Dense linear algebra for hybrid GPU-multicore systems. Application to linear systems.

Marc Baboulin

Universidade de Coimbra (Portugal)

joint work with
Jack Dongarra (University of Tennessee and Oak Ridge National Laboratory)
and **Stanimire Tomov** (University of Tennessee)

General framework

How to speed up numerical simulations ?

- Exploit advances in hardware (e.g multicore, GPUs, FPGAs, Cell),
  manage to use hardware efficiently for HPC applications
- Better numerical methods

Impact on numerical libraries

- Dense Linear Algebra (DLA) calculations
- LAPACK, ScaLAPACK, sparse solvers, iterative solvers...have to be rethought and rewritten

Outline

Outline

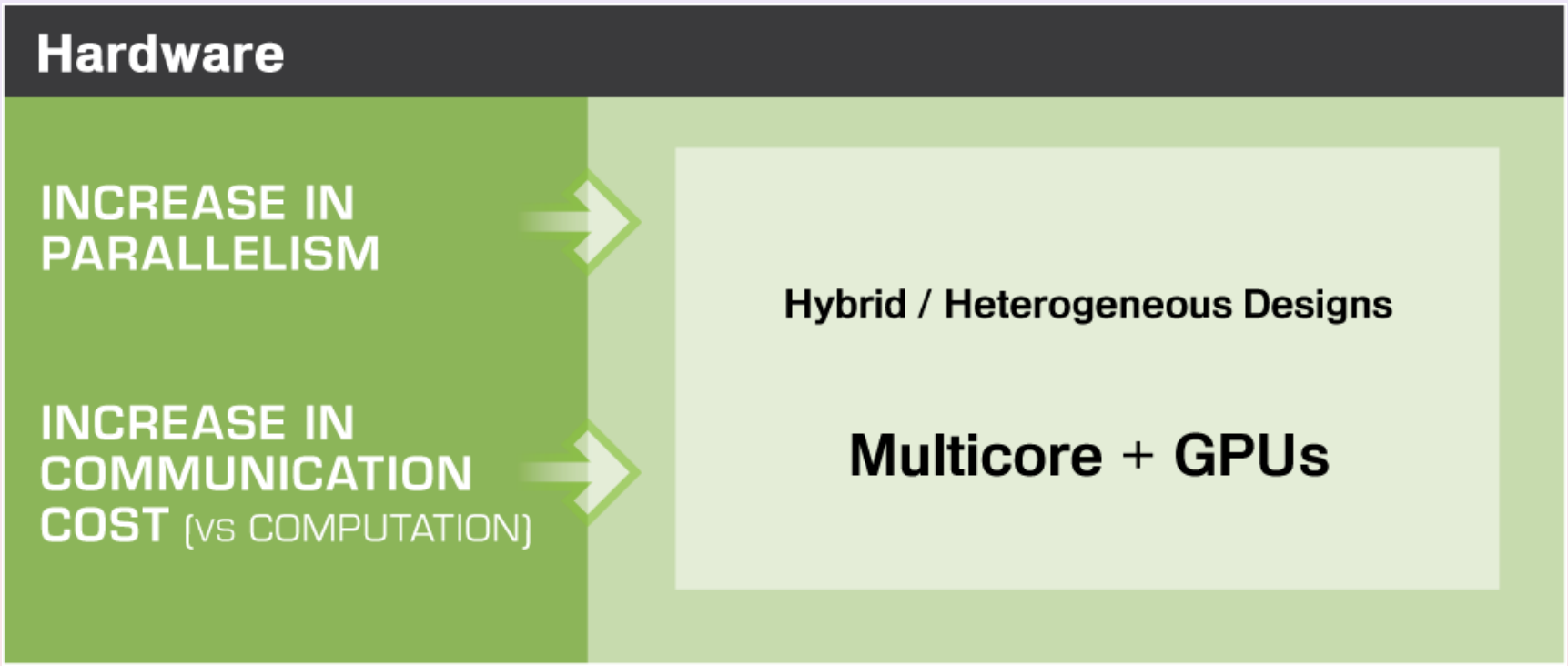
1. Taking advantage of new parallel architectures
   - Towards hybrid GPU-multicore algorithms
   - Mixed precision algorithms
2. Getting faster through statistics
   - Randomization in linear systems
   - Accuracy and performance results

Outline

Outline

Outline

Hardware to software trends



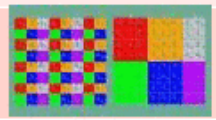
Processor speed improves 59% / year but memory bandwidth only by 23%, latency by 5.5%

GPUs evolution

- Used in applications far beyond graphics (GPGPU)
- High bandwidth
- Significantly outperform current multicores for linear algebra calculations (since 2008)
- More programmability (CUDA)
- Multiple levels of memory hierarchy
- Numerical libraries available (CUDA...)
- New generation supports double precision arithmetic

A New Generation of Software:

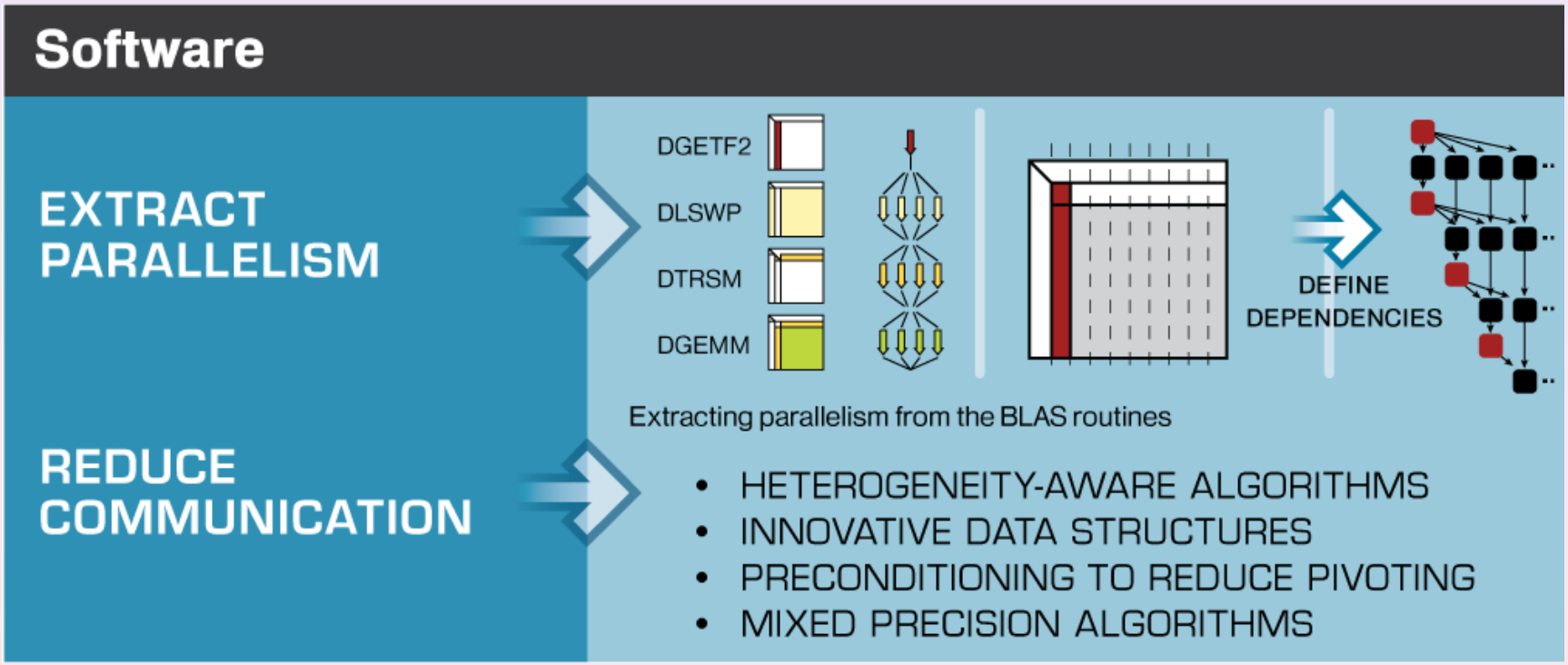
Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (00's) New Algorithms (many-core friendly)		Rely on - a DAG/scheduler - block data layout - some extra kernels

