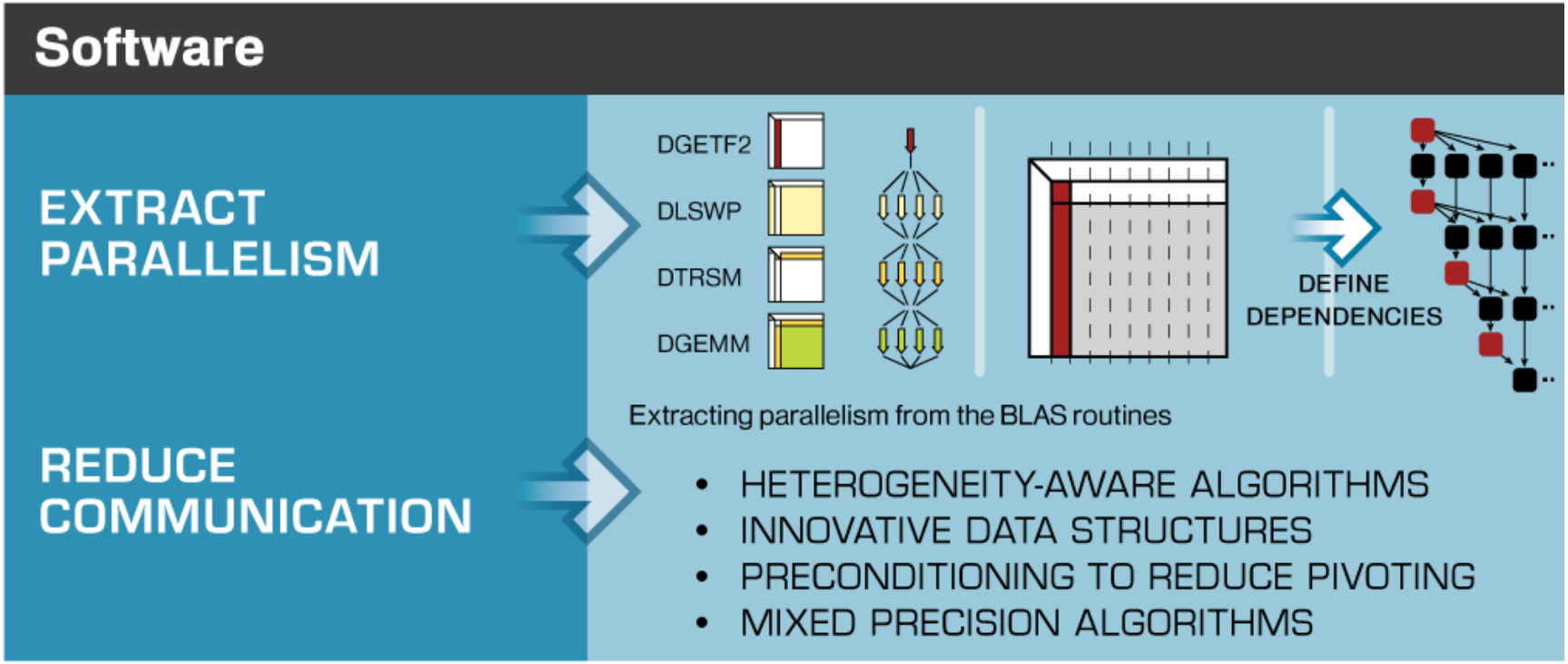
Hardware trends: power issues and the move towards multicore, hybrid GPU-accelerated systems

Hardware to software trends

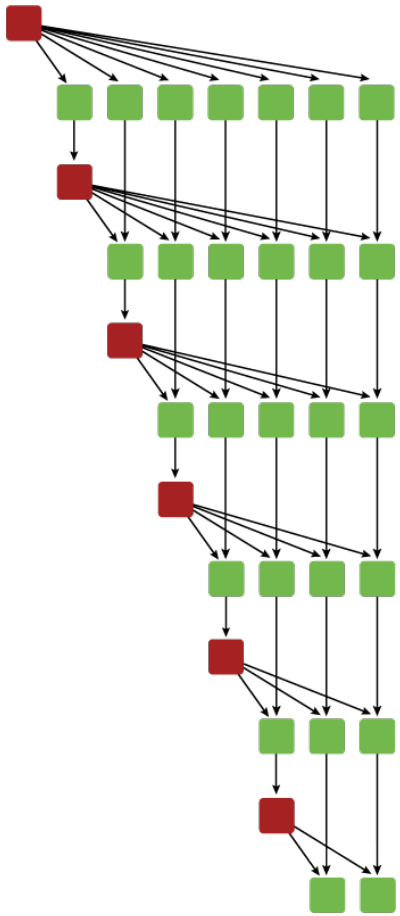


Processor speed: + 59% / year
Memory bandwidth: + 23% / year
Latency: - 5.5% / year

Hardware to software trends



Task splitting for Multicore



Algorithms as Directed Acyclic Graph (DAG)
(small tasks/tiles for multicore)

Outline

Outline

Objective

Propose a parallel solver for dense symmetric indefinite systems.
(There is currently no parallel implementation for such systems in dense public domain libraries)

- Issues on pivoting for symmetric indefinite matrices
- Symmetric randomization – recursive butterfly transformations
- Parallel tiled LDL^T factorization
- Numerical and performance results

Symmetric indefinite linear systems

- **Symmetric** (dense) linear system $Ax = b$
- A is **indefinite** when $x^T Ax$ can take on both positive and negative values
- **Applications**: least-squares via augmented system method, Maxwell equations in electromagnetics, optimization problems...
- Factorization

$$A = LDL^T$$

  where L is unit lower triangular and D is diagonal
- Solve $Ax = b$ by solving successively

$$Lz = b, \quad Dy = z, \quad L^T x = y$$

- **Not stable** – to ensure stability pivoting is usually required
- Requires $n^3/3$ flops (half the cost of LU)

Symmetric indefinite linear systems - pivoting

- Factorization

$$PAP^T = LDL^T$$

where

P is a permutation matrix
L is unit lower triangular
D is **block-diagonal**, with blocks of size 1×1 or 2×2

