Mixing LU and QR on multi-core cluster systems

ICL Friday Lunch

J. Herrmann, M. Faverge, J. Langou, Y. Robert

July 6, 2012

Outline

1. How to solve a linear system?
   - LU Factorizations
   - QR Factorizations
2. Mixing LU and QR kernels
3. Hierarchical Partial Pivoting (HPP)
4. Accuracy study
5. Current work
6. Conclusion

Outline

How to solve a linear system?

- Goal: Solve a linear system $Ax = b$ where A is a $n \times n$ matrix on large cluster of multicores.
- Existing Solutions:
  - **Factorization $A = LU$**,
    $\hookrightarrow \frac{2}{3}n^3$ floating-point operations
    L unit lower triangular and U upper triangular.
    Solve $Ax = b$ by solving successively
    $Ly = b$ and $Ux = y$
  - **Factorization $A = QR$**,
    $\hookrightarrow \frac{4}{3}n^3$ floating-point operations
    Q orthogonal matrix and R upper triangular.
    Solve $Ax = b$ by solving $Rx = Q^T b$

How to solve a linear system?

- Goal: Solve a linear system $Ax = b$ where A is a $n \times n$ matrix on large cluster of multicores.
- Existing Solutions:
  - **Factorization $A = LU$**,
    $\hookrightarrow \frac{2}{3}n^3$ floating-point operations
    L unit lower triangular and U upper triangular.
    Solve $Ax = b$ by solving successively
    $Ly = b$ and $Ux = y$
  - **Factorization $A = QR$**,
    $\hookrightarrow \frac{4}{3}n^3$ floating-point operations
    Q orthogonal matrix and R upper triangular.
    Solve $Ax = b$ by solving $Rx = Q^T b$

LU Factorizations

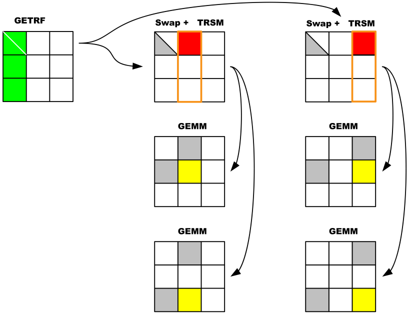
Tile Algorithms

- Partial pivoting
- Incremental pivoting
- Tournament pivoting

LU Factorizations

Partial Pivoting

- ☺ Highly parallel update (GEMM)
- ☺ Good stability
- ☹ Synchronization on each panel to apply the pivots
- ☹ Involves row swapping

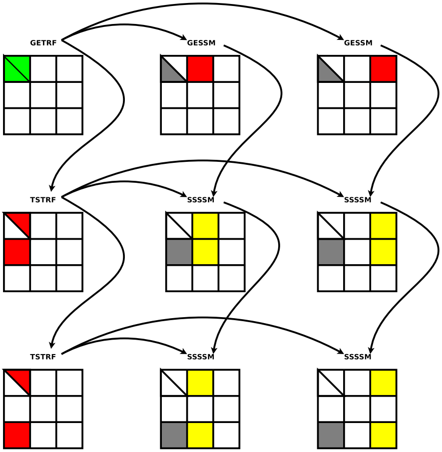


DAG for one step of LU Partial Pivoting

LU Factorizations

Incremental Pivoting

- ☺ Updates released before the end of panel factorization
- ☹ Update kernels don't exceed 80% of peak
- ☹ Extra computations compared to LUPP
- ☹ Less stable than LUPP
- ☹ Pipeline of communications on each column

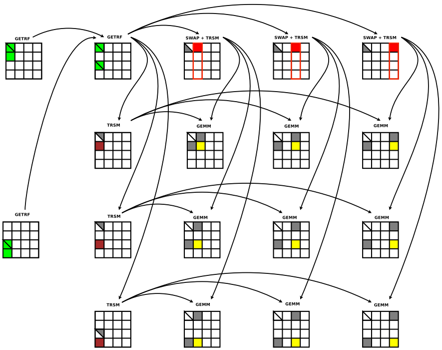


DAG for one step of LU

LU Factorizations

Tournament Pivoting

- ☺ Highly parallel update (GEMM)
- ☺ Enough stability
- ☺ Reduce the number of synchronisations
- ☹ Synchronization on each panel to apply the pivots
- ☹ Involves row swapping
- ☹ Extra computations compared to LUPP
- ☹ Less stable than LUPP



