Solving general dense linear systems on hybrid multicore-GPU systems

Adrien RÉMY, Marc BABOULIN

June 14, 2012

Outline

The issue of pivoting in linear systems

- General square system $Ax = b$, solved by **Gaussian Elimination**
- Difficulties when zero or small diagonal elements
  $\rightarrow$ interchange rows: **partial pivoting (GEPP)**
- GEPP is implemented in most numerical libraries (LAPACK...).
  Used in the LINPACK benchmark for TOP500 list
- Factorization $PA = LU$, where P is a permutation matrix
- No floating point operation is performed in pivoting but it involves irregular movements of data
- **Communication overhead due to pivoting :** $O(n^2)$ comparisons

Pivoting is expensive

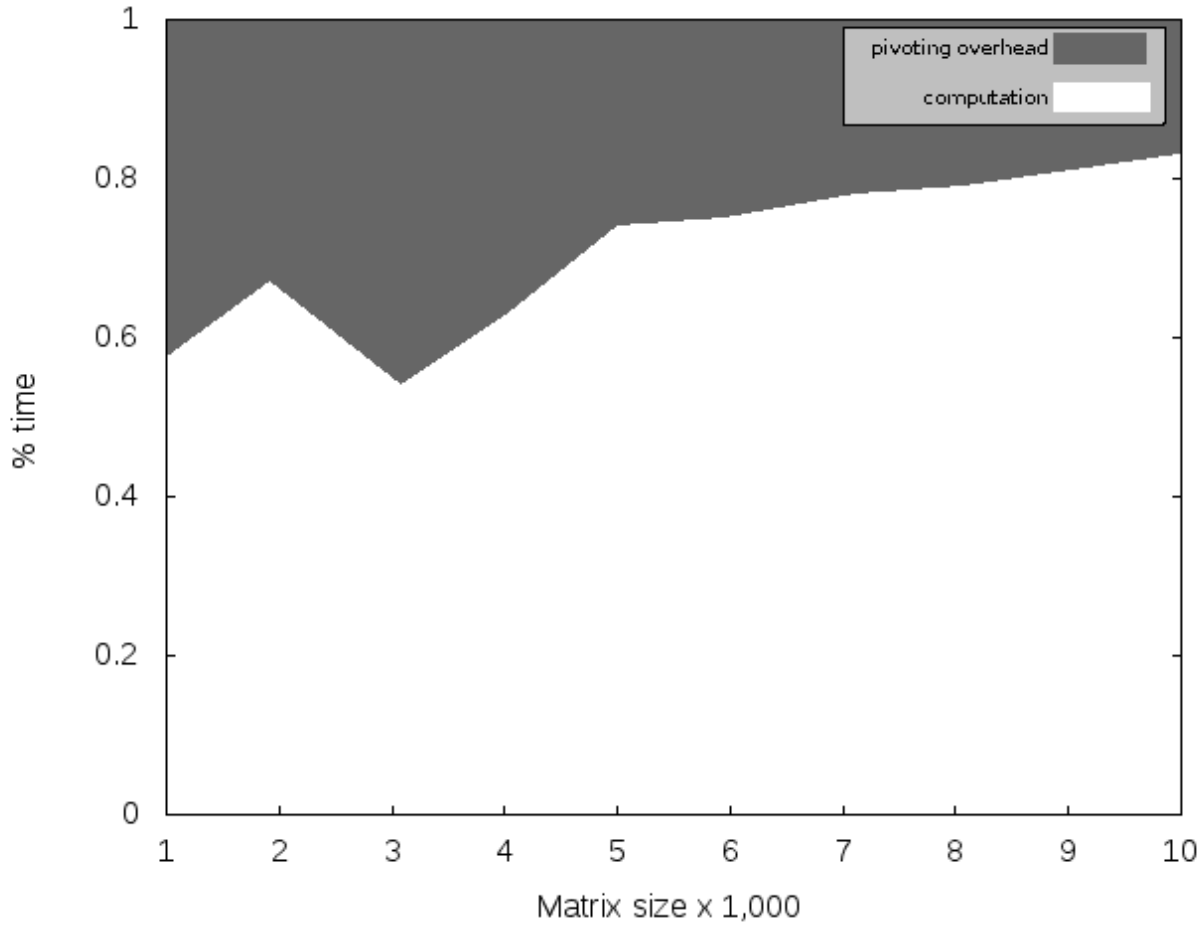
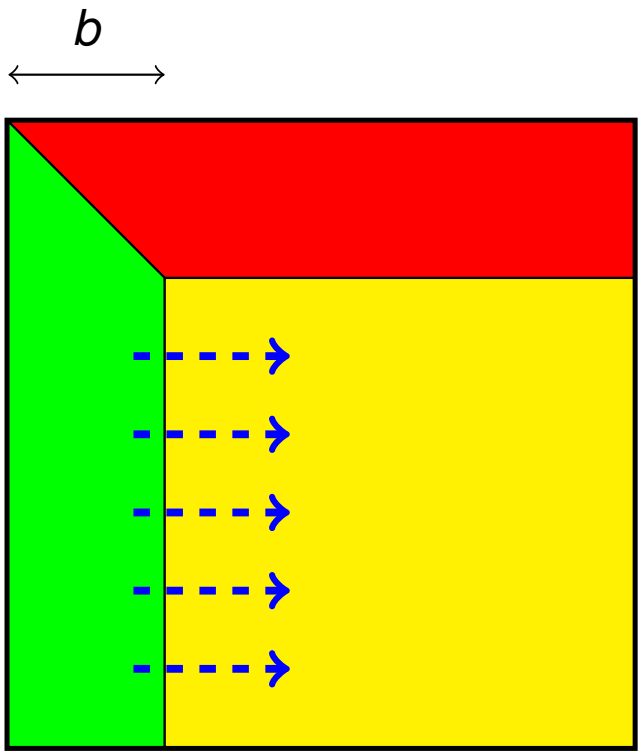


Figure: Cost of partial pivoting in LU factorization (MAGMA)
CPU 1 $\times$ Quad-Core Intel Core2 Q9300 @ 2.50 GHz - GPU C2050 @ 1.15 GHz

Right-looking block LU factorization

- **Factorization** $\rightarrow A = L * U$
- **Pivoting** $\rightarrow P * A = L * U$

1. Panel (block column) is factored using Gaussian elimination.
2. Permutations are applied to trailing submatrix.
3. Solve triangular system to compute the b first rows.
4. Update trailing submatrix.