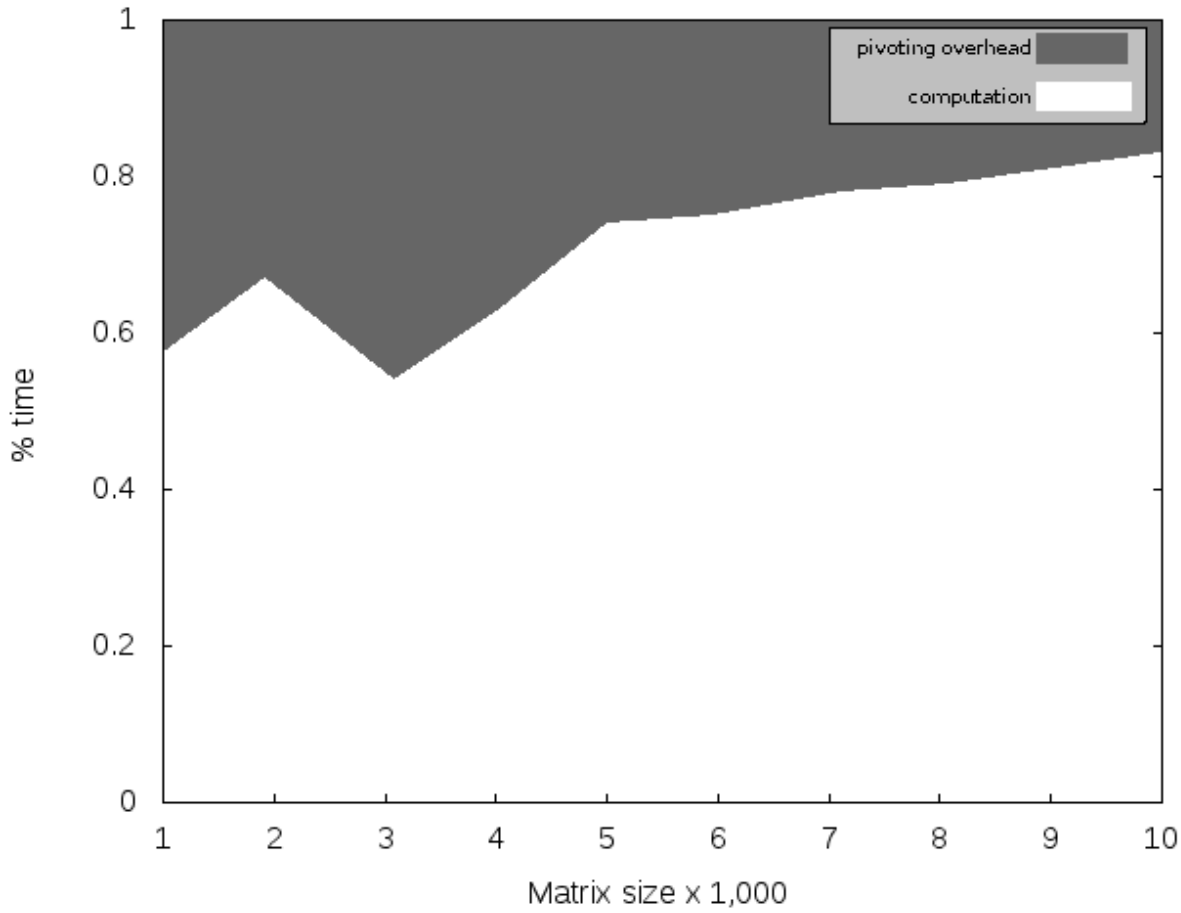
- Solve $Ax = b$ by solving the triangular or block-diagonal systems

$$Lz = Pb, \quad Dw = z, \quad L^T y = w, \quad x = P^T y$$

Pivoting

No floating-point operation in pivoting but it involves irregular data movements and between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ comparisons.

Pivoting is expensive



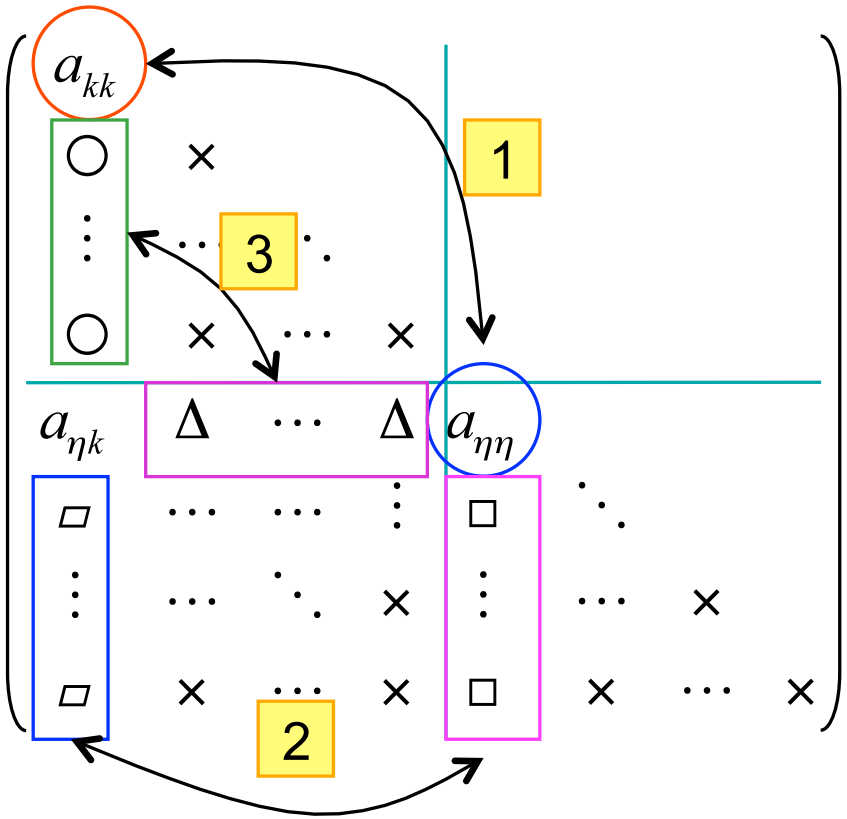
Cost of partial pivoting in LU factorization (MAGMA)

CPU 1 $\times$ Quad-Core Intel Core2 Q9300 @ 2.50 GHz - GPU C2050 @ 1.15 GHz

Symmetric pivoting

- To maintain symmetry, columns and rows must be interchanged

- Compromises data locality
- Increases data dependence



Reducing communication due to pivoting

- **Communication-avoiding techniques**

  L. Grigori, J. Demmel, and H. Xiang, **Communication avoiding Gaussian elimination** *Supercomupting 2008 proceedings.*

- **New data structures**

  V. Volkov, J. Demmel, **LU, QR, Cholesky factorizations using vector capabilities of GPUs**, *Lapack Working note 204*.

- **No pivoting by randomizing instead**

  initially proposed by [Parker, 1995] (for LU factorization)
  revisited in [MB, Dongarra, Herrmann, Tomov, TOMS, to appear]

  Idea: the original matrix is transformed into a matrix that would be sufficiently "random" so that, with a probability close to 1, pivoting is not needed.

How to avoid pivoting?

Symmetric Random Butterfly Transformation (SRBT)

- To solve $Ax = b$:
  1. Compute $A_r = U^T A U$ (U random matrix)
  2. Factorize A_r **without pivoting** (LDL^T)
  3. Solve

$$A_r y = U^T b \quad \longrightarrow \quad LDL^T y = U^T b$$

     and then the solution is $x = Uy$
- U is a **Recursive Butterfly Matrix**
- Properties:
  - **Randomization is cheap** ($\mathcal{O}(n^2)$ operations)
  - **LDL^T with no pivoting is fast** (strives for a “Cholesky speed”)
  - **Accuracy** is in practice similar to Bunch-Kaufman (with iterative refinement)

Butterfly Matrix

A **butterfly matrix** is defined as any n-by-n matrix of the form:

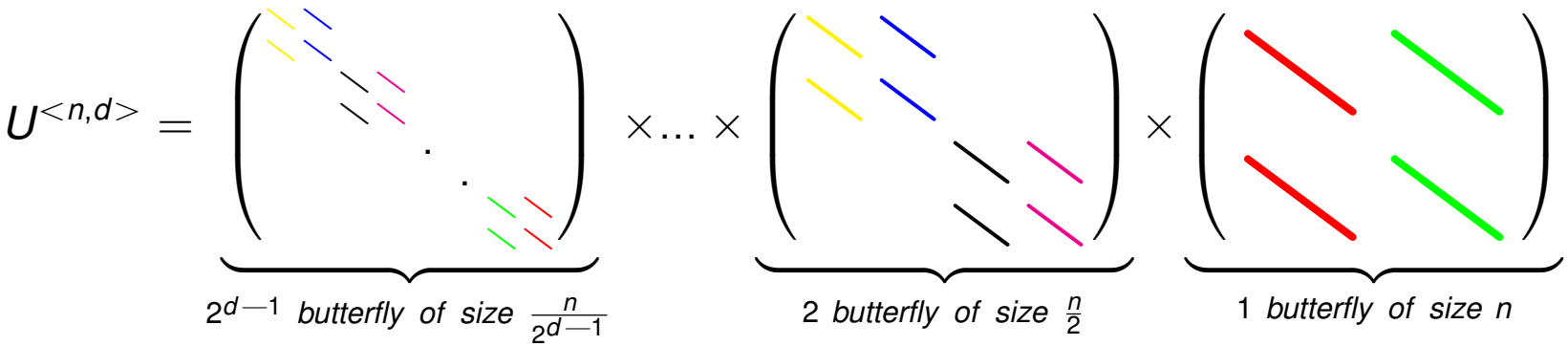
$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix}$$

where R and S are random diagonal matrices.

$$B = \left(\begin{array}{cc} \diagdown & \diagup \\ \diagdown & \diagup \end{array} \right)$$

Recursive Butterfly Matrix

A **recursive butterfly matrix** of size n and depth d is :



We consider a limited number of recursions.