Those new algorithms
- have a very low granularity, they scale very well (multicore, petascale computing, ...)
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

Hardware to software trends



Heterogeneity-aware algorithms

- Architecture trends have moved towards heterogeneous (CPU+GPU) designs
- Objective: fully exploit the power that each of the hybrid components offers
- There are significant differences between the new algorithms and those for conventional CPUs
- Need for linear algebra routines for hybrid systems: there is no self contained library like LAPACK

MAGMA project

- **MAGMA**: **Matrix Algebra on GPU and Multicore Architectures**
- DLA library for heterogeneous/hybrid architectures starting with current Multicore+GPU systems
  aims to fastest possible time to an accurate solution
- Similar to LAPACK in functionality and interface
- Institutions: U. Tennessee, U. California Berkeley, U. Coimbra, U. Colorado Denver, INRIA
- Funding: DOE, NSF, FCT
- Industrial support: Microsoft, NVIDIA, MathWorks
- Other projects: FLAME (UT Austin), ViennaCL

MAGMA project

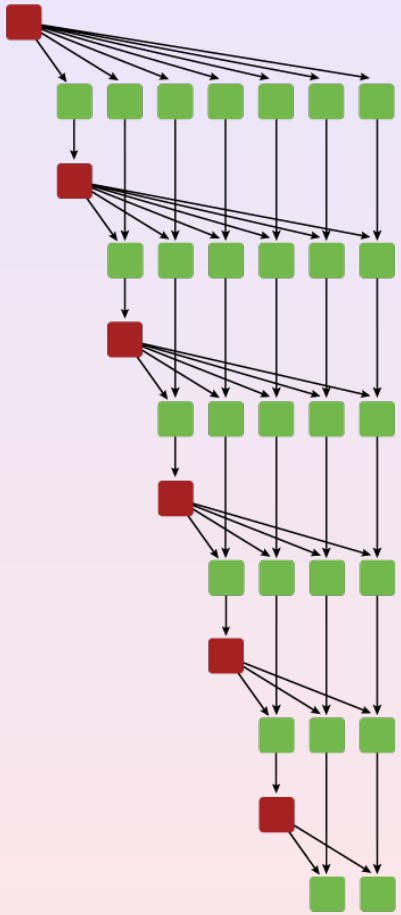
- **MAGMA version 0.2**
  `http://icl.cs.utk.edu/magma/`
- Single GPU, one-sided matrix factorizations and solvers, including mixed-precision iterative refinement solvers
- Precision: single, double, single complex, double complex
- LAPACK-style interfaces (prefixed by *magma*)
- MAGMA BLAS : complementary to CUBLAS
- Work in progress: two-sided factorizations, eigensolvers, multiple GPUs
- MAGMA users do not have to know CUDA in order to use the library

GPUs for linear algebra calculations

Design philosophy:

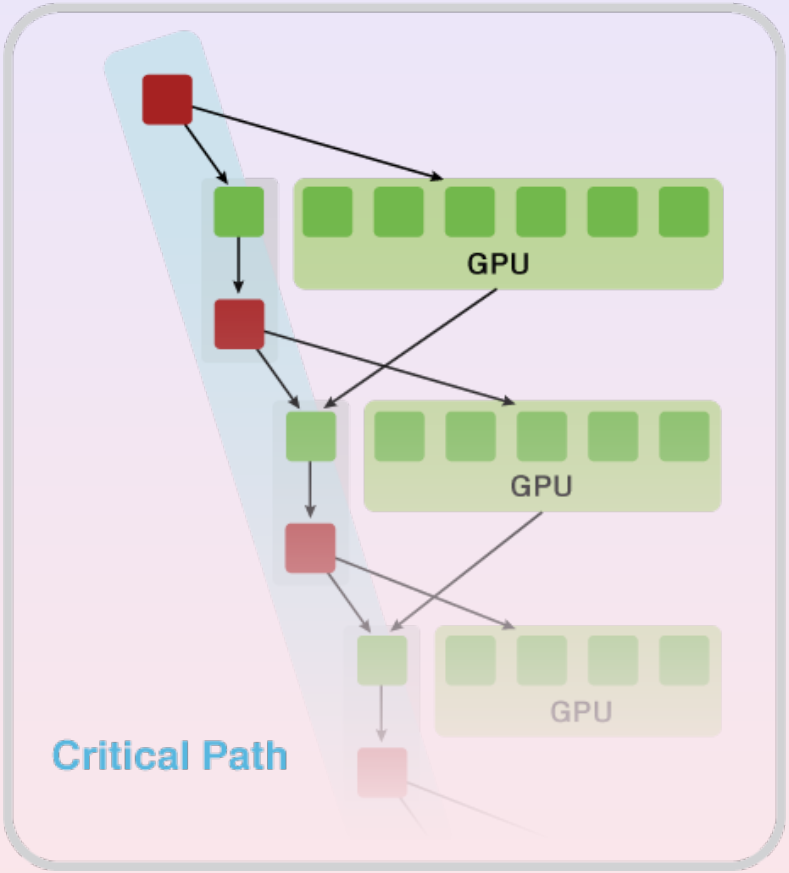
- Represent LAPACK algorithms as a collection of BLAS-based tasks and dependencies among them
  $\longrightarrow$ rely on high performance of BLAS implementations for current multicore and GPUs
- Reuse parts of LAPACK in a systematic way
- Abstract us from specificities in programming a GPU
- Properly schedule the tasks execution over the multicore and the GPU