DAG for one step of LU

QR Factorizations

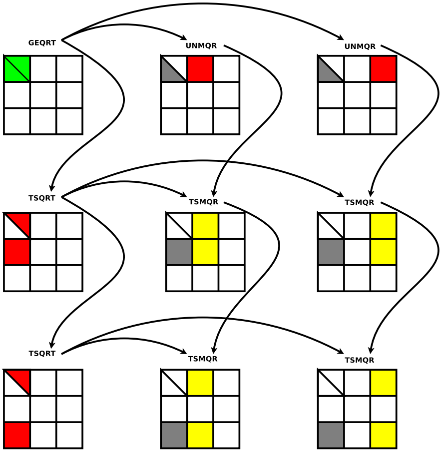
Tile Algorithms

- Flat tree using TS kernels
- Hierachical trees using TS and TT kernels

QR Factorizations

Flat tree algorithm

- ☺ More stable than LU factorization
- ☺ Some updates are released before the panel factorization is completed
- ☹ Update kernels don't exceed 80% of peak
- ☹ Twice as costly as LU factorization
- ☹ Pipeline of communications on each column

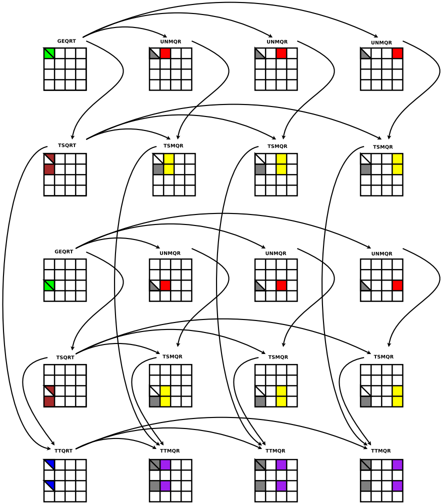


DAG for one step of QR

QR Factorizations

Hierarchical QR Factorization

- ☺ More stable than LU factorization
- ☺ Updates released before the end of panel factorization
- ☺ Communications reduced
- ☺ Wide space of composability on each panel
  Flat, Greedy, Binary, Fibonacci, . . .
- ☹ Update kernels don't exceed 80% of peak
- ☹ Twice as costly as LU factorization



DAG for one step of QR

Outline

Mixing LU and QR

Main Idea

- Algorithms have the same patern:

$$A = \begin{pmatrix} A_{11} & C \\ B & D \end{pmatrix} \Leftrightarrow \begin{pmatrix} \textit{factor} & \textit{apply} \\ \textit{kill} & \textit{update} \end{pmatrix}$$

- Keep the flops as close as possible to LU
- Remove the communications along the panel
  Incremental Pivoting or Flat QR
- Panel done as in Tournament pivoting or HQR
- Remove the swap on the trailing submatrix
  - Suppress the synchronization at each step
  - Reduce the number of communications
- Does not give a QR or LU factorization
- We apply all transformations on $\tilde{A} = (A, b)$

Mixing LU and QR

Main Idea

Diagonal-based algorithms

- One killer per panel
- Killer constructed with 1 or $f > 1$ tiles
- The killer kills all remaining tiles
- f can be determined dynamically

Multi-killer algorithms

- Several killers per panel
- In each domain, killer constructed with 1 or $f > 1$ tiles
- The killer kills the remaining tiles of its own domain
- Use a reduction tree across domains

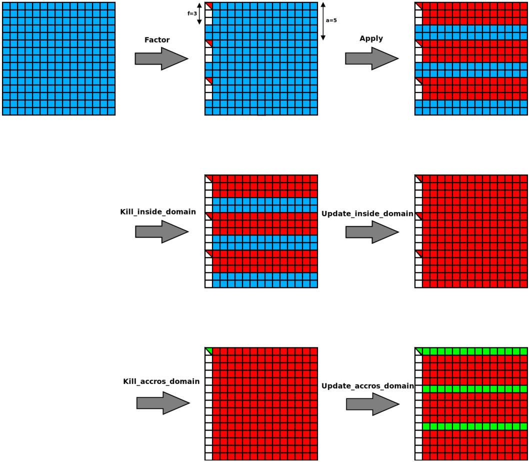


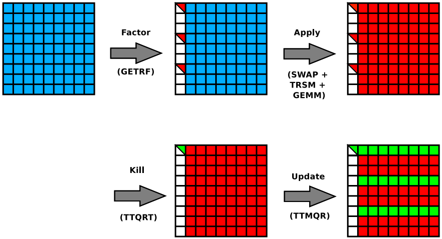
Illustration of the multi-domain approach