Recursive butterfly matrix (e.g. for 3 recursions)

$$U^{<3>} = \begin{bmatrix} B_4 & & & \\ & B_5 & & \\ & & B_6 & \\ & & & B_7 \end{bmatrix} \begin{bmatrix} B_2 & \\ & B_3 \end{bmatrix} \begin{bmatrix} & \\ B_1 & \end{bmatrix}$$

U is $n \times n$

B_1 is $n \times n$

B_2 and B_3 are $\frac{n}{2} \times \frac{n}{2}$

B_4 to B_7 are $\frac{n}{4} \times \frac{n}{4}$

$$B_i = \frac{1}{\sqrt{2}} \begin{pmatrix} \diagdown & \diagup \\ \diagdown & \diagup \end{pmatrix}$$

-Packed storage

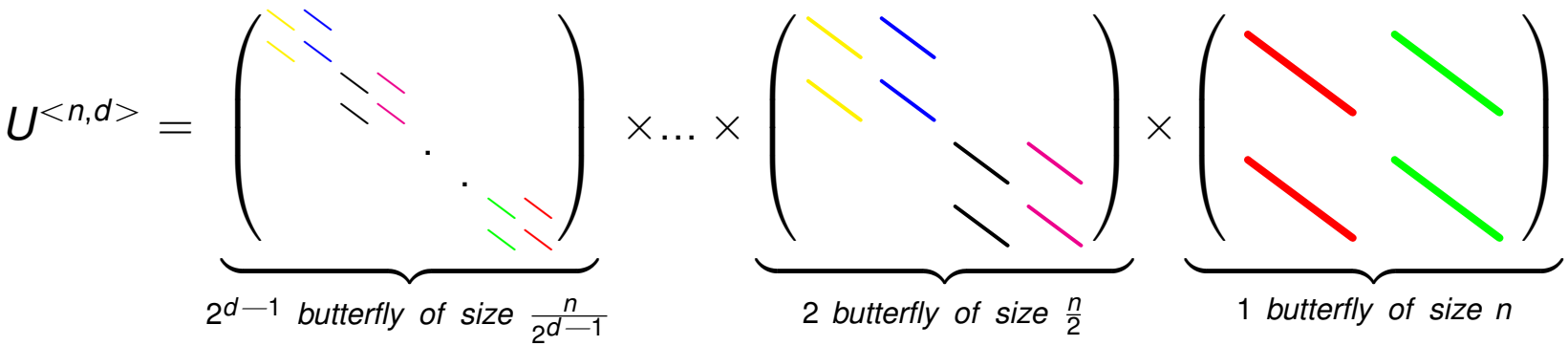
-Efficient kernels for applying butterflies ($U^T AU$)

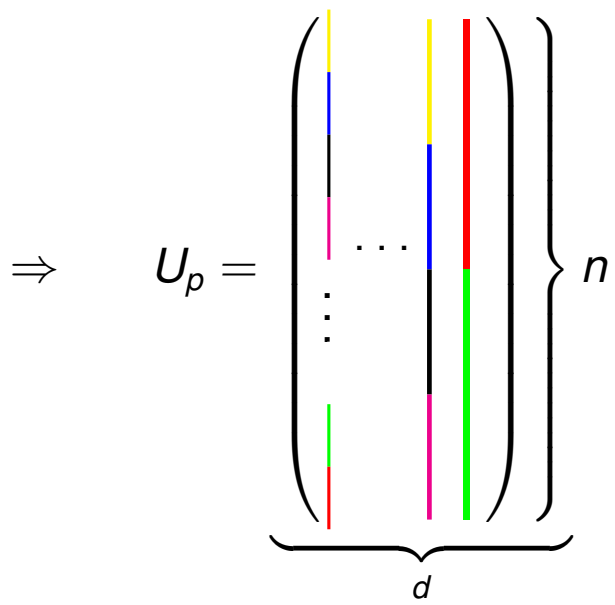
-keep the condition number almost unchanged

Packed storage for a recursive butterfly matrix

$$U^{<n,d>} = \underbrace{\begin{pmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{pmatrix}}_{2^{d-1} \text{ butterfly of size } \frac{n}{2^{d-1}}} \times \ldots \times \underbrace{\begin{pmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{pmatrix}}_{2 \text{ butterfly of size } \frac{n}{2}} \times \underbrace{\begin{pmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{pmatrix}}_{1 \text{ butterfly of size } n}$$

Packed storage for a recursive butterfly matrix





Computational cost of SRBT

- The elementary operation is

$$B^T \times C \times B$$

  where B is a butterfly matrix and C is a square matrix, both of size $m \times m$
- $B^T CB$ requires $2m^2$ flops
- $A_r = U^T AU$ requires

$$C(n, d) \approx 2dn^2$$

  where U is a recursive butterfly of size n and depth d
- Maximum cost: $C(n, log_2 n + 1) \approx 2n^2 log_2 n$
- We aim at choosing d such that $d < log_2 n \ll n$
  In practice $d = 2$ is sufficient

Condition number of the randomized matrix

- The **multiplicative preconditioning** has to keep the CN as "unchanged" as possible

$$A_r = U^T A U \Rightarrow cond_2(A_r) \leqslant cond_2(U)^2 cond_2(A)$$

- Choosing the random values in $[e^{-1/20}, e^{1/20}]$, we get

$$cond_2(A_r) \leqslant 1.2214^d cond_2(A)$$

- In practice, $d = 2$: $cond_2(A_r) \leqslant 1.5\ cond_2(A)$

Tiled LDL^T Factorization

Decomposing A in $nt \times nt$ tiles, where each A_{ij} is a tile of size $nb \times nb$.
For $nt = 3$:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

Same decomposition can be applied to L and D:

$$LDL^T = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} D_{11} & & \\ & D_{22} & \\ & & D_{33} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ & L_{22}^T & L_{32}^T \\ & & L_{33}^T \end{bmatrix}$$