Task splitting and scheduling



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

Task splitting and scheduling



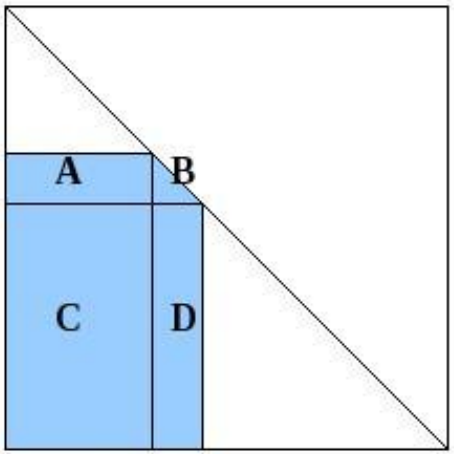
DAGs for hybrid systems
(both small and large tasks)

GPUs for linear algebra calculations

Principles of hybrid implementation:

- BLAS-level parallelism where the matrix resides on the GPU (BLAS calls replaced by CUBLAS)
- Offload to the CPU small kernels that are inefficient for the GPU
- Use asynchronism between CPU and GPU whenever possible

Example: Cholesky factorization



(1) $B = B - A A^T$ ssyrk (GPU)
(2) $B = LL^T$ spotrf (CPU)
(3) $D = D - CA^T$ sgemm (GPU)
(4) $D = D \backslash B$ strsm (GPU)

Hybrid implementation:
- a_ref points to the GPU memory
- GPU kernels are started asynchronously which results in overlapping the GPU's sgemm with CPU's spotrf

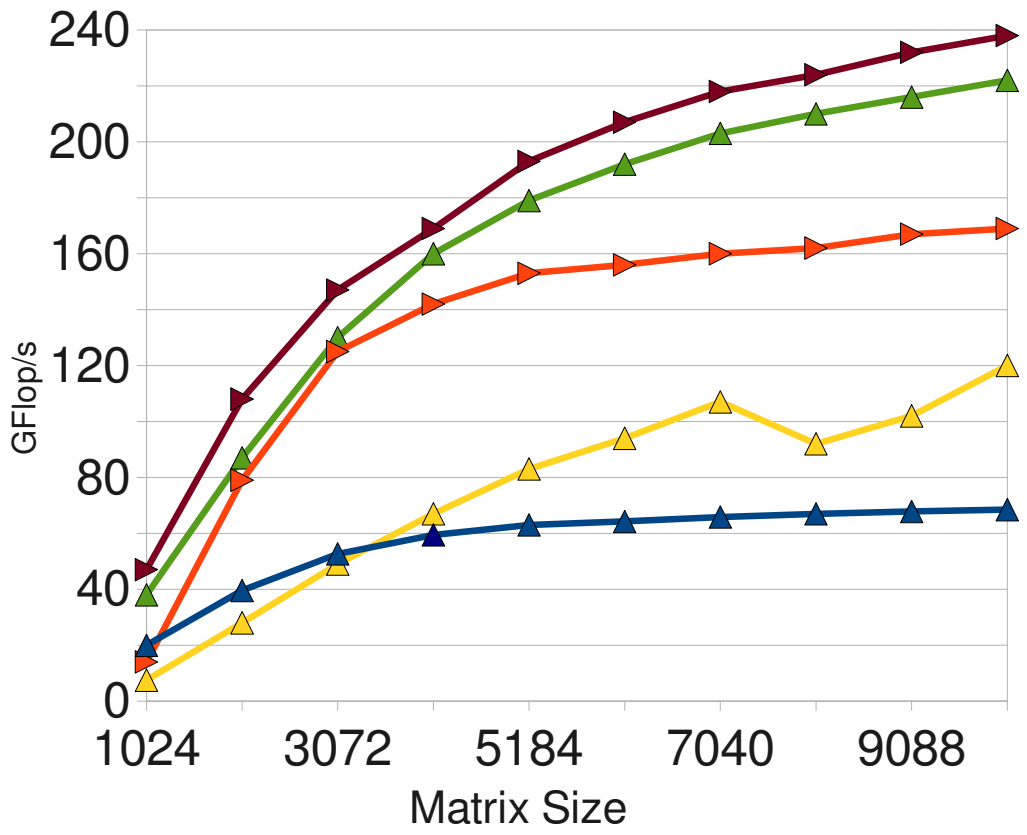
Standard LAPACK pseudo-code

```
ssyrk_("L", "N", &jb, &i__3, &c_b13, a_ref(j,1), ... )

spotrf_("L", &jb, a_ref(j, j), lda, info)
sgemm_("N", "T", &i__3, ... )

strsm_("R", "L", "T", "N", &i__3, ... )
```

