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

Marc Baboulin and Dulceneia Becker and Jack Dongarra

Inria Saclay/ Université Paris-Sud

Innovative Computing Laboratory, University of Tennessee

Outline

Objective

Develop a parallel solver for dense symmetric indefinite linear 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

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

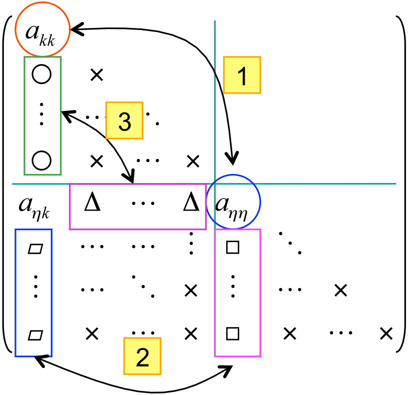
Pivoting

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

Symmetric pivoting

- To maintain symmetry, columns and rows must be interchanged

- Compromises data locality
- Increases data dependence



How to avoid pivoting?

Symmetric Random Butterfly Transformation (SRBT)

- To solve $Ax = b$:
  1. Compute $A_r = U^T A U$
  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**
- Requirements:
  - SRBT transformation must be cheap; $\mathcal{O}(n^2)$ operations
  - LDL^T with no pivoting must be fast

Butterfly Matrix

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

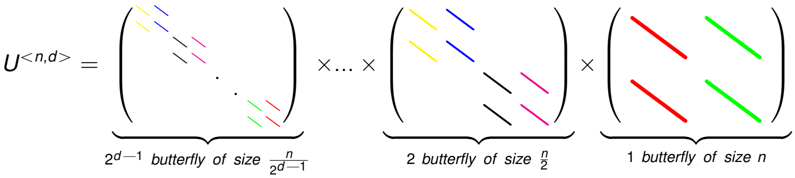
$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix}$$

where R and S are random diagonal matrices.

$$B = \begin{pmatrix} \ddots & \ddots \\ \ddots & \ddots \end{pmatrix}$$

Recursive Butterfly Matrix

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



We consider a limited number of recursions, resulting in a so-called **Partial Symmetric Random Butterfly Transformation (PSRBT)**.

Recursive butterfly matrix with d=3

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

U is $n \times n$

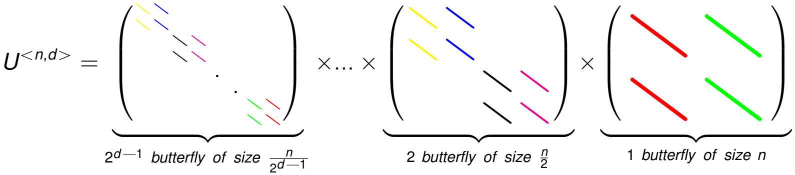
B_1 is $n \times n$

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

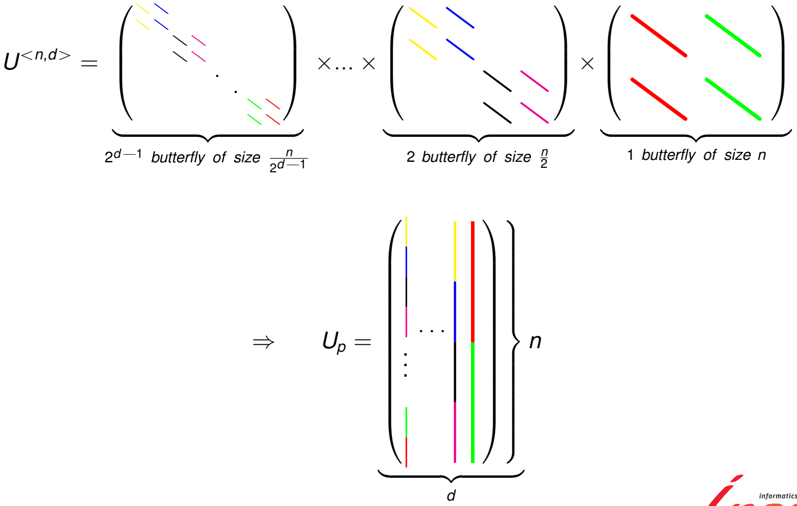
B_4 to B_7 are $\frac{n}{4} \times \frac{n}{4}$

$$B_i = \frac{1}{\sqrt{2}} \begin{pmatrix} R_i & S_i \\ R_i & -S_i \end{pmatrix}$$