LU implementation for GEPP in MAGMA

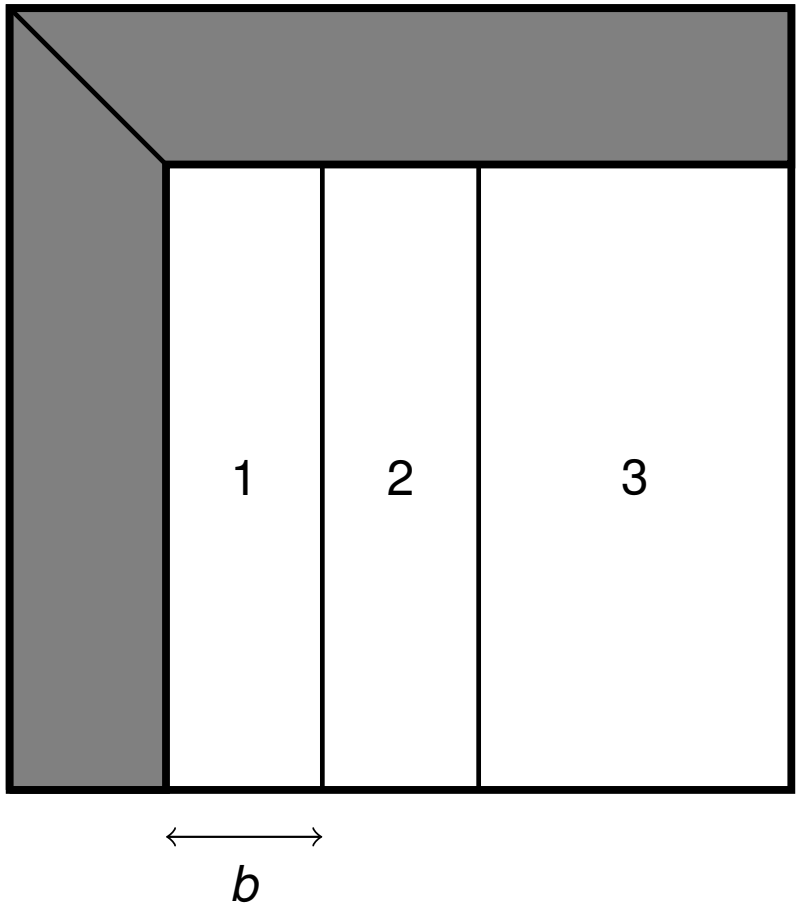


Figure: Block splitting in hybrid LU factorization

MAGMA implementation

Initial matrix has been transferred to the GPU.
Current iteration:

- Current panel (1) is downloaded to the CPU
- (1) is factored by the CPU using GEPP and the result is sent back to the GPU
- The GPU updates (2) (next panel)
- The updated panel (2) is sent back to the CPU to be factored while the GPU updates the rest of the matrix (3) (look-ahead)

Communication issues:

- Only panels are transferred between CPU and GPU ($O(n*b)$ data vs $O(n*n*b)$ computation in the updates)
- Total overlap of the panel computation by the updates for n large enough.

PRBT Solver

- PRBT (Partial Random Butterfly Transformation) is an LU solver based on randomization (see [Baboulin et al., TOMS, to appear]).
- Using the PRBT solver, we solve the general linear system $Ax = b$ by the following steps:

Algorithm 1 Solving $Ax = b$ with PRBT

1: Compute randomized matrix $A_r = U^T AV$, with U and V recursive butterflies.
2: Factorize A_r with GENP.
3: Solve $A_r y = U^T b$.
4: Solution is $x = Vy$.

- Properties :
  - **Randomization is cheap** ($O(n^2)$ operations)
  - **GENP is fast** (take advantage of the GPU)
  - **Accuracy** is in practive similar to GEPP (with iterative refinement)

Communication-avoiding LU

Panel factorization (on CPU) based on **tournament pivoting**
(see [Grigori et al., SIMAX 2011])

Implemented as a reduction operation :

- Partition the panel in blocks
- Select in parallel a set of local pivots using PP
- Perform tournament on the local sets to select global pivots
- Global pivots are moved to diagonal positions and GENP is performed on the entire panel

Tournament pivoting

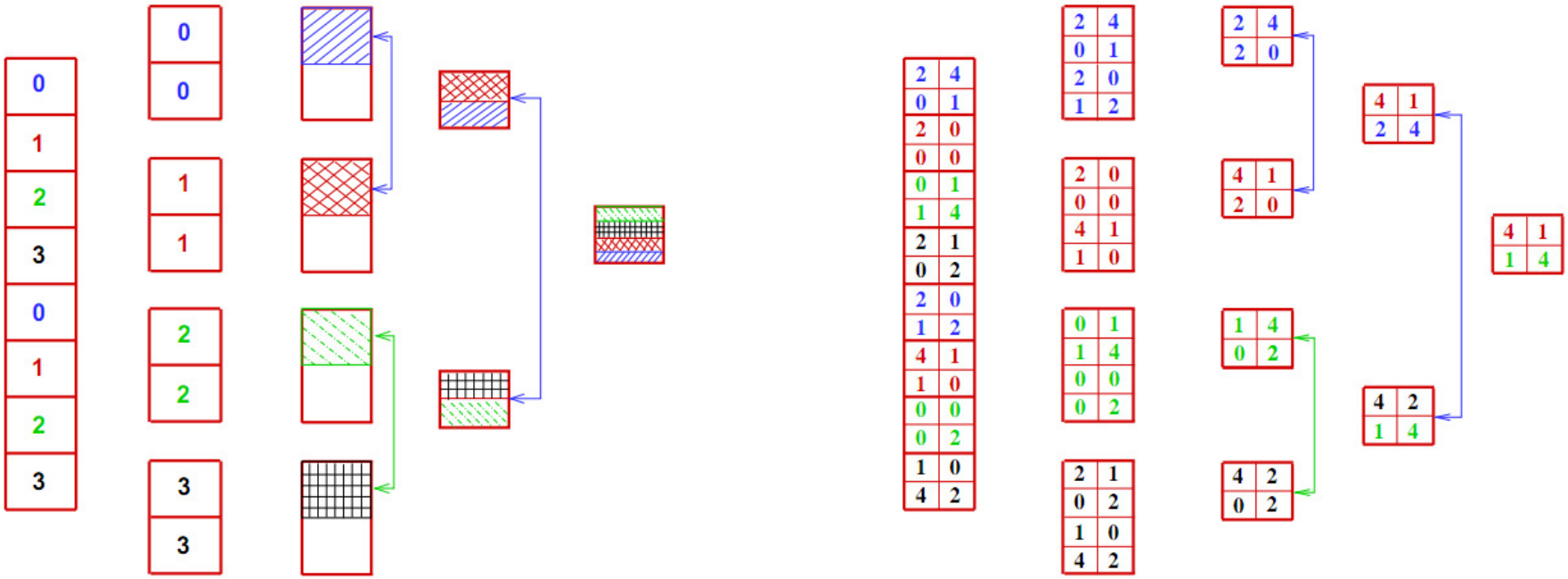


Figure: Panel factorization on CPU using Tall and Skinny LU with tournament pivoting

Hybrid version of CALU (H-CALU)