Hybrid Single Core-GPU code

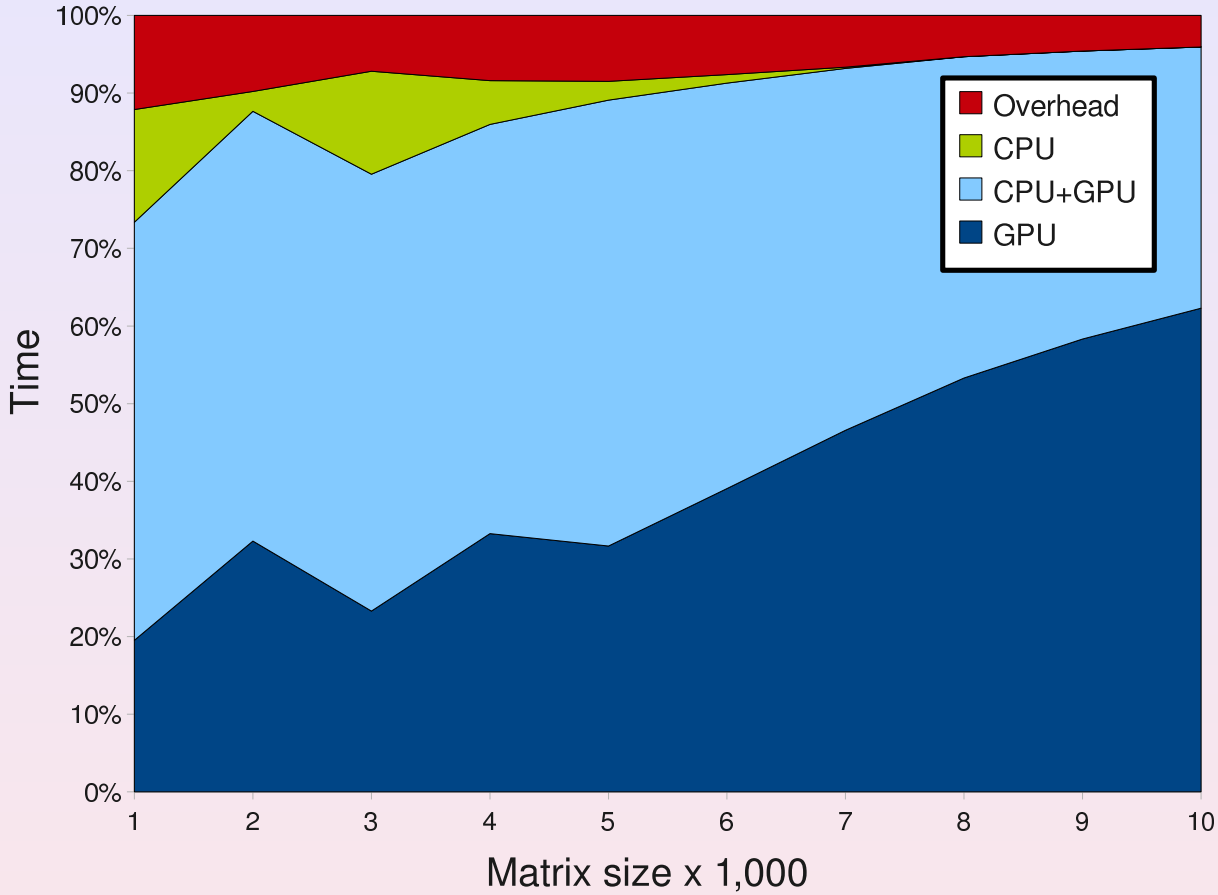
```
cublasSsyrk('L', 'N', jb, i__3. c_b13, a_ref(j,1), ... )

cublasGetMatrix(jb, jb, 4, a_ref(j, j), *lda, work, jb)
cublasSgemm('N', 'T', i__3, ... )
spotrf_("L", &jb, work, &jb, info)
cublasSetMatrix(jb, jb, 4, work, jb, a_ref(j, j), *lda)

cublasStrsm('R', 'L', 'T', 'N', i__3, ... )
```

LU factorization in double precision



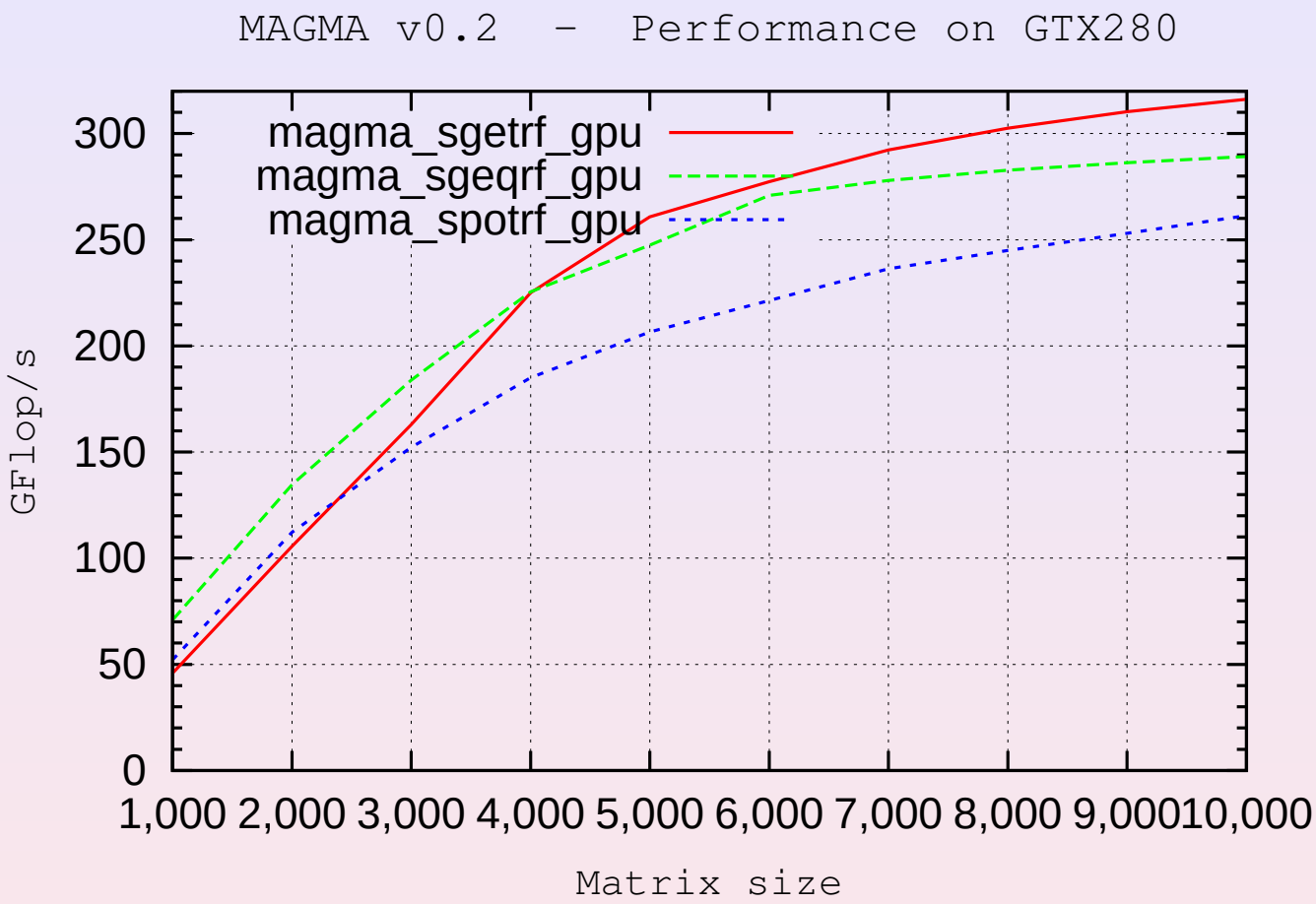
FERMI Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 GFlop/s