Packed storage for a recursive butterfly matrix



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

- **2-norm condition number** (CN): measures the degree of distortion of the unit sphere under tranformation by a 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 factorization with pivoting

Adding pivoting to LDL^T:

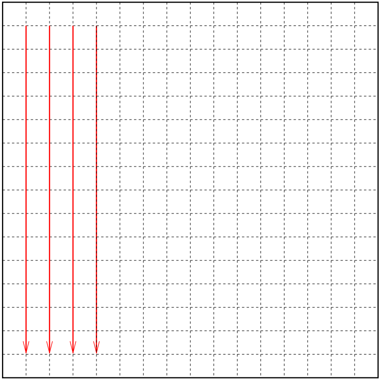
$$
\begin{aligned}
[L_{11}, D_{11}, P_{11}] &= \mathbf{LDL}^T(A_{11}) \\
L_{21} &= P_{22}^T A_{21} P_{11} (D_{11} L_{11}^T)^{-1} \\
L_{31} &= P_{33}^T A_{31} P_{11} (D_{11} L_{11}^T)^{-1} \\
\tilde{A}_{22} &= A_{22} - (P_{22} L_{21}) D_{11} (P_{22} L_{21})^T \\
\tilde{A}_{32} &= A_{32} - (P_{33} L_{31})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

Tiled LDL^T Factorization with tile-wise pivoting

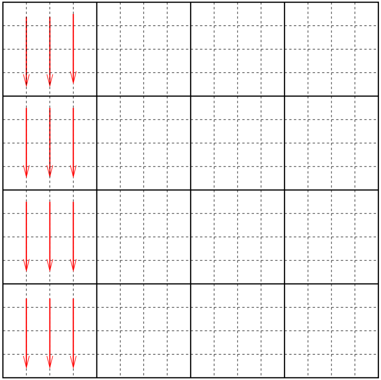
- The tile-wise pivoting restricts the search of pivots to the tile A_{kk}
- Does not guarantee the accuracy of the solution; it strongly depends on the matrix to be factorized and how the pivots are distributed.
- Guarantees numerical stability of the factorization of each tile A_{kk}, with appropriate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