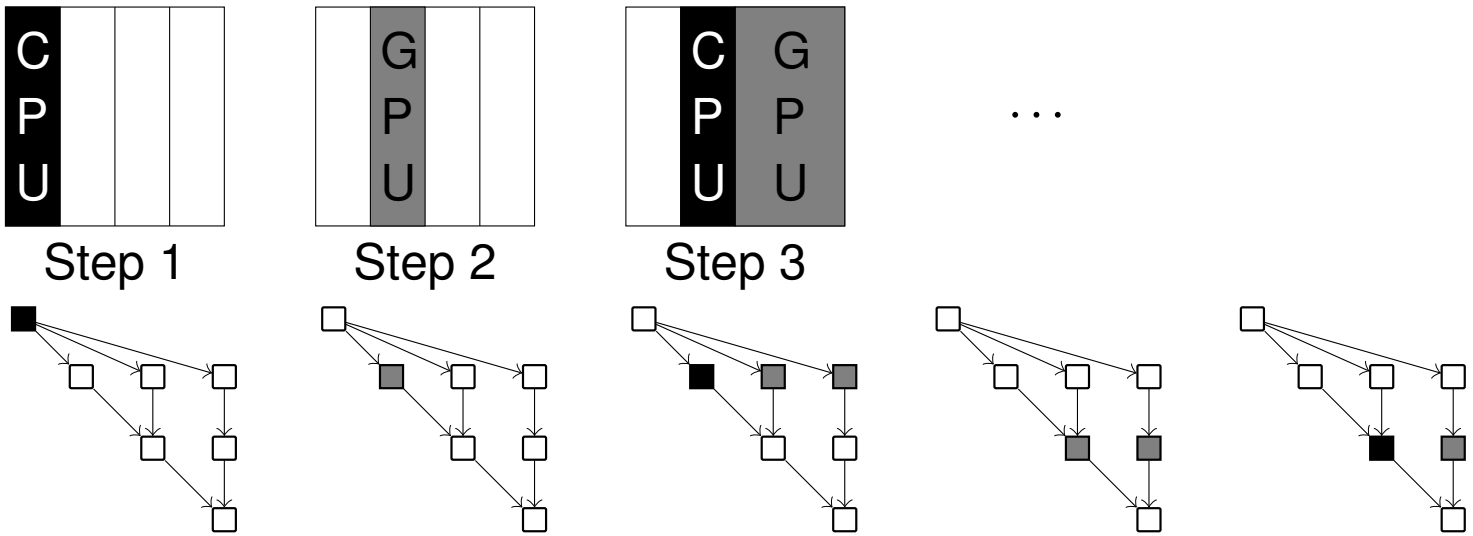
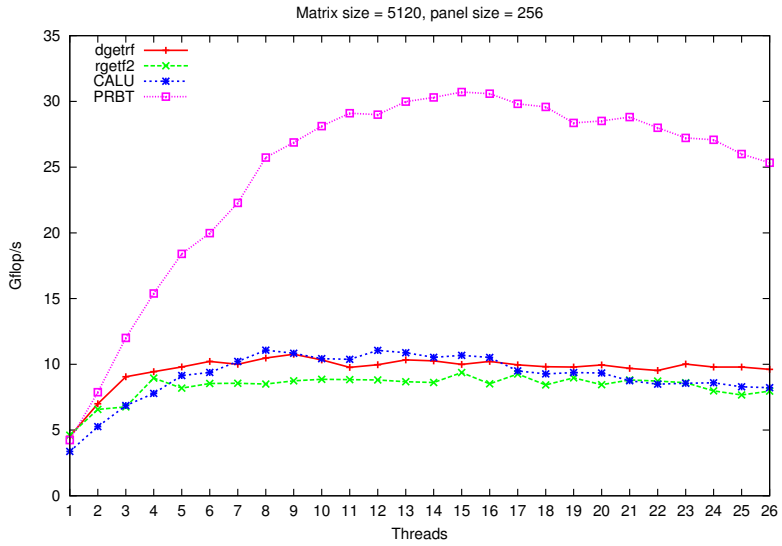


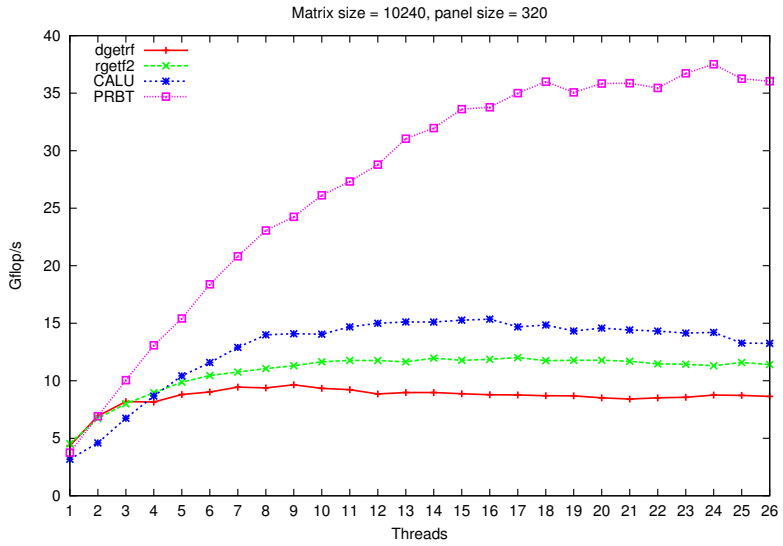
Figure: Hybrid LU factorization (4 panels).

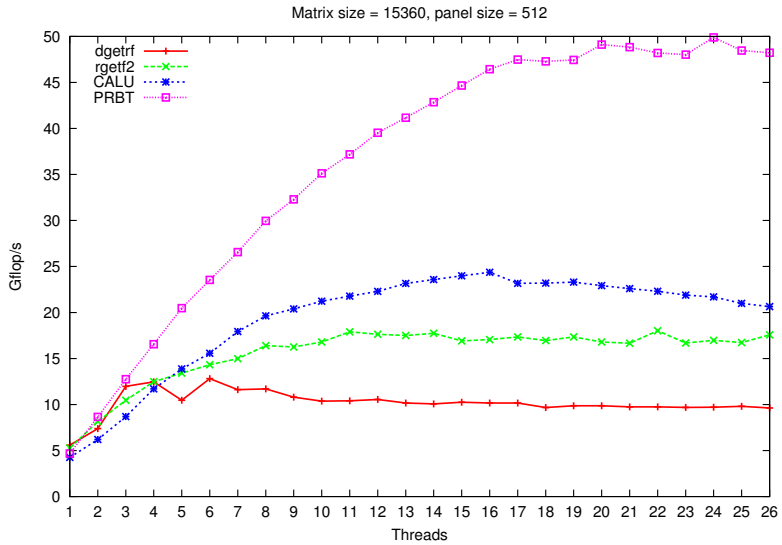
Performance results

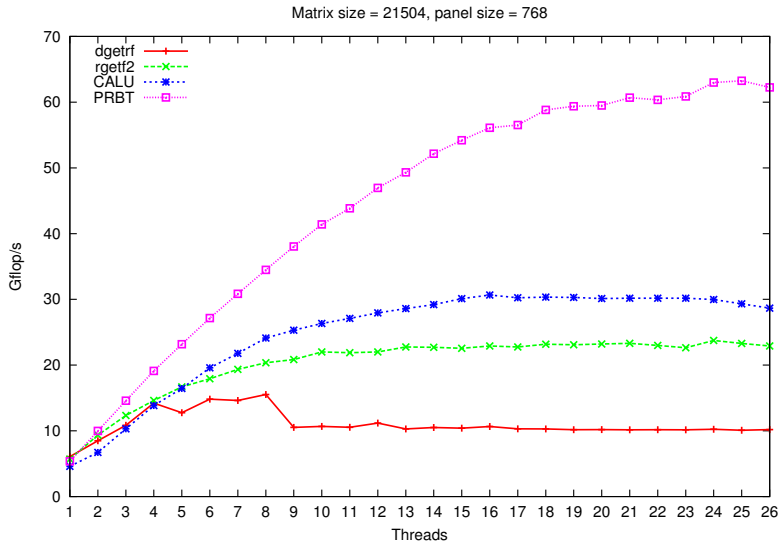
- Hybrid CPU/GPU algorithms implemented following MAGMA development guidelines
- GPU device: NVIDIA Fermi Tesla S2050 (448 CUDA cores)
  Multicore host: 4$\times$12-Core AMD Opteron 6172 Magny-Cours @ 2.1 GHz, 128GB memory, theroretical peak 403.2 Gflop/s (8.4 Gflop/s per core) in double precision
- **Panel factorization:** comparisons against MKL multithreaded
- **Hybrid LU solvers:** We compare MAGMA, PRBT and H-CALU