ISTANBUL AMD 8 socket 6 core (48 cores) @2.8GHz
SP/DP peak is 1075 / 538 GFlop/s



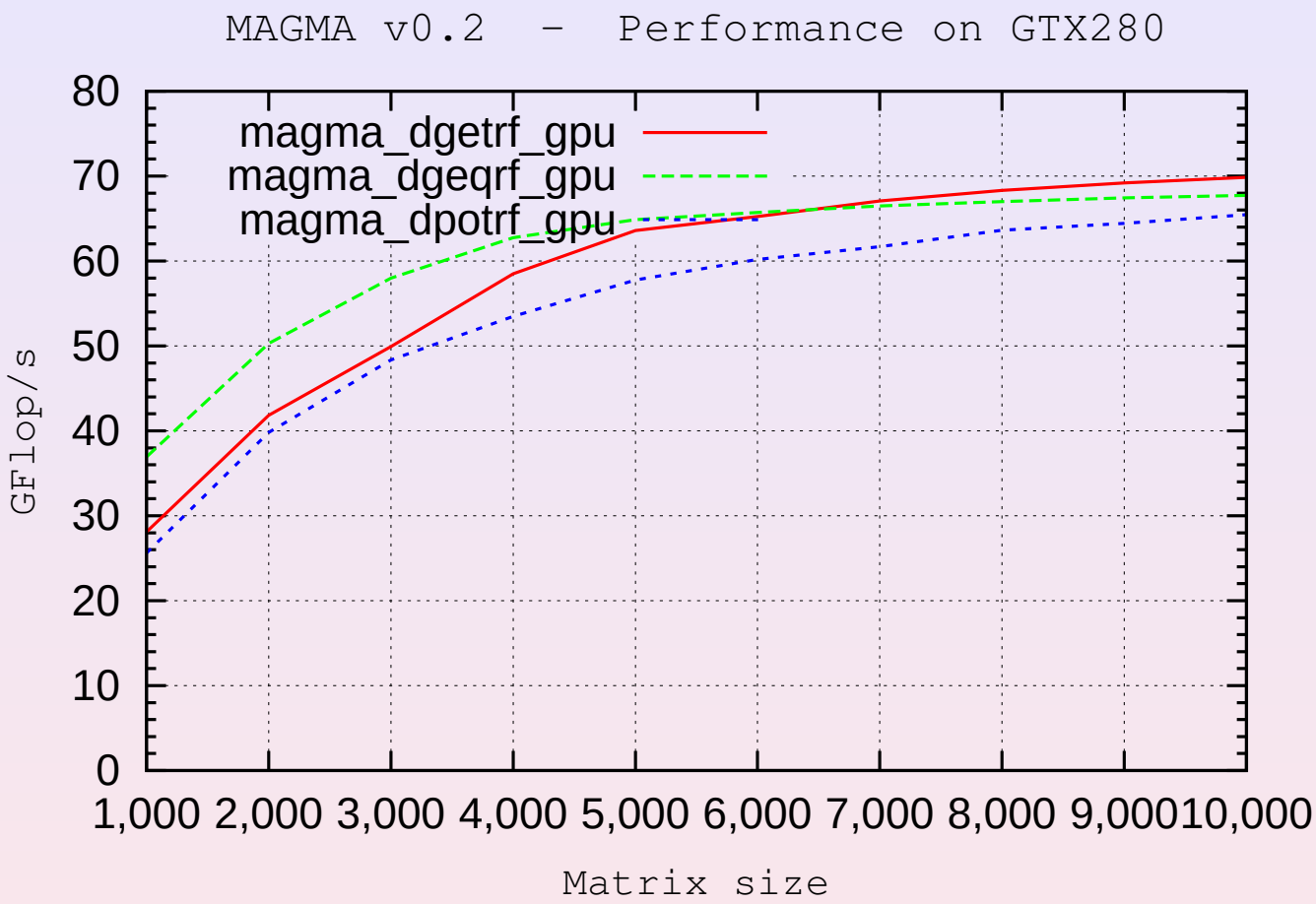
Time breakdown for MAGMA QR (single precision)

Intel Xeon (2 x 4 cores @ 2.33 GHz) - GeForce GTX 280 (240 Cores @ 1.30 GHz).



Performance of one-sided factorizations (single precision)

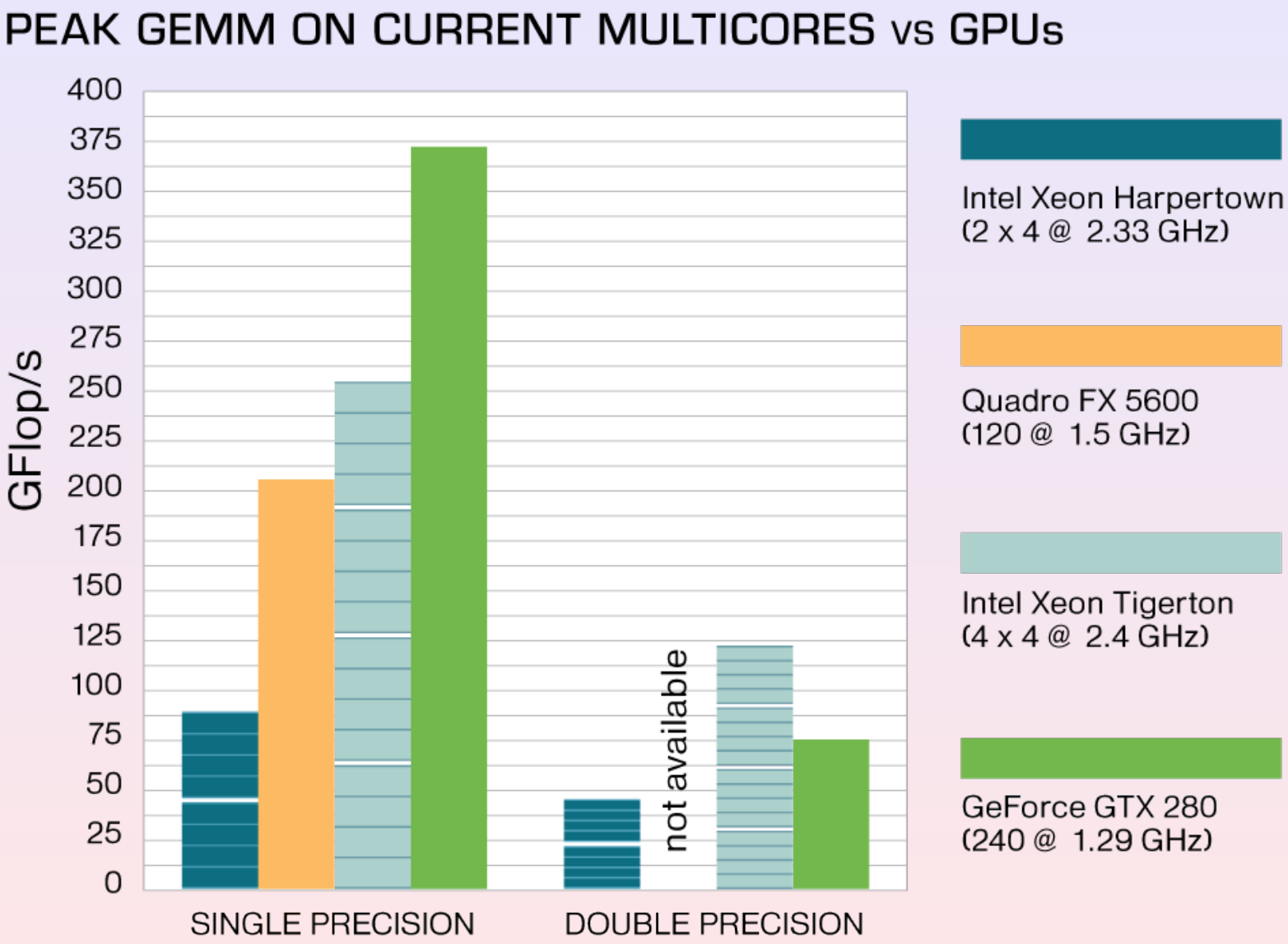
Intel Xeon (2 x 4 cores @ 2.33 GHz) - GeForce GTX 280 (240 Cores @ 1.30 GHz).