Outline

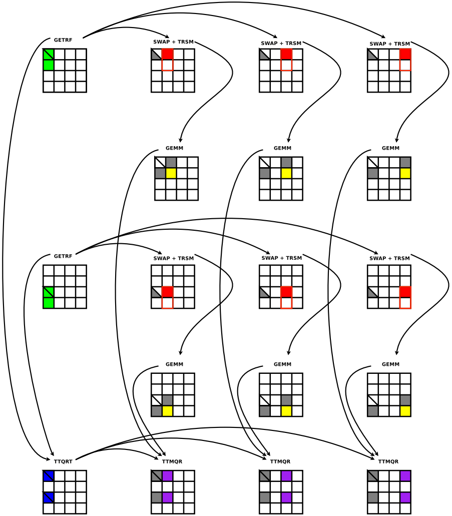
Hierarchical Partial Pivoting (HPP)

HPP

- *Factor & Apply*
  ↪ LU with Partial pivoting & Swap + TRSM + GEMM
- *Kill_inside_domains & Update_inside_domains*
  ↪ Non-existent since $f = a$
- *Kill_accross_domains & Update_accross_domains*
  ↪ *TTQRT* & *TTMQR*



HPP pattern, on one processor, for a 9×9 tiled matrix, and with $a = 3$



DAG for one step of the HPP algorithm, on one processor, for a 4×4 tiled matrix, and with $a = 2$

Hierarchical Partial Pivoting (HPP)

Kernels Cost

- LUPP kernels are highly parallel, more performant...
- QR is more stable, but more costly

	LU-based		QR-based	
factor A	$2/3$	GETRF	$4/3$	GEQRT
kill B	$(n-1)$	TRSM	$2(n-1)$	TSQRT
apply C	$(n-1)$	TRSM	$2(n-1)$	UNMQR
update D	$2(n-1)^2$	GEMM	$4(n-1)^2$	TSMQR

Table: Computational cost of each kernel. The unit is n_b^3 flops.

Hierarchical Partial Pivoting (HPP)

Computational cost

Proof.

$$
\begin{aligned}
C_{HPP} &= \sum_{k=0}^{n-1} \frac{n-k}{a} \times (C_{GETRF} + (n-k) \times C_{TRSM} + (n-k) \times (a-1) \times C_{GEMM}) \\
&\quad + (\frac{n-k}{a} - 1) \times (C_{TTQRT} + (n-k) \times C_{TTMQR}) \\
&\approx \frac{1}{a} \times \frac{n^3}{3} \times (C_{TRSM} + (a-1) \times C_{GEMM} + C_{TTMQR}) \\
&\approx \frac{1}{a} \times \frac{n^3}{3} \times (n_b^3 + (a-1) \times 2n_b^3 + 2n_b^3) \\
&\approx (1 + \frac{1}{2a}) \times \frac{2}{3} N^3
\end{aligned}
$$

□

Hierarchical Partial Pivoting (HPP)

Computational cost

$$C_{HPP} \approx (1 + \frac{1}{2a}) \times \frac{2}{3}N^3$$

Remarks

- When $a = \infty$, $C_{HPP} \approx C_{LU}$
- When $a = 1$, $C_{HPP} \approx N^3$
  - ↪ computation overhead of 50% as compared LU factorization
  - ↪ still smaller than $C_{QR} = \frac{4}{3}N^3$

Hierarchical Partial Pivoting (HPP)

Implementation

- Use DAGuE runtime
  ↪ we just specify the task graph
  ↪ deal with MPI communications and shared memory accesses
- Data flow description of the algorithm
  ↪ tasks are local
  ↪ problem with GETRF and SWAP

Outline

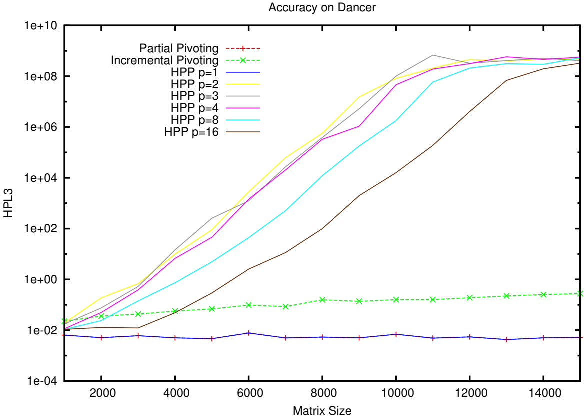
Accuracy study

Framework

- Every test in distributed memory has been run on Dancer
  $\hookrightarrow$ 16 nodes of 8 cores: Intel(R) Xeon(R) CPU E5520 @ 2.27GHz
- Every test in shared memory has been run on Remus
  $\hookrightarrow$ 48 shared-memory processors: AMD Opteron(tm) Processor 6180 SE
- The algorithm was run with a block size equal to $n_b = 240$.
- A and b are random matrix and vector
- $$HPL3 = \frac{||Ax - b||_\infty}{||A||_\infty ||x||_\infty \times N \times \epsilon}$$