Comparison of CPU multi-threaded panel factorizations

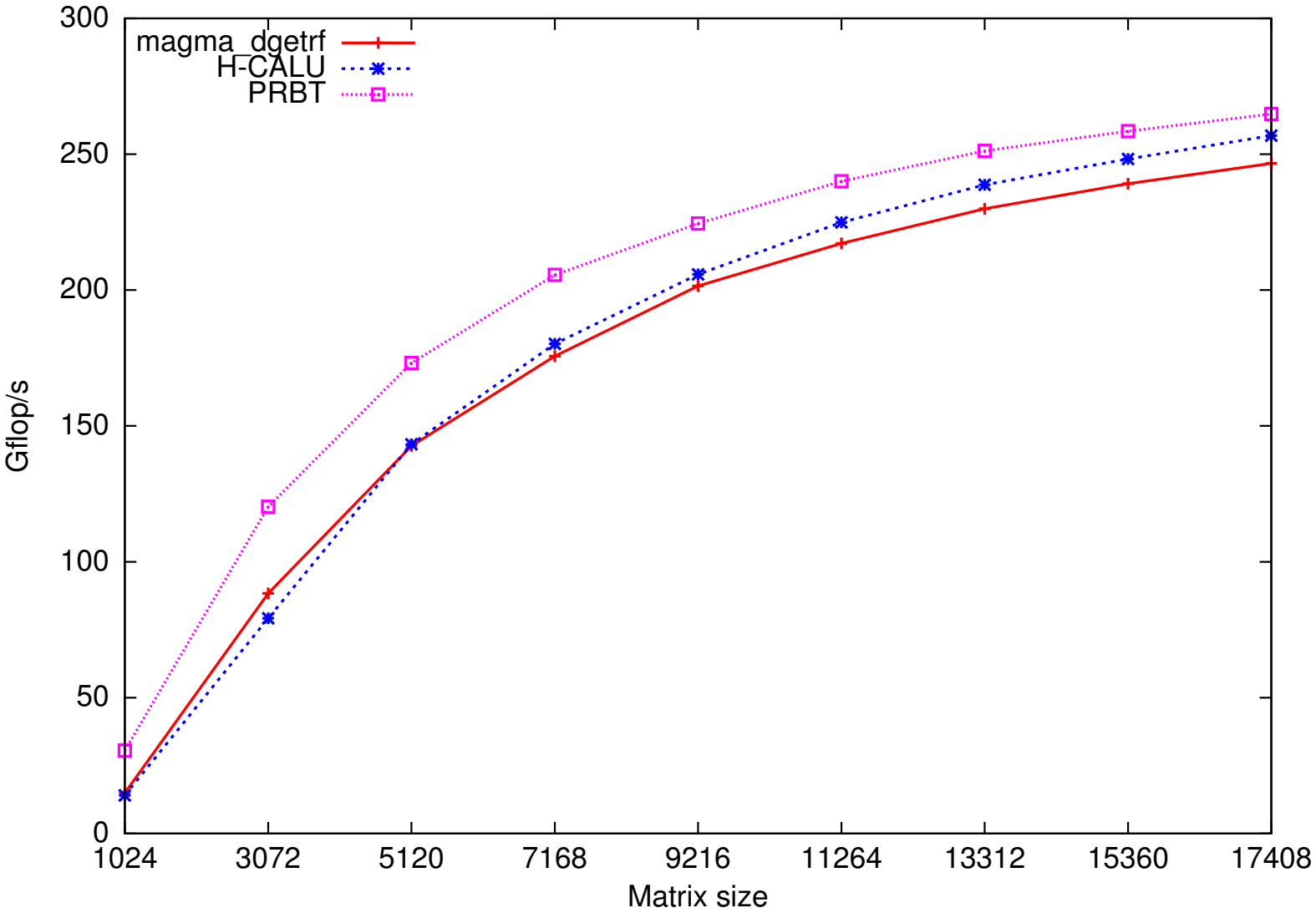








Performance on square matrices



Tests on accuracy

- We compare 3 solvers:
  - MAGMA/GEPP
  - H-CALU
  - PRBT (2 recursions)
- 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	7	Last n/2 columns zero
2	Upper triangular	8	Random, $\kappa = \sqrt{0.1/\varepsilon}$
3	Lower triangular	9	Random, $\kappa = 0.1/\varepsilon$
4	Random, $\kappa = 2$	10	Scaled near underflow
5	First column zero	11	Scaled near overflow
6	Last column zero		

Accuracy Comparison

Table: Componentwise Backward Error

Matrix Type	MAGMA LU (`magma_dgetrf`)	H-CALU	PRBT	No pivoting
1	0.0	0.0	1.42e-16(1)	0.0
2	1.32e-16	1.32e-16	4.02e-16(3)	6.19e-16
3	1.85e-16	1.85e-16	2.46e-16(3)	2.14e-16
4	2.16e-16	2.76e-16	2.93e-16(2)	1.13e-11
5	-	-	-	-
6	-	-	-	-
7	-	-	-	-
8	2.10e-16	3.76e-16	2.64e-16(3)	2.94e-12
9	2.70e-16	6.37e-16	1.16e-13(1)	1.41e-13
10	7.60e-14	7.40e-14	4.01e-14(2)	2.42e-11
11	2.27e-16	2.11e-16	2.41e-16(2)	2.90e-11

Multi GPU implementation

Original matrix is distributed among the GPUs (1D block cyclic)