Tiled LDL^T Factorization

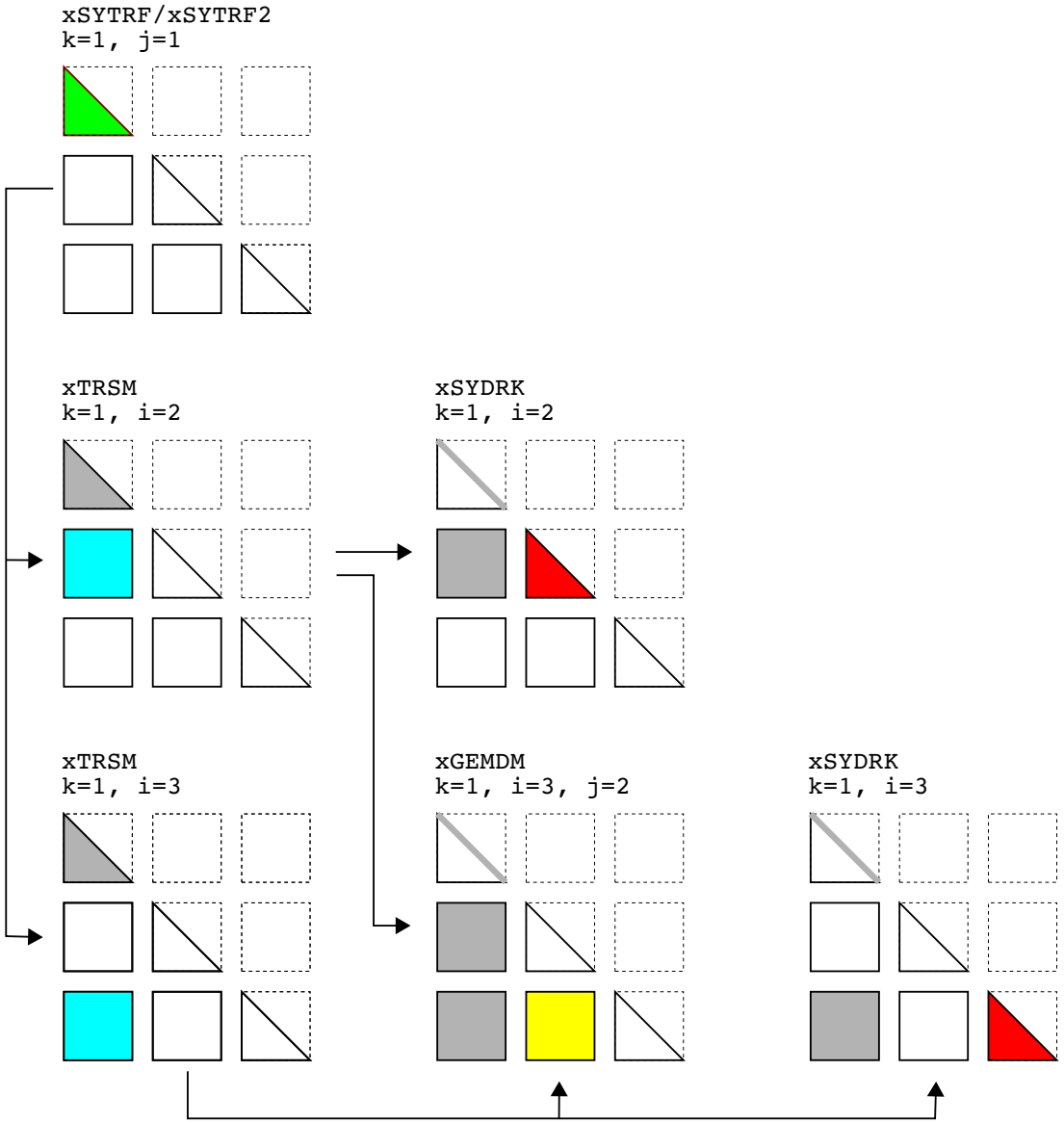
$$\left[\begin{array}{c|c|c} L_{11}D_{11}L_{11}^T & L_{11}D_{11}L_{21}^T & L_{11}D_{11}L_{31}^T \\ \hline L_{21}D_{11}L_{11}^T & L_{21}D_{11}L_{21}^T + L_{22}D_{22}L_{22}^T & L_{21}D_{11}L_{31}^T + L_{22}D_{22}L_{32}^T \\ \hline L_{31}D_{11}L_{11}^T & L_{31}D_{11}L_{21}^T + L_{32}D_{22}L_{22}^T & L_{31}D_{11}L_{31}^T + L_{32}D_{22}L_{32}^T + L_{33}D_{33}L_{33}^T \end{array}\right]$$

Tiled LDL^T Factorization

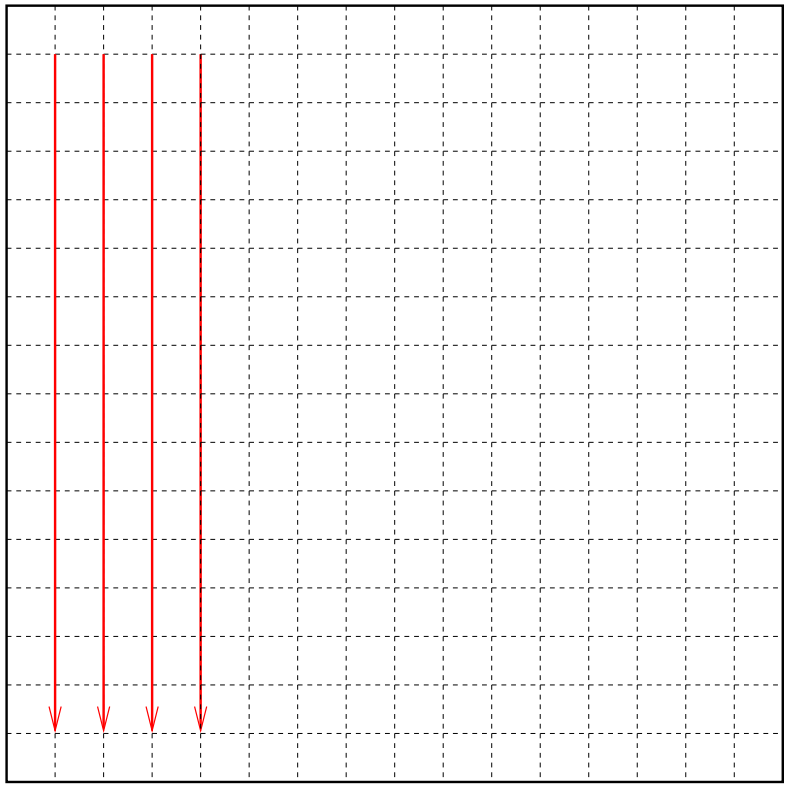
Using the same principle as the Schur complement, a series of tasks can be set to calculate each L_{ij} and D_{ii}:

$$
\begin{aligned}
[L_{11}, D_{11}] &= \mathbf{LDL}^T(A_{11}) \\
L_{21} &= A_{21}(D_{11}L_{11}^T)^{-1} \\
L_{31} &= A_{31}(D_{11}L_{11}^T)^{-1} \\
\tilde{A}_{32} &= A_{32} - L_{31}D_{11}L_{21}^T \\
\tilde{A}_{22} &= A_{22} - L_{21}D_{11}L_{21}^T \\
[L_{22}, D_{22}] &= \mathbf{LDL}^T(\tilde{A}_{22}) \\
L_{32} &= \tilde{A}_{32}(D_{22}L_{22}^T)^{-1} \\
\tilde{A}_{33} &= A_{33} - L_{31}D_{11}L_{31}^T - L_{32}D_{22}L_{32}^T \\
[L_{33}, D_{33}] &= \mathbf{LDL}^T(\tilde{A}_{33})
\end{aligned}
$$