Performance of one-sided factorizations (double precision)

Intel Xeon (2 x 4 cores @ 2.33 GHz) - GeForce GTX 280 (240 Cores @ 1.30 GHz).

Outline


PEAK GEMM ON CURRENT MULTICORES vs GPUs
GFlop/s
400
375
350
325
300
275
250
225
200
175
150
125
100
75
50
25
0
SINGLE PRECISION
DOUBLE PRECISION
not available
Intel Xeon Harpertown
(2 x 4 @ 2.33 GHz)
Quadro FX 5600
(120 @ 1.5 GHz)
Intel Xeon Tigerton
(4 x 4 @ 2.4 GHz)
GeForce GTX 280
(240 @ 1.29 GHz)

Mixed precision algorithms

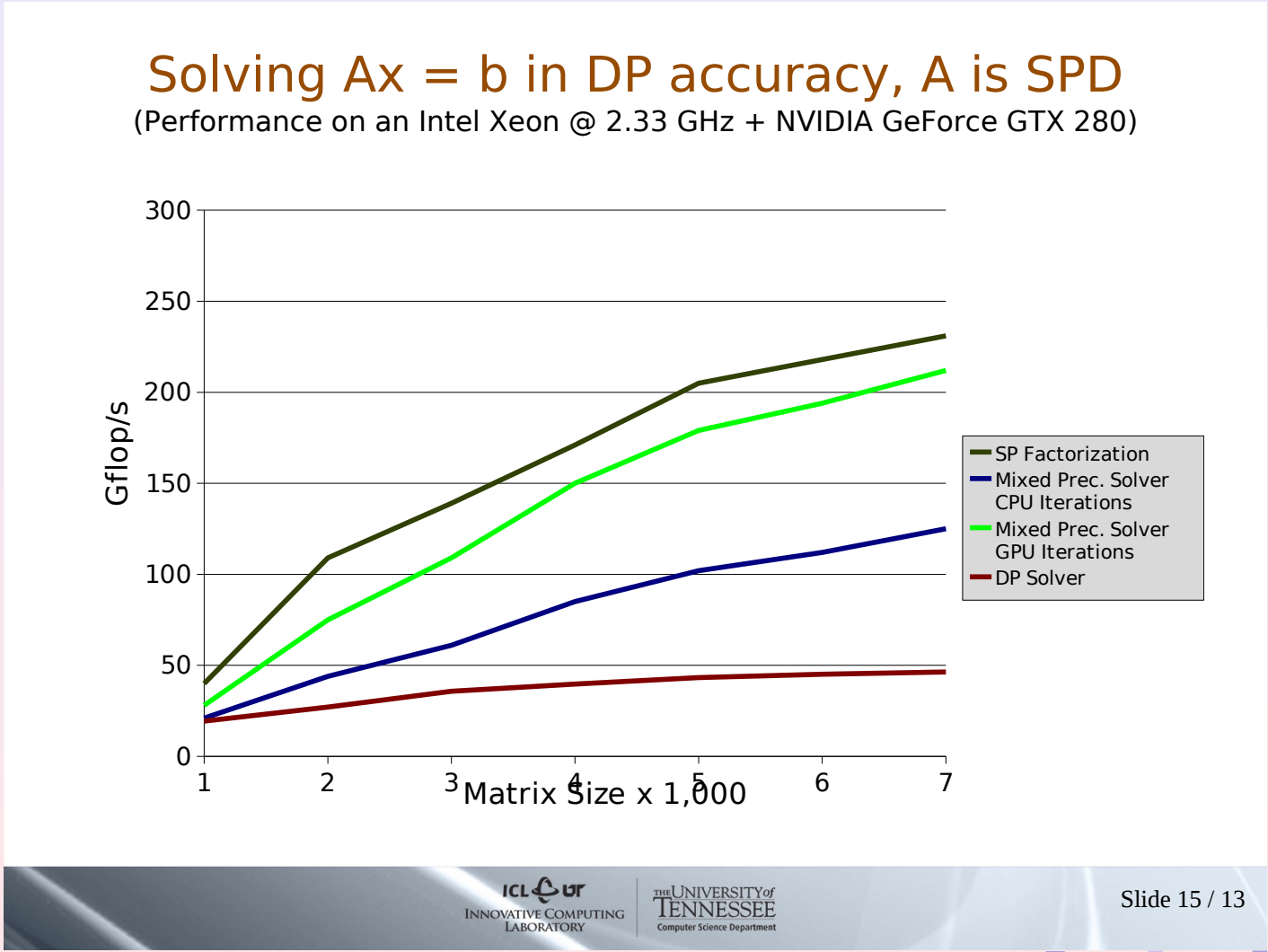
- Bulk of the computation in 32-bit arithmetic
- Postprocess the 32-bit solution by refining it into a solution that is 64-bit accurate
- Problem must be ”not ill-conditioned”
- Software details in:
  M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, S. Tomov,
  Accelerating scientific computations with mixed precision algorithms.
  Computer Physics Communications , Vol. 180, No 12, pp. 2526-2533 (2009).