$$\begin{bmatrix} A_{11} & & \\ & A_{22} & \\ & & A_{33} \end{bmatrix}$$

Data organization

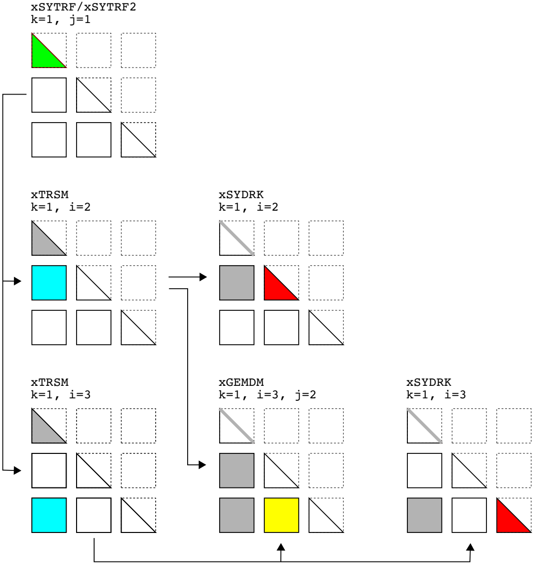


Column-major layout

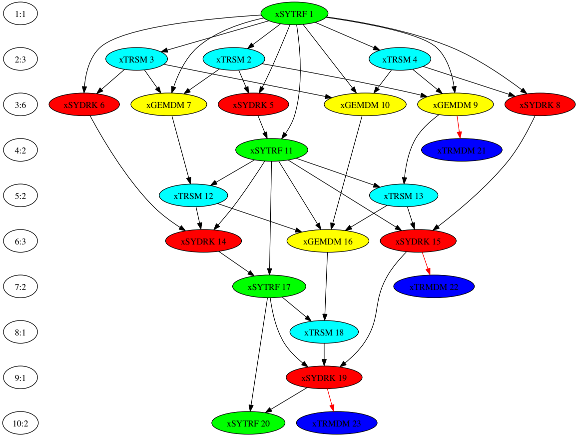


Tile layout

Tiled LDL^T Algorithm

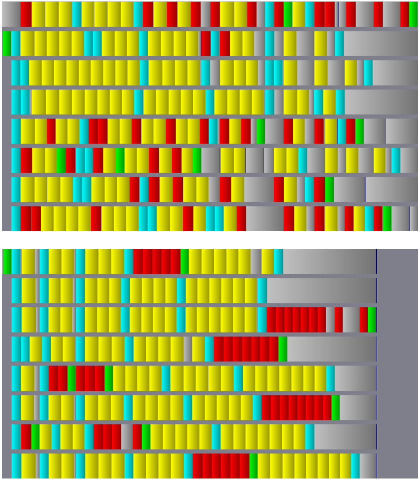


Static and dynamic scheduling



DAG for DSYTRF ($nt = 4$)

Static and dynamic scheduling



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

Tests on accuracy

- We compare 3 solvers:
  - SRBT (2 recursions) + Tiled LDL^T
  - LAPACK LDL^T (Bunch-Kaufman pivoting strategy)
  - SRBT + PP
- We report componentwise backward error $\omega = \max_i \frac{|Ax-b|_i}{(|A|\cdot|x|+|b|)_i}$
- Iterative refinement is systematically added
- Test matrices from the LAPACK tester:

1	Diagonal	6	Random, $\kappa = 2$
2	First column zero	7	Random, $\kappa = \sqrt{1/\varepsilon}$
3	Last column zero	8	Random, $\kappa = 1/\varepsilon$
4	Middle column zero	9	Scaled near underflow
5	Last $n/2$ columns zero	10	Scaled near overflow

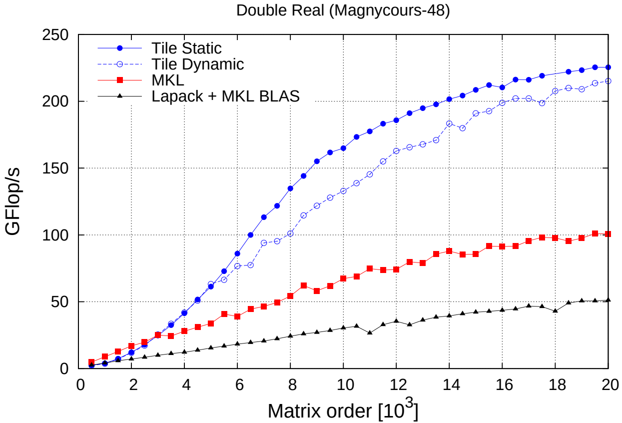
Accuracy Comparison

Matrix Type	SRBT-LDL^T	LAPACK LDL^T	SRBT + LDL^T PIV
1	0.8815e-13 (0)	0.1079e-15 (0)	0.1975e-13 (0)
2	0.4067e-13 (1)	0.2830e-13 (+)	0.4244e-13 (1)
3	0.2395e-13 (1)	-	0.1242e-13 (1)
4	0.2504e-13 (1)	-	0.3696e-13 (1)
5	0.5466e-13 (1)	-	0.8008e-13 (1)
6	-	-	0.1219e-13 (1)
7	0.3037e-13 (1)	0.3810e-13 (+)	0.6795e-13 (1)
8	0.6048e-13 (1)	0.2930e-13 (+)	0.5195e-13 (0)
9	-	0.5898e-13 (+)	0.2212e-13 (1)
10	0.3674e-13 (1)	0.8683e-13 (+)	0.1612e-13 (1)