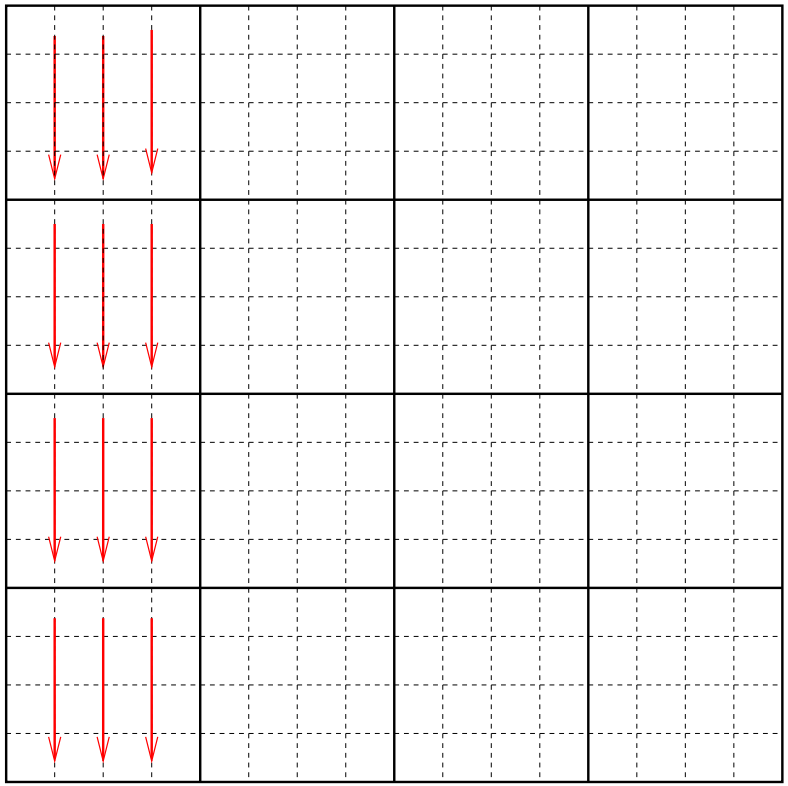
Tiled LDL^T Algorithm



Data organization



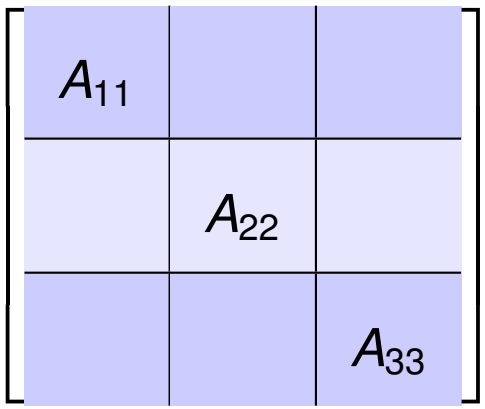
Column-major layout



Tile layout

Tiled LDL^T Factorization with tile-wise pivoting

- The tile-wise pivoting restricts the search of pivots to the tile A_{kk}
- Does not guarantee the accuracy of the solution; it strongly depends on the matrix to be factorized and how the pivots are distributed.
- Guarantees numerical stability of the factorization of each tile A_{kk}, with appropriate pivoting used
- Pivoting is sequential, which means that the pivot search and hence the permutations are also serial

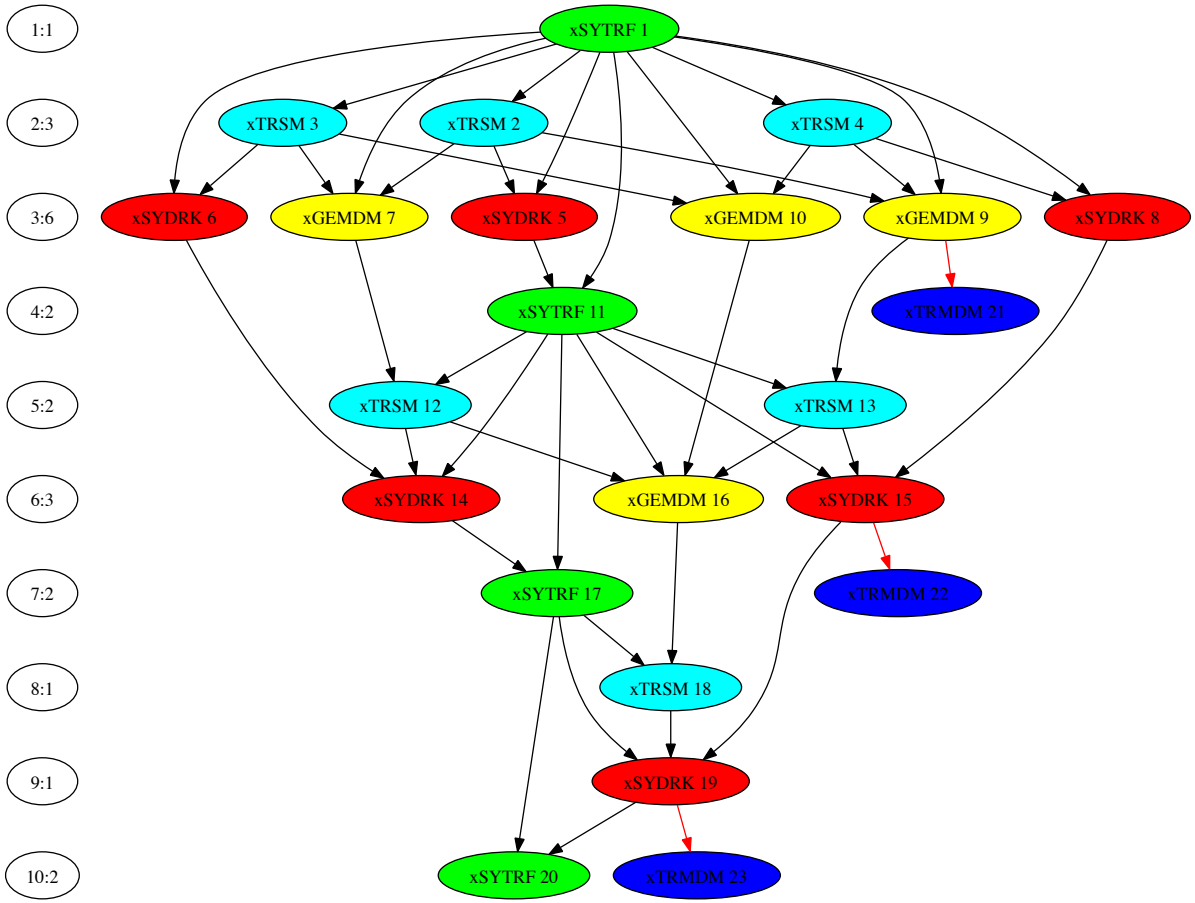


Tiled LDL^T factorization with pivoting

Adding pivoting to LDL^T:

$$
\begin{aligned}
[L_{11}, D_{11}, P_{11}] &= \mathbf{LDL}^T(A_{11}) \\
L_{21} &= P_{22}^T A_{21} P_{11} (D_{11} L_{11}^T)^{-1} \\
L_{31} &= P_{33}^T A_{31} P_{11} (D_{11} L_{11}^T)^{-1} \\
\tilde{A}_{22} &= A_{22} - (P_{22} L_{21}) D_{11} (P_{22} L_{21})^T \\
\tilde{A}_{32} &= A_{32} - (P_{33} L_{31}) D_{11} (P_{22} L_{21})^T \\
[L_{22}, D_{22}, P_{22}] &= \mathbf{LDL}^T(\tilde{A}_{22}) \\
L_{32} &= P_{33}^T \tilde{A}_{32} P_{22} (D_{22} L_{22}^T)^{-1} \\
\tilde{A}_{33} &= A_{33} - (P_{33} L_{31}) D_{11} (P_{33} L_{31})^T - (P_{33} L_{32}) D_{22} (P_{33} L_{32})^T \\
[L_{33}, D_{33}, P_{33}] &= \mathbf{LDL}^T(\tilde{A}_{33})
\end{aligned}
$$