Accuracy study

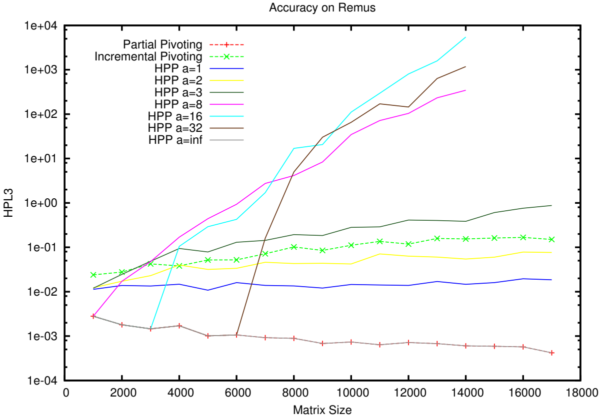
Distributed memory results



Accuracy of HPP depending on the number of processors *p* on Dancer

Accuracy study

Shared memory results



Accuracy of HPP depending on the domain size *a* on Remus

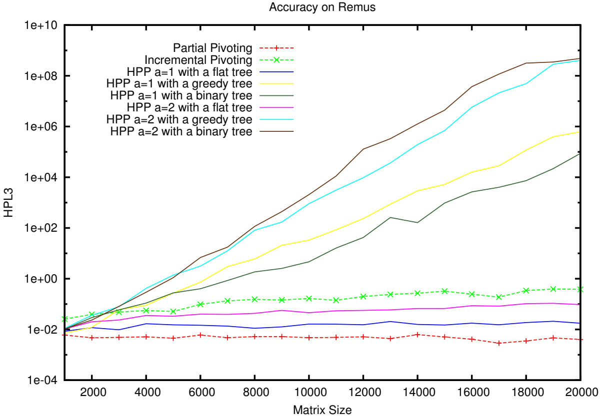
Accuracy study

Growth Factor

- The worst-case bound growth factor of HPP is the same as that of LU with Partial Pivoting
  - ↪ The domains are factored using LU with Partial Pivoting
  - ↪ The QR reduction across domains is an orthogonal transformation
- So why is HPP unstable?
- Where does the difference of stability between the shared memory and the distributed versions come from?

Accuracy study

Impact of the tree



Impact of the reduction tree on the accuracy of HPP

Outline

1. How to solve a linear system?
   - LU Factorizations
   - QR Factorizations
2. Mixing LU and QR kernels
3. Hierarchical Partial Pivoting (HPP)
4. Accuracy study
5. Current work
6. Conclusion

Current work

New version?

- LU factorization on the diagonal domain
- If result **good enough**, update with a Cholesky-like algorithm
- If not, do a QR factorization at this step

Problem

What is good enough? $(K * cond_2(A_{k,k}) >= \sum_{j=k}^{Mt}(||A_{k,j}||_1)$

Current work

New version?

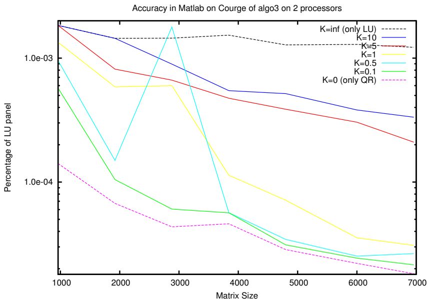
- LU factorization on the diagonal domain
- If result **good enough**, update with a Cholesky-like algorithm
- If not, do a QR factorization at this step

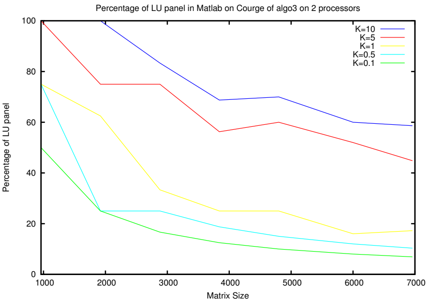
Problem

What is good enough? $(K * cond_2(A_{k,k}) >= \sum_{j=k}^{Mt}(||A_{k,j}||_1)$

Current work

Accuracy





Outline

Conclusion

- We introduced a taxonomy for algorithms mixing LU and QR kernels
- We described and implemented one of them
- A second version is in progress
- Results turned out to be less positive than expected
- But encouraging for dynamic algorithms