Mixed precision algorithms

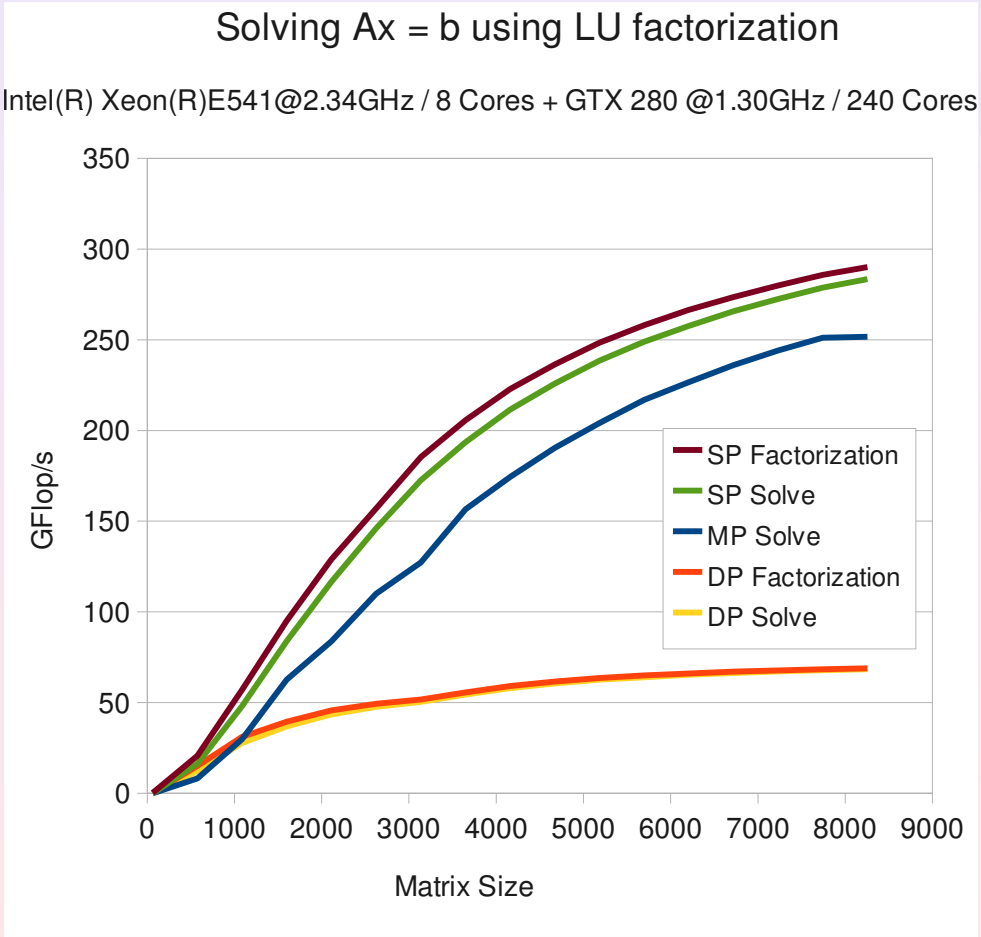
Example of the Cholesky factorization

1:	$LL^T \leftarrow A$	(ε_s)	$\mathcal{O}(n^3)$
2:	solve $Ly = b$	(ε_s)	$\mathcal{O}(n^2)$
3:	solve $L^T x_0 = y$	(ε_s)	$\mathcal{O}(n^2)$
	do $k = 1, 2, \ldots$		
4:	$r_k \leftarrow b - Ax_{k-1}$	(ε_d)	
5:	solve $Ly = r_k$	(ε_s)	
6:	solve $L^T z_k = y$	(ε_s)	
7:	$x_k \leftarrow x_{k-1} + z_k$	(ε_d)	
	check convergence		
	done		

Mixed precision Cholesky factorization



Linear Solvers



- Direct solvers
  - Factor and do triangular solves in the same, working precision
- Mixed Precision Iterative Refinement
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g. $x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$

Outline

Outline

The issue of pivoting in linear systems

- General square system $Ax = b$, solved by Gaussian Elimination (GE)
- We interchange rows: partial pivoting (PP) $\rightarrow$ stability
- Factorization $PA = LU$, where P permutation matrix
- Partial pivoting implemented in LAPACK, matlab...
- No floating point operation in pivoting but it involves irregular movement of data
- Communication overhead due to pivoting: $\mathcal{O}(n^2)$ comparisons, for some architectures (multicore, GPUs), up to 50% of the global computational time

Other approaches

- **Communication avoiding algorithms:**

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

  J. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, **Communication-optimal parallel and sequential QR and LU factorizations**, *In review, SISC*.

  Minimize the number of messages exchanged during the panel factorization, stable in practice.

- **GPU algorithms**:

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

  Reduce the pivoting overhead from 56% to 1-10% by using innovative data structure.

Random butterfly transformation (RBT)

- [Parker,95] proposed to make the matrix sufficiently "random" so that, with probability close to 1, pivoting is not needed
- **Precondition A with random matrices:** UAV
  to solve $Ax = b$, we instead solve $(UAV)y = Ub$ followed by $x = Vy$
- Random matrices U and V are chosen among a particular class of matrices called "butterfly matrices" which are of the form $\begin{pmatrix} P & Q \\ R & S \end{pmatrix}$. where P, Q, R and S are diagonal $n/2 \times n/2$ matrices.

Random butterfly transformation (RBT)

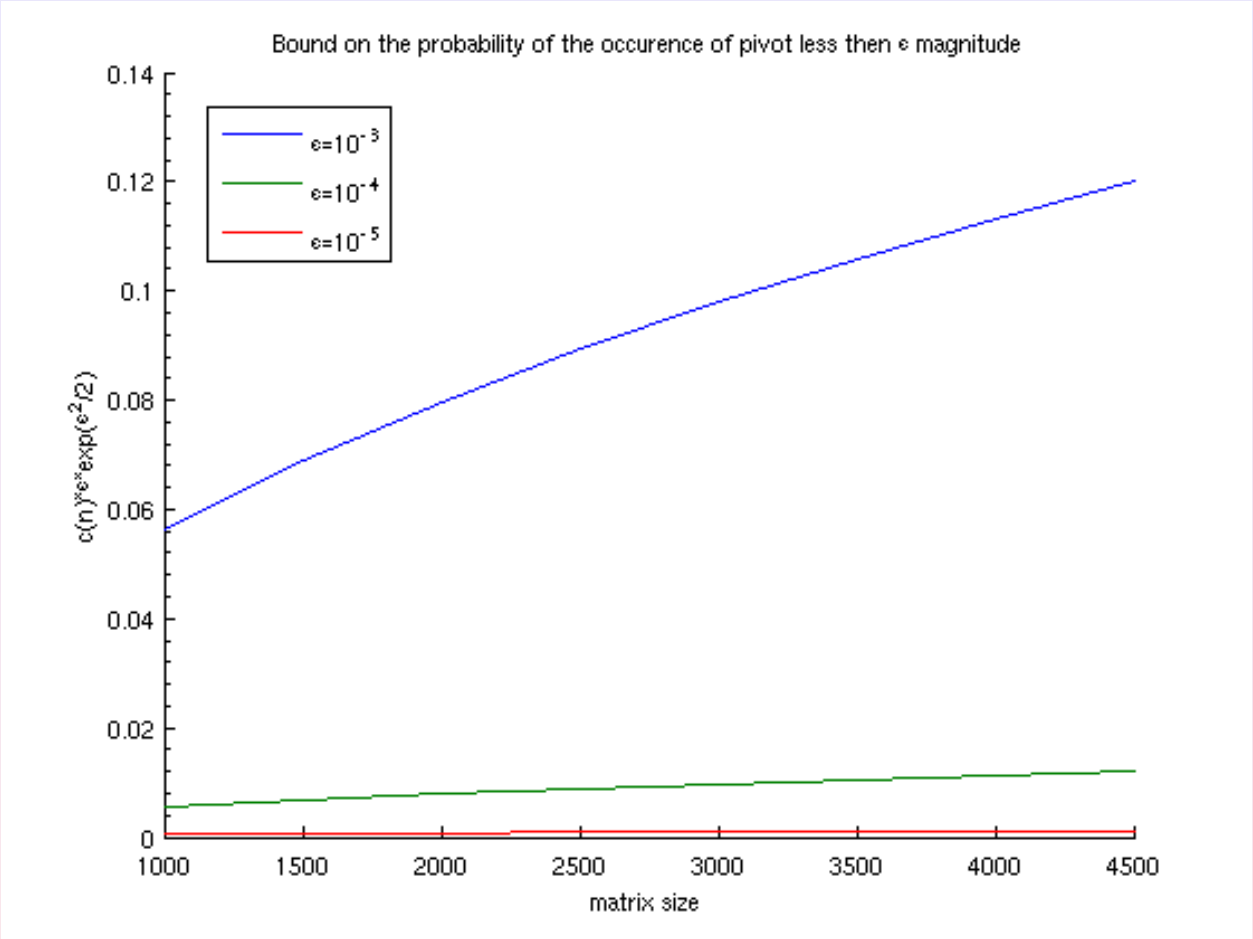
- Method: LU with no pivoting on a preconditioned matrix
- The preconditioning is "cheap" ($\mathcal{O}(n^2)$ operations)
- We do not pivot (RBT NP) or just within the first few rows of the panel (RBT LP)
  $\longrightarrow$ we have a fully BLAS 3 algorithm
- RBT may require some steps of iterative refinement in the working precision
- We take advantage of the GPU for all these calculations (preconditioning, factorization in SP, iterative refinement)
- More details in [Baboulin, Dongarra, Tomov, 2008]