DAG for tiled LDL^T factorization

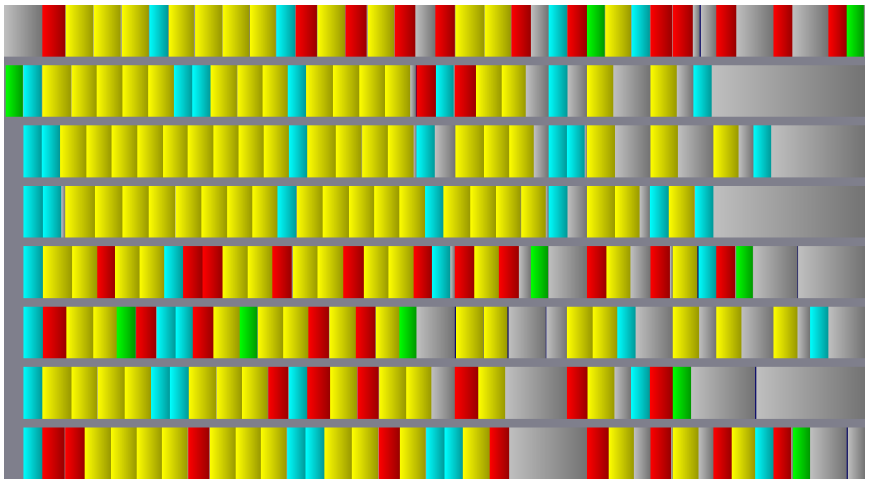


DAG for DSYTRF (4×4 tiles)

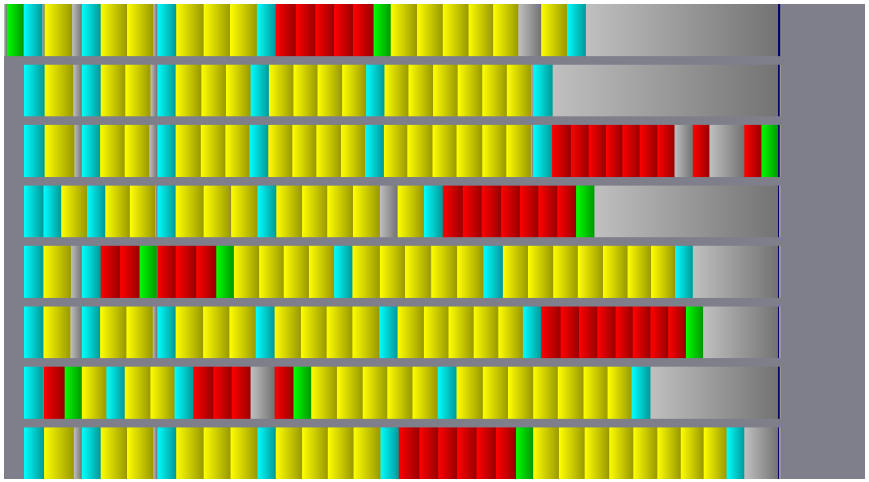
Scheduling issues

- Tasks are scheduled from the DAG, asynchronously and independently, as long as dependencies are respected.
- **Dynamic scheduling:**
  - Out-of-order execution, very few synchronization
  - Based on computational resources available
  - Overhead due to the scheduler (here QUARK)
- **Static scheduling:**
  - Each core's workload is predetermined
  - More difficult to implement
  - Take advantage of data locality and cache re-use.

Traces of tiled LDL^T (MagnyCours-48 with 8 threads)



Dynamic scheduling



Static scheduling

Accuracy Comparison

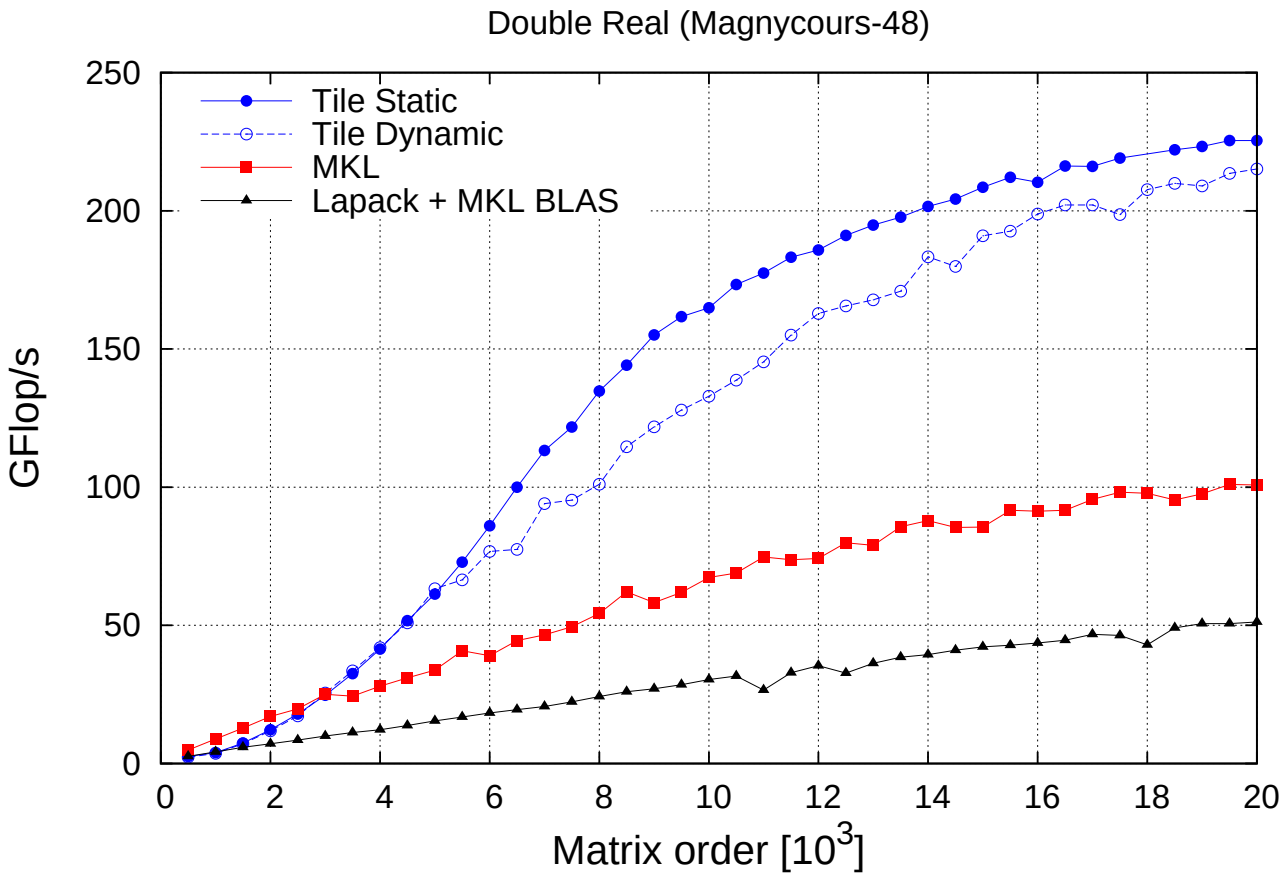
Matrix	Cond A	NP	PP	TP	SRBT (IR)
condex	$1 \cdot 10^{2}$	$5 \cdot 10^{-15}$	$6 \cdot 10^{-15}$	$7 \cdot 10^{-15}$	$6 \cdot 10^{-15}$ (0)
fiedler	$7 \cdot 10^{5}$	Fail	$2 \cdot 10^{-15}$	$7 \cdot 10^{-15}$	$9 \cdot 10^{-15}$ (0)
orthog	$1 \cdot 10^{0}$	$8 \cdot 10^{-1}$	$1 \cdot 10^{-14}$	$5 \cdot 10^{-1}$	$3 \cdot 10^{-16}$ (1)
randcorr	$3 \cdot 10^{3}$	$4 \cdot 10^{-16}$	$3 \cdot 10^{-16}$	$4 \cdot 10^{-16}$	$5 \cdot 10^{-16}$ (0)
augment	$5 \cdot 10^{4}$	$7 \cdot 10^{-15}$	$4 \cdot 10^{-15}$	$8 \cdot 10^{-15}$	$2 \cdot 10^{-16}$ (1)
prolate	$6 \cdot 10^{18}$	$8 \cdot 10^{-15}$	$8 \cdot 10^{-16}$	$2 \cdot 10^{-15}$	$2 \cdot 10^{-15}$ (0)
toeppd	$1 \cdot 10^{7}$	$5 \cdot 10^{-16}$	$7 \cdot 10^{-16}$	$6 \cdot 10^{-16}$	$2 \cdot 10^{-16}$ (0)
$\lvert i-j \rvert$	$7 \cdot 10^{5}$	$2 \cdot 10^{-15}$	$2 \cdot 10^{-15}$	$7 \cdot 10^{-15}$	$1 \cdot 10^{-14}$ (0)
max(i,j)	$3 \cdot 10^{6}$	$2 \cdot 10^{-14}$	$2 \cdot 10^{-15}$	$5 \cdot 10^{-15}$	$1 \cdot 10^{-14}$ (0)
Hadamard	$1 \cdot 10^{0}$	0	0	0	$7 \cdot 10^{-15}$ (0)
rand0	$2 \cdot 10^{5}$	$1 \cdot 10^{-12}$	$7 \cdot 10^{-14}$	$1 \cdot 10^{-13}$	$1 \cdot 10^{-15}$ (1)
rand1	$2 \cdot 10^{5}$	Fail	$1 \cdot 10^{-13}$	$2 \cdot 10^{-11}$	$1 \cdot 10^{-15}$ (1)
rand2	$1 \cdot 10^{5}$	Fail	$5 \cdot 10^{-14}$	$6 \cdot 10^{-13}$	$1 \cdot 10^{-15}$ (1)
rand3	$8 \cdot 10^{4}$	$4 \cdot 10^{-13}$	$7 \cdot 10^{-14}$	$4 \cdot 10^{-13}$	$1 \cdot 10^{-15}$ (1)