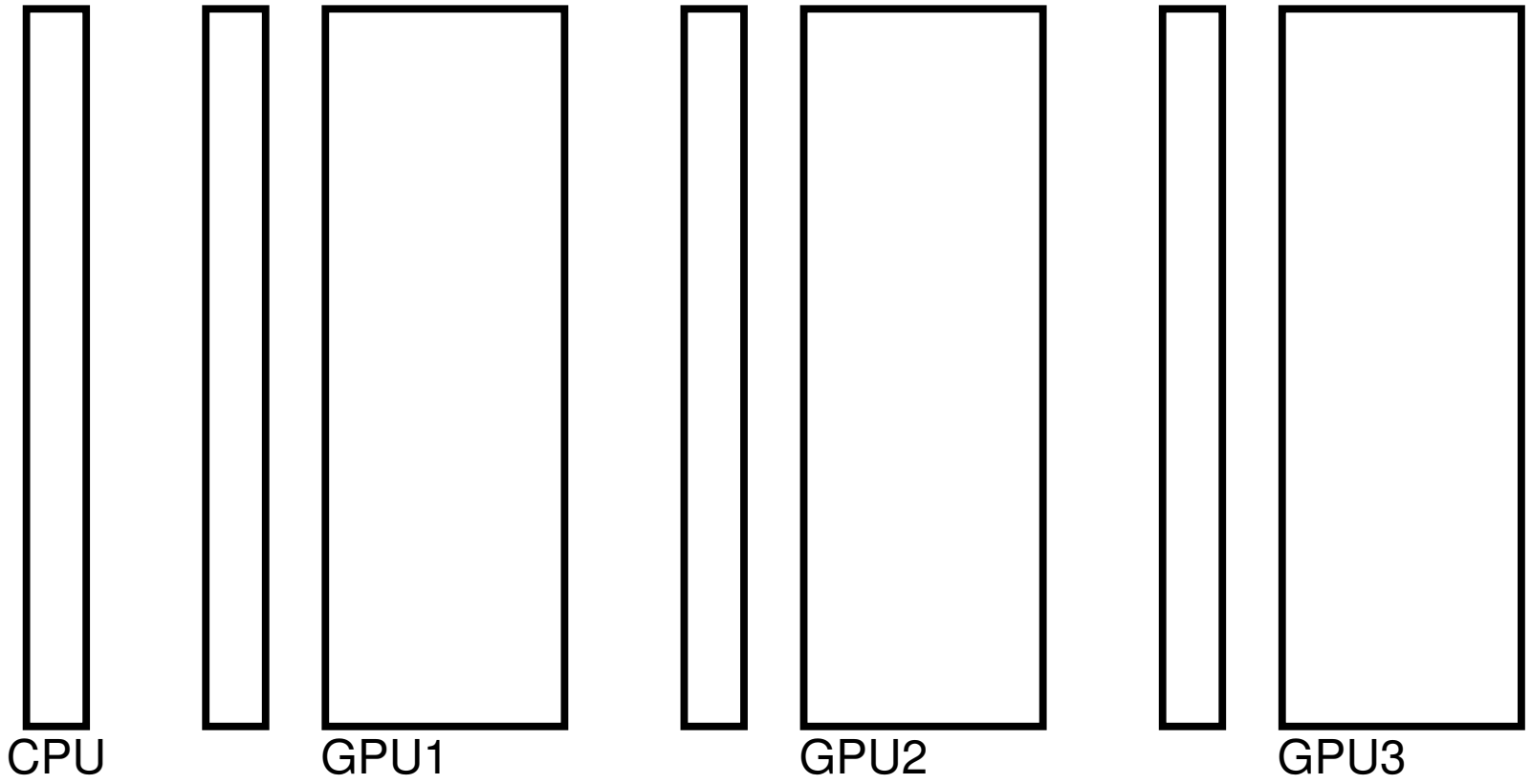


Figure: LU factorization using 3 GPUs

Multi GPU implementation

First panel is sent to the CPU

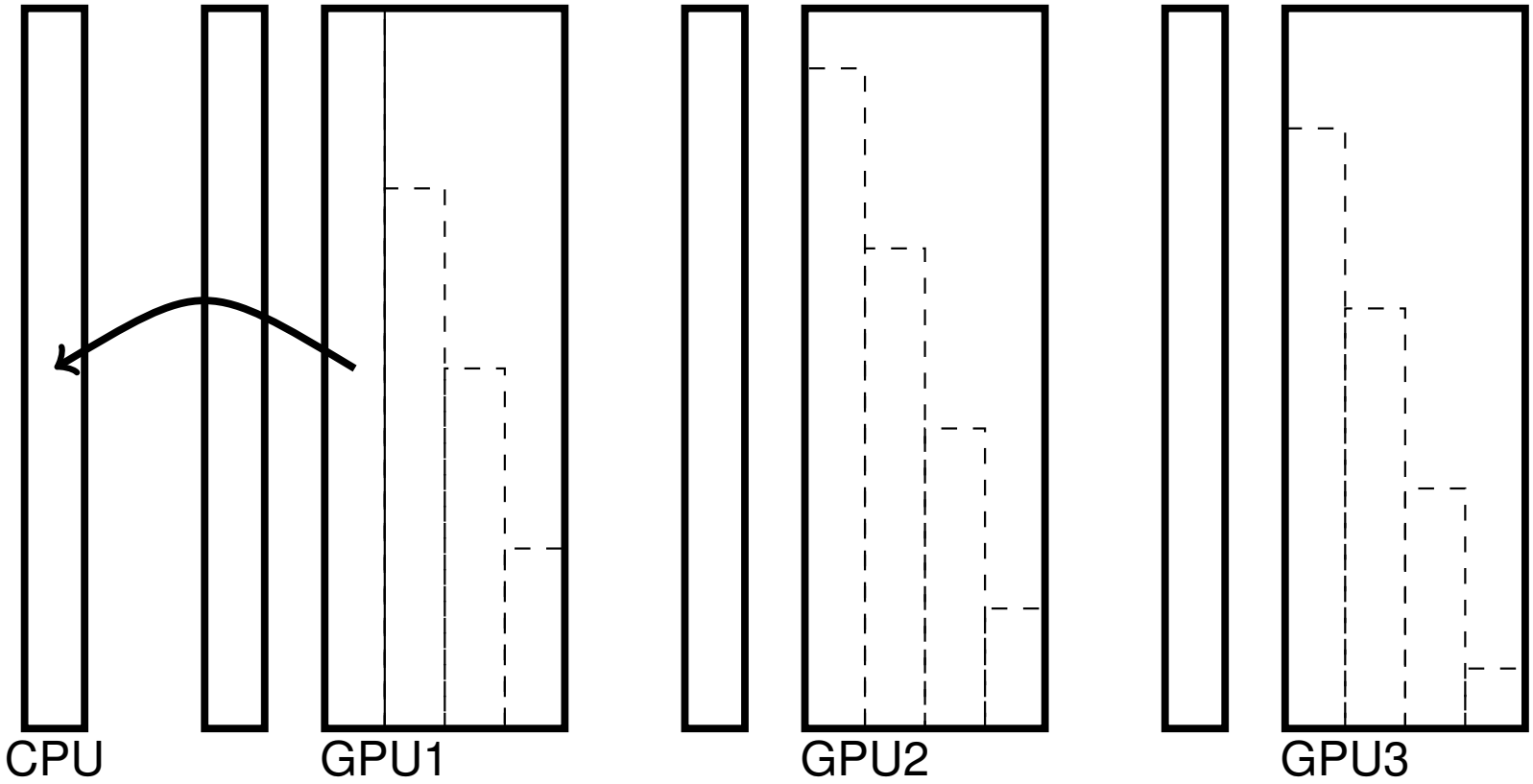


Figure: LU factorization using 3 GPUs

Multi GPU implementation

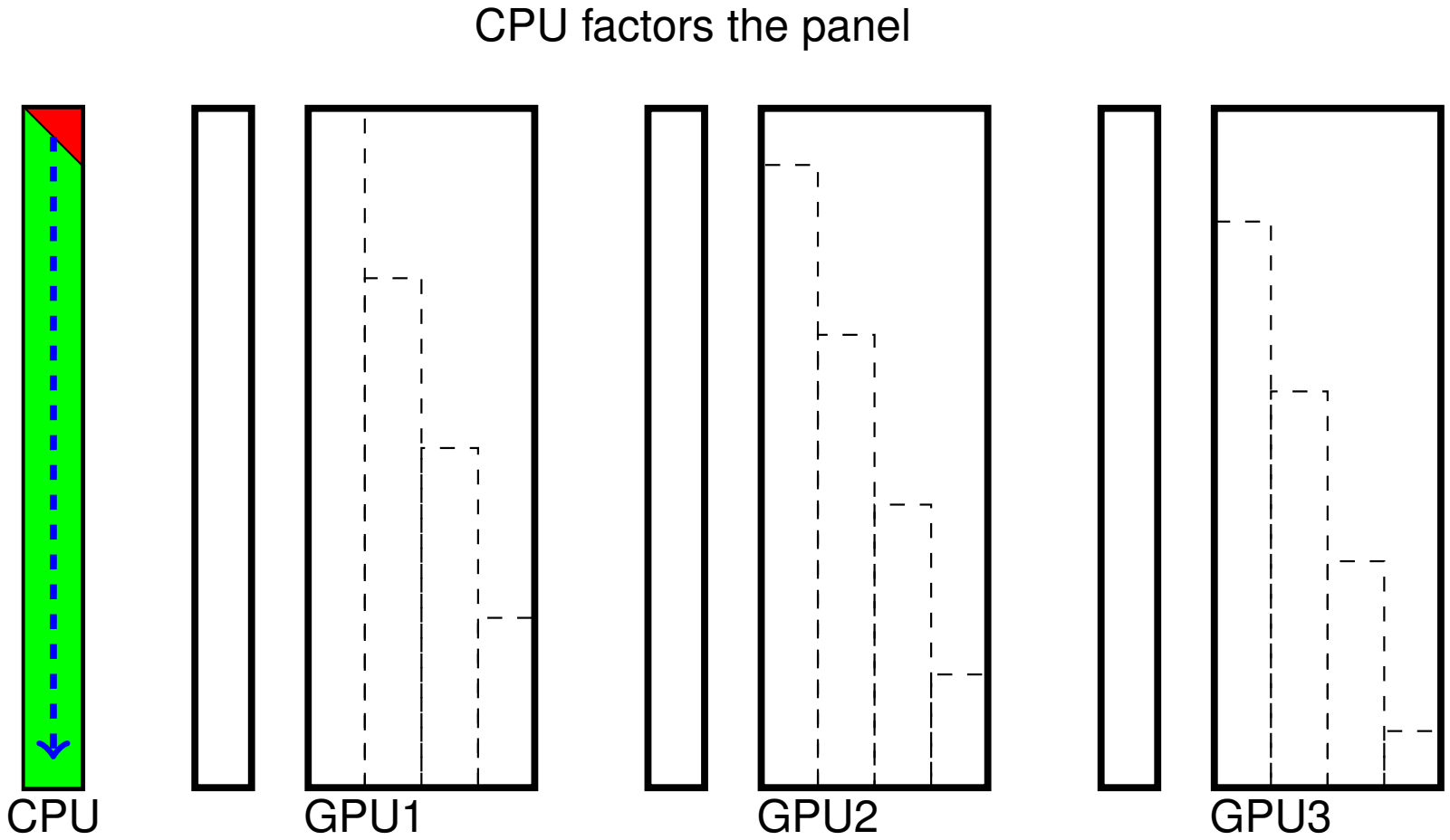


Figure: LU factorization using 3 GPUs

Multi GPU implementation

Factored panel is sent to the GPUs

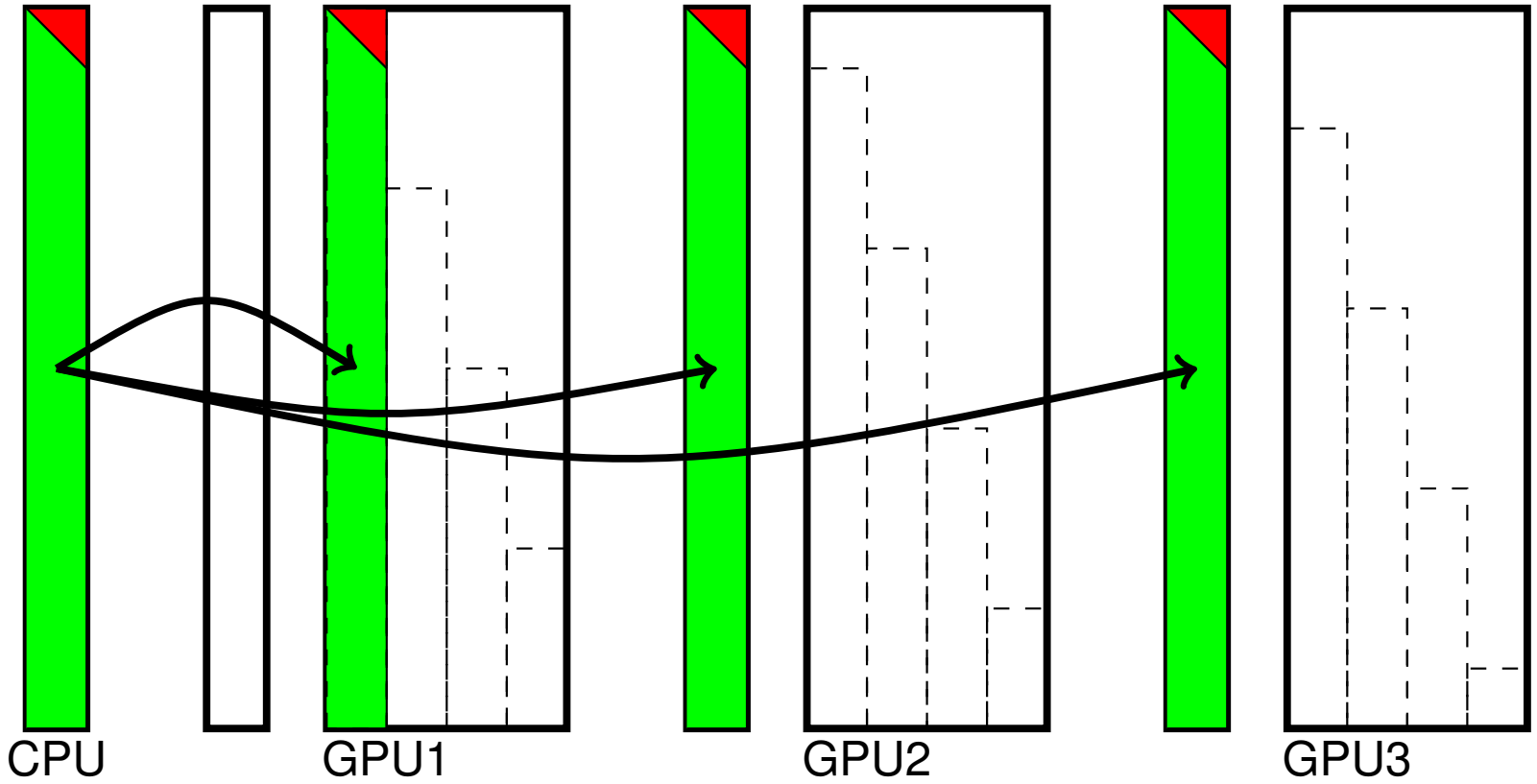


Figure: LU factorization using 3 GPUs

Multi GPU implementation

GPUs update trailing submatrices

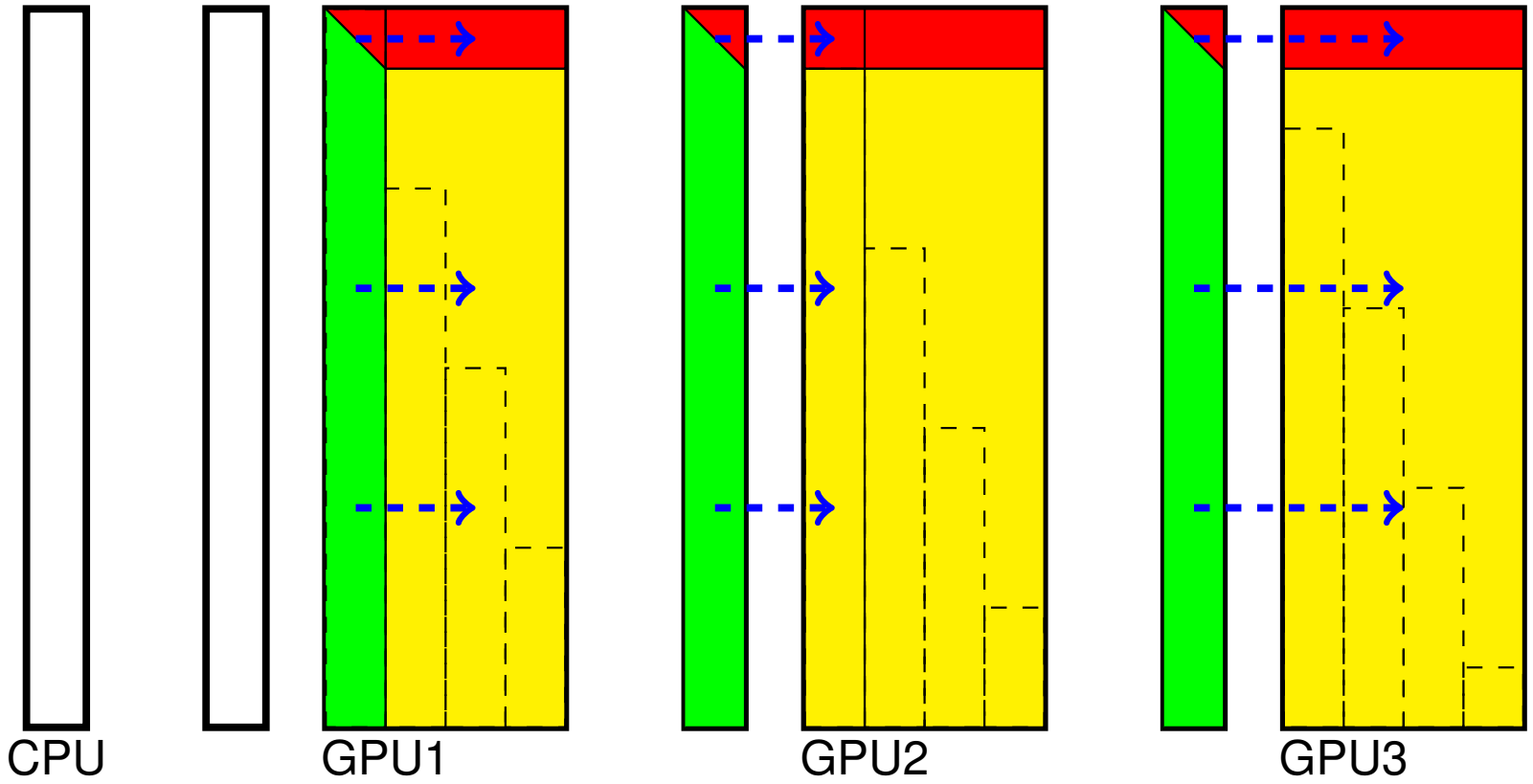


Figure: LU factorization using 3 GPUs

Multi GPU implementation

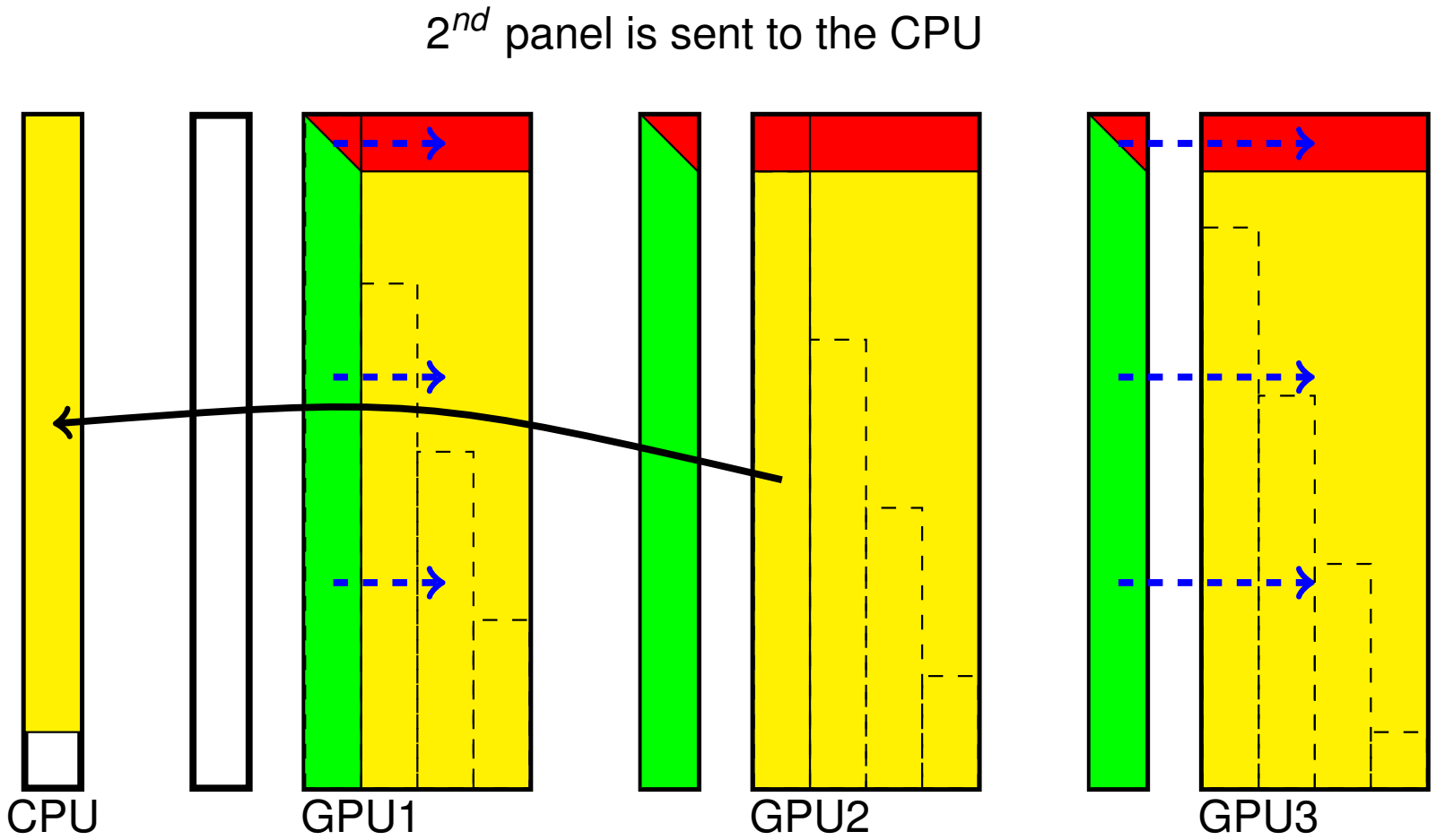