Preliminary performance results

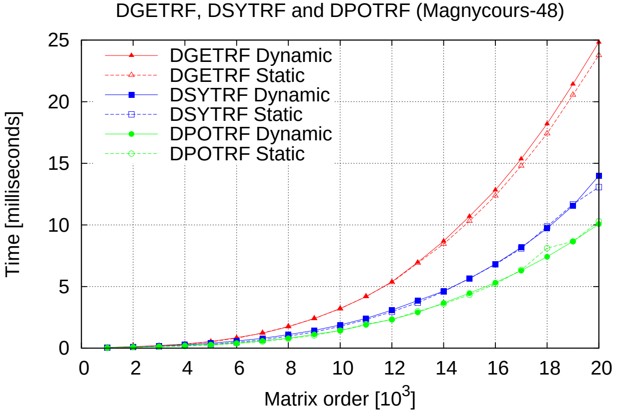
- Tiled LDL^T algorithm implemented following PLASMA development guidelines, tile size $nb = 250$.
- Machine: $4\times$ 12-Core AMD Opteron 6172 Magny-Cours @ 2.1 GHz, 128GB memory, theroretical peak 403.2 Gflop/s (8.4 Gflop/s per core) in double precision.
- Comparisons against MKL library for multicore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

Performance of the SRBT-LDL^T solver



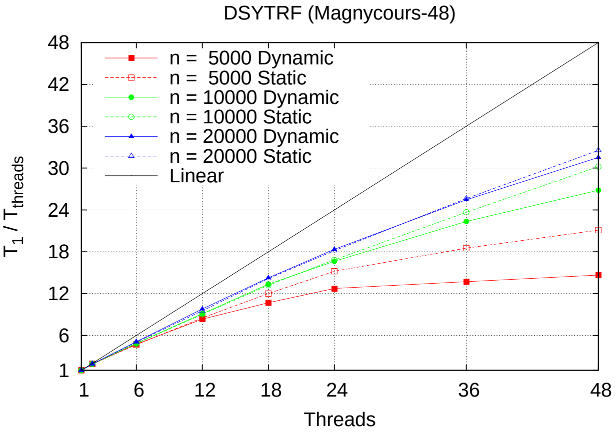
Performance of SRBT-LDL^T against MKL and LAPACK (double precision)

Tiled LDL^T vs other solvers



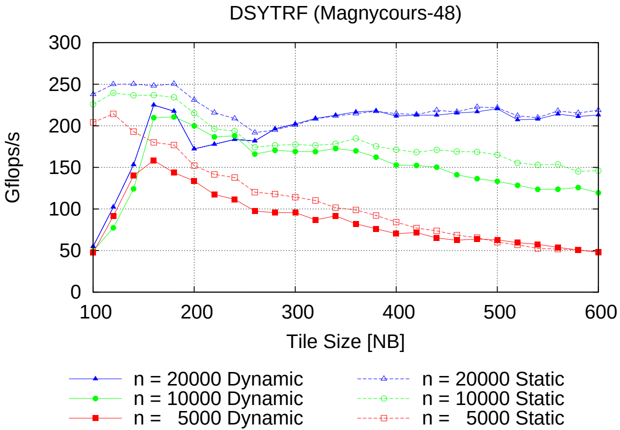
Execution time of Cholesky (PLASMA), LU (PLASMA) and tiled LDL^T, dynamic (solid line) and static (dashed line) scheduling.

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
- SRBT is computationally very affordable and negligible compared to the communication overhead due to classical pivoting
- It gives accurate results on most test cases including pathological ones
- Very promising performance that makes SRBT+tiled LDL^T very competitive compared to other solvers

What next?

Integration into the PLASMA library (next release).
Development of LDL^T on GPU for integration into MAGMA.

Collaboration with UIUC

- **Fast linear solvers for CPU/GPU architectures**
- People:
  - Wen-Mei Hwu (UIUC)
  - Liwen Chang (UIUC)
  - Marc Baboulin (Inria Saclay)
  - Laura Grigori (Inria Saclay)
  - Adrien Remy (Inria Saclay)
  - Yushan Wang (Inria Saclay)
- Tridiagonal solver developed at UIUC (integration into MAGMA for utilization in a CFD application at INRIA)
- Randomized algorithms for GPUs
- Communication avoiding algorithms for GPUs