Table: Component-wise backward error ($n = 1024$, tile size=8)

Performance results

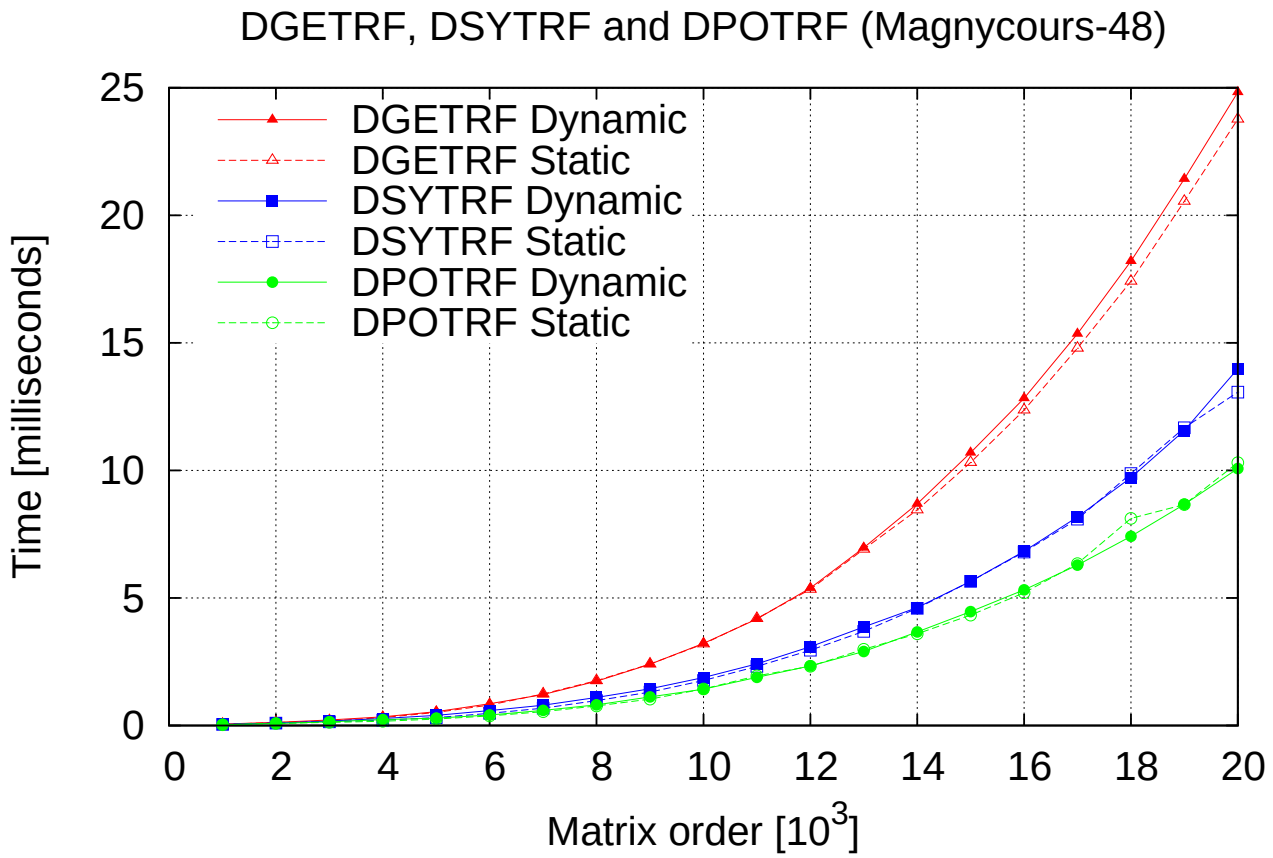
- Tiled LDL^T algorithm implemented following PLASMA development guidelines, tile size $NB = 250$.
- Machine: $4\times$ 12-Core AMD Opteron 6172 Magny-Cours @ 2.1 GHz, 128GB memory, theroretical peak 403.2 Gflop/s (8.4 Gflop/s per core) in double precision.
- Static and Dynamic scheduling.
- Comparisons against MKL library for multicore and LAPACK with multithreaded MKL BLAS
- Comparisons with other solvers (Cholesky and *LU*) from the PLASMA library.
- Scalability of LDL^T solver up de 48 cores.
- Details in [MB, Becker, Dongarra, IPDPS 2012]