Figure: LU factorization using 3 GPUs

Multi GPU implementation

CPU factors new the panel while GPUs still update trailing submatrices

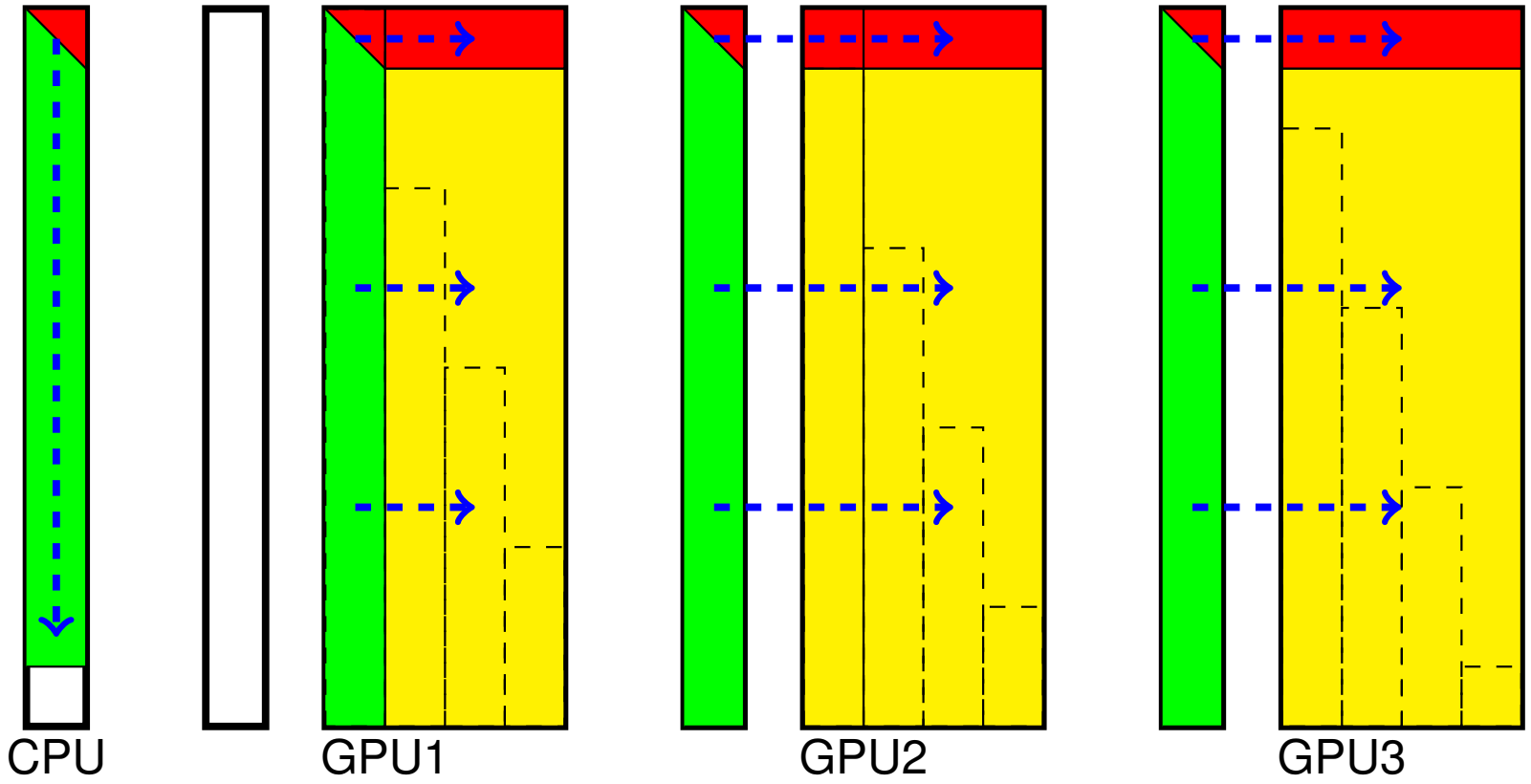
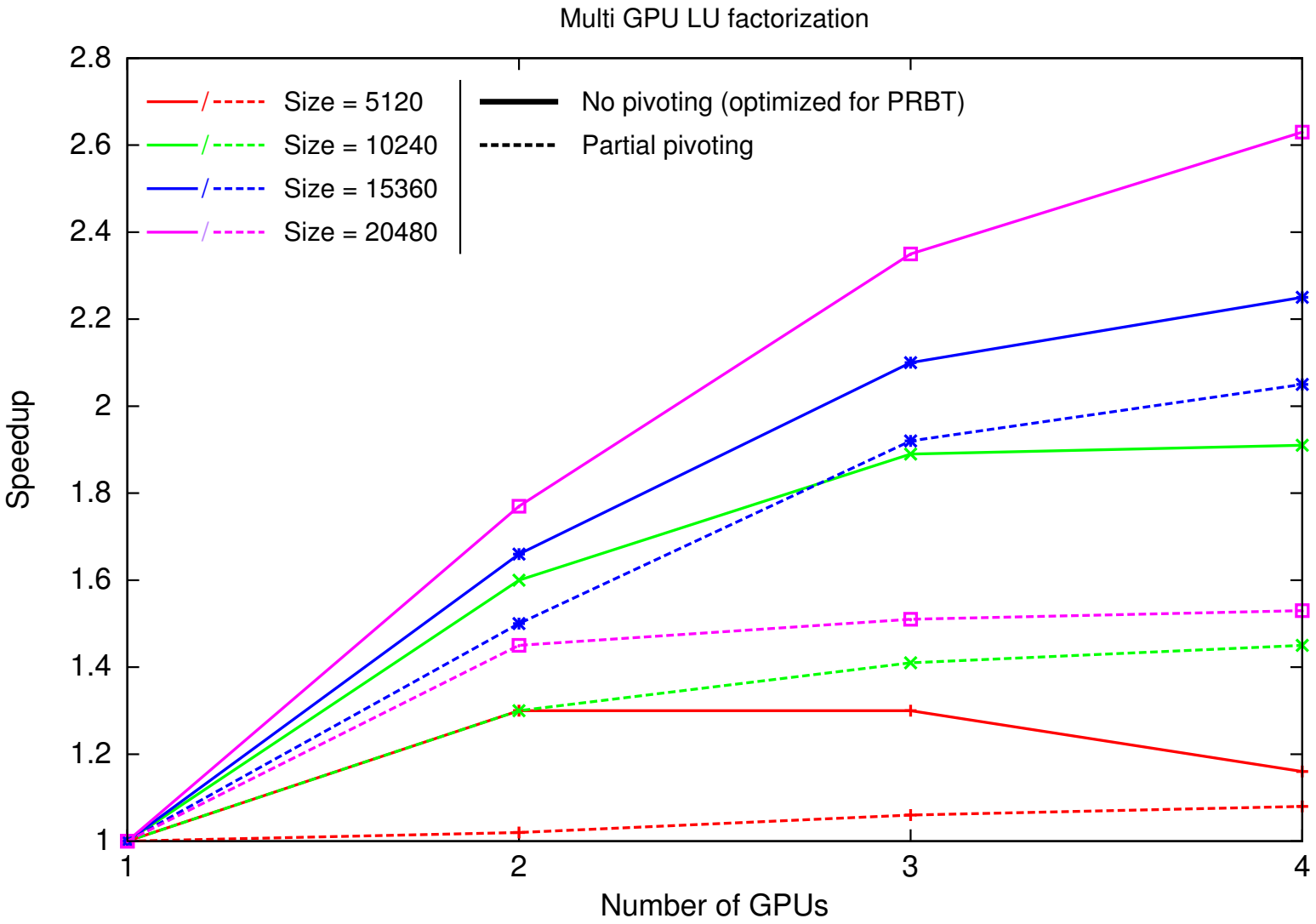


Figure: LU factorization using 3 GPUs

Performance results



Going to a bigger scale

- Using clusters of GPUs
- Distributed memory version of LU factorization with multiple GPUs **in progress**
- Using GMAC for managing communication

What is GMAC ?

- GMAC [I. Gelado et al. ASPLOS'10](Global Memory for ACcelerators) provides unified virtual address for CUDA
- Simplify the CPU code
- Single virtual address space for CPUs and GPUs
- Provide advanced CUDA features for free :
  - Asynchronous data transfer
  - Pinned memory
  - GPU to GPU communication
  - Get access to any GPU from any CPU thread
- Collaboration with Wen-Mei Hwu (University of Illinois at Urbana-Champaign)

Summary

- Efficient and accurate solvers for hybrid architectures :
  - Solutions for multicore accelerated with **one GPU**
  - Solutions for multicore accelerated with **several GPUs**
  - Give similar accuracy results on most test cases
- Difference between the solvers comes from the pivoting strategy for factoring the panel
- Distributed version in progress to use clusters of GPUs

Related papers

[1] M. Baboulin, S. Donfack, J. Dongarra, L. Grigori, A. Rémy, S. Tomov,
A class of algorithms for solving general dense linear systems on CPU/GPU parallel machines.
Proceedings of ICCS 2012.

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

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

[4] I. Gelado, J. Cabezas, N. Navarro, J. E. Stone, S. Patel, W. Hwu,
An asymmetric distributed shared memory model for heterogeneous parallel systems.
Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010) . Pittsburgh, USA. March 2010.

[5] 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).

[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).