Outline

Experiments with RBT LU

- we apply RBT to A and then factorize using an algorithm for GE (without pivoting) to UAV,
- matrices of size 1024 from Higham's collection
- we report $\omega = \max_i \frac{|r_i|}{(|A|\cdot|\bar{x}|+|b|)_i}$ (iterative refinement if $\omega > (n+1)u$)

Matrix	chebspec	circul	condex	fiedler	orthog	gfpp
Cond	$6 \cdot 10^{14}$	$5 \cdot 10^{2}$	$1 \cdot 10^{2}$	$2 \cdot 10^{5}$	$1 \cdot 10^{0}$	$2 \cdot 10^{2}$
GEPP # iter	$5 \cdot 10^{-16}$ 0	$1 \cdot 10^{-15}$ 0	$2 \cdot 10^{-15}$ 0	$2 \cdot 10^{-15}$ 0	$2 \cdot 10^{-15}$ 0	$2 \cdot 10^{-2}$ 10
GE # iter	$5 \cdot 10^{-16}$ 0	$1 \cdot 10^{-15}$ 1	$4 \cdot 10^{-15}$ 0	Fail –	Fail –	Fail –
QR # iter	$9 \cdot 10^{-16}$ 0	$2 \cdot 10^{-15}$ 0	$3 \cdot 10^{-15}$ 0	$6 \cdot 10^{-15}$ 0	$3 \cdot 10^{-16}$ 0	$1 \cdot 10^{-16}$
RBT+GE # iter	$6 \cdot 10^{-14}$ 3	$1 \cdot 10^{-15}$ 1	$4 \cdot 10^{-15}$ 1	$1 \cdot 10^{-15}$ 1	$4 \cdot 10^{-16}$ 2	$2 \cdot 10^{-16}$ 1



Bound on the probability of the occurrence of small pivots:

$Prob(\{|u_{pp}| < \epsilon;\ p = 1..n\}) \leq c(n)\epsilon e^{\frac{1}{2}\epsilon^2}$, where u_{pp} is the pivot at step p.

Backward error results for LU

If GE produces computed LU factors $\hat{L}$ and $\hat{U}$, and a computed solution $\hat{x}$ to $Ax = b$. Then

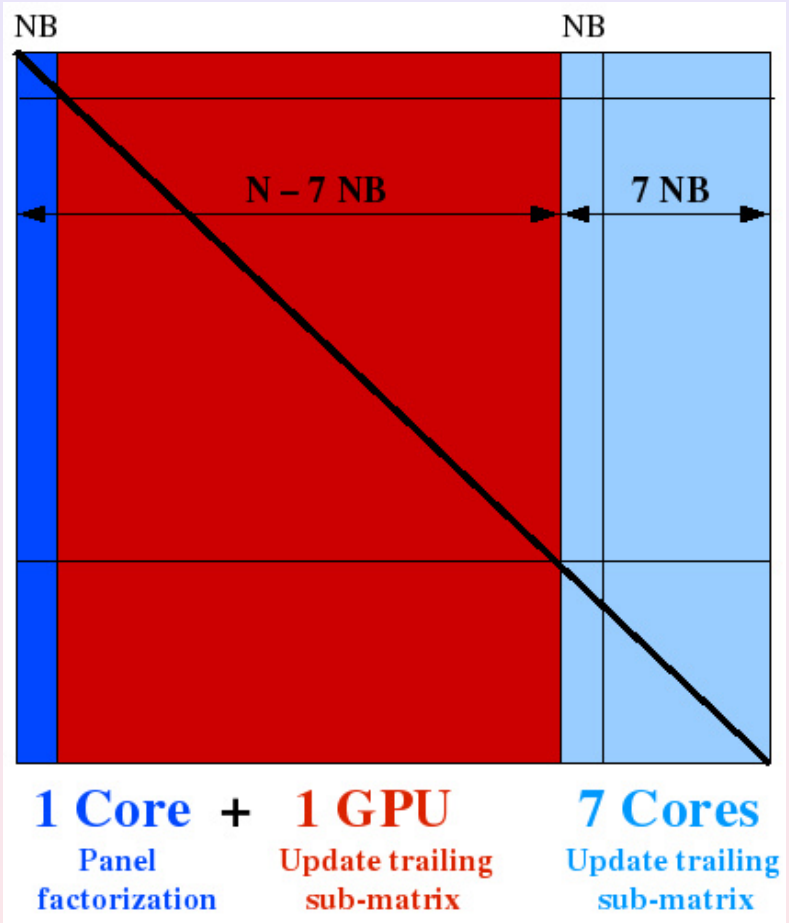
$$(A+\Delta A)\hat{x} = b, \quad \|\Delta A\|_\infty \leq nu\|A\|_\infty \left(3 + 5\|L\|_\infty \frac{\|U\|_\infty}{\|A\|_\infty}\right) + \mathrm{O}(\mathrm{u}^2)$$

We define $\rho_L = \|L\|_\infty$ and $\rho_U = \|U\|_\infty / \|A\|_\infty$.

Bound on the probability of large growth factors (no pivoting)