Performance of the SRBT-LDL^T solver



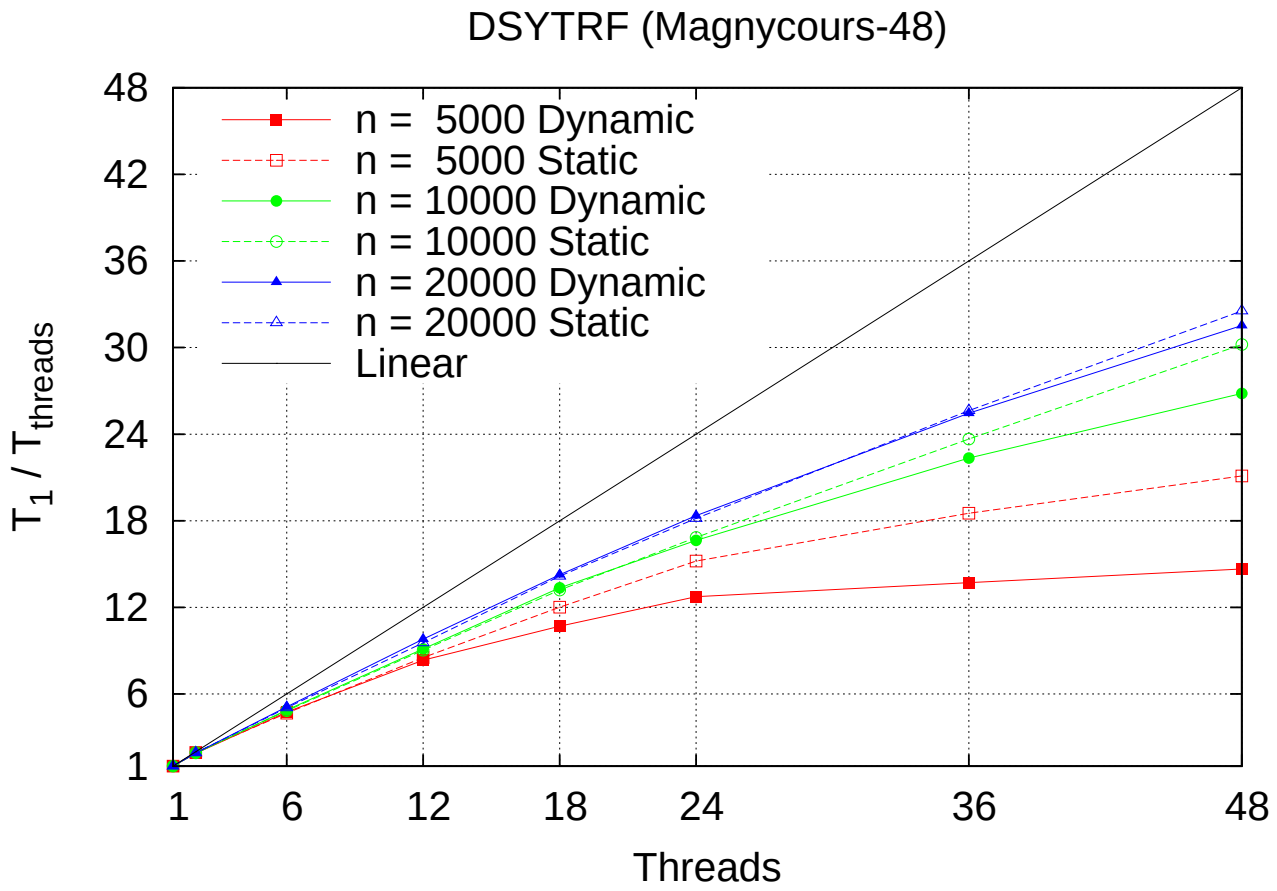
Performance of SRBT-LDL^T against MKL and LAPACK (double precision)

Tiled LDL^T vs other solvers



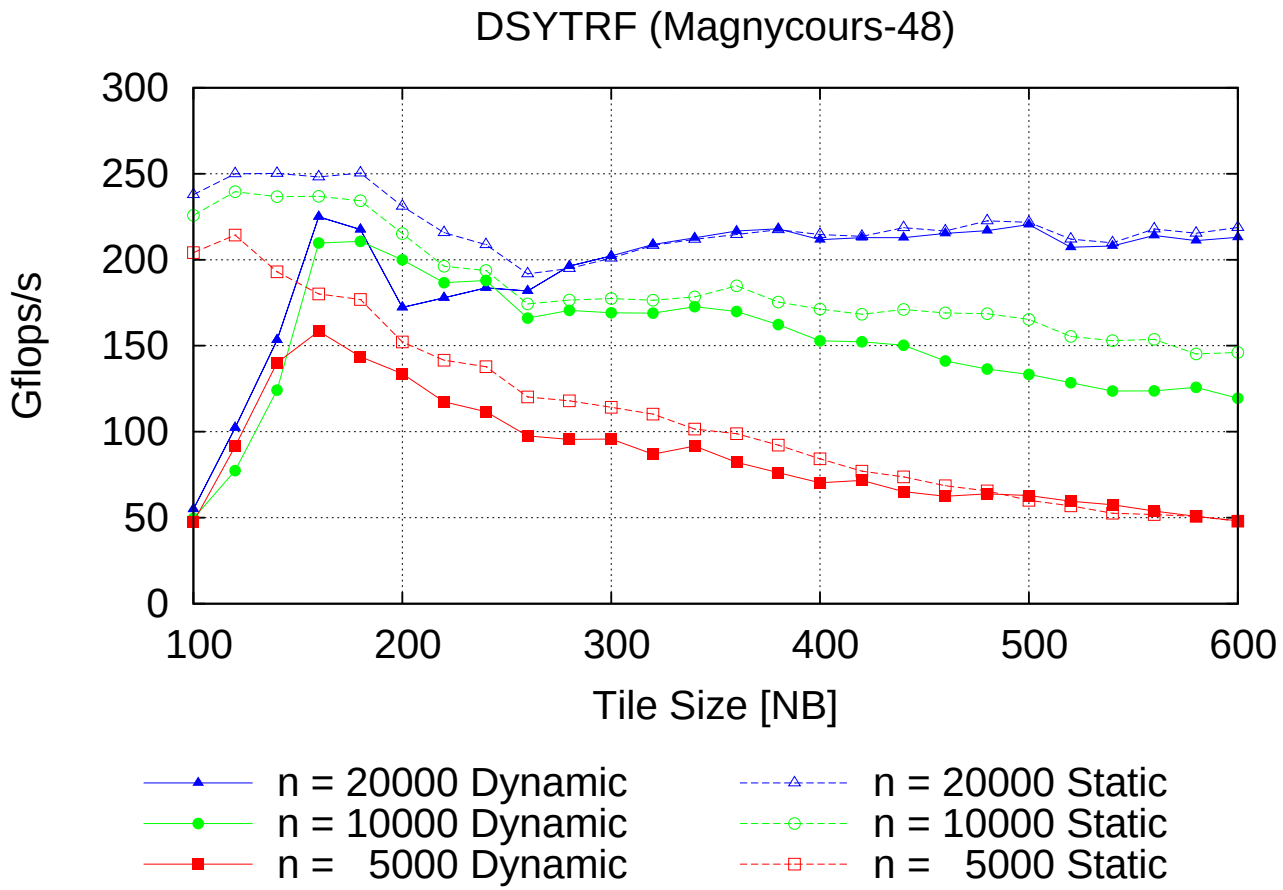
Execution time of Cholesky (PLASMA), LU (PLASMA) and tiled LDL^T, dynamic (solid line) and static (dashed line) scheduling.

Scalability of tiled LDL^T



Parallel speed-up; dynamic (solid line) and static (dashed line) scheduling.

Tuning elements



Tile-size performance of tiled LDL^T.

Summary

- Tiled LDL^T factorization without pivoting, and a randomization technique (SRBT) to avoid pivoting in LDL^T
- New class of linear system solvers based on various pivoting (or not) strategies
- Further optimizations are possible with e.g. additional tuning and scheduling

What next?

- Integration of randomized LDL^T solver in the next release of PLASMA
- Very soon extended to large-scale distributed systems and GPUs
- Minisymposium at SIAM Conference Applied Linear Algebra, Valencia (Spain), June 18-22, 2012:
  Application of Statistics to Numerical Linear Algebra Algorithms

Related papers

[1] M. Baboulin, J. Dongarra, J. Herrmann, S. Tomov,
Accelerating linear system solutions using randomization techniques.
ACM Transactions on Mathematical Software (TOMS), also LAPACK Working Note 246.

[2] M. Baboulin, D. Becker, J. Dongarra,
A Parallel Tiled Solver for Dense Symmetric Indefinite Systems on Multicore Architectures. *Proceedings of IPDPS 2012, also LAPACK Working Note 261*.

[3] D. Becker, M. Baboulin, J. Dongarra,
Reducing the amount of pivoting in symmetric indefinite systems. *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics (PPAM 2011)*.

[4] S. Tomov, J. Dongarra, M. Baboulin,
Towards dense linear algebra for hybrid GPU accelerated manycore systems.
Parallel Computing, Vol. 36, No 5&6, pp. 232-240 (2010).

[5] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, S. Tomov,
Accelerating scientific computations with mixed precision algorithms.
Computer Physics Communications, Vol. 180, No 12, pp. 2526-2533 (2009).

[6] M. Baboulin, J. Dongarra, S. Tomov,
Some issues in dense linear algebra for multicore and special purpose architectures.
Springer LCNS Series, 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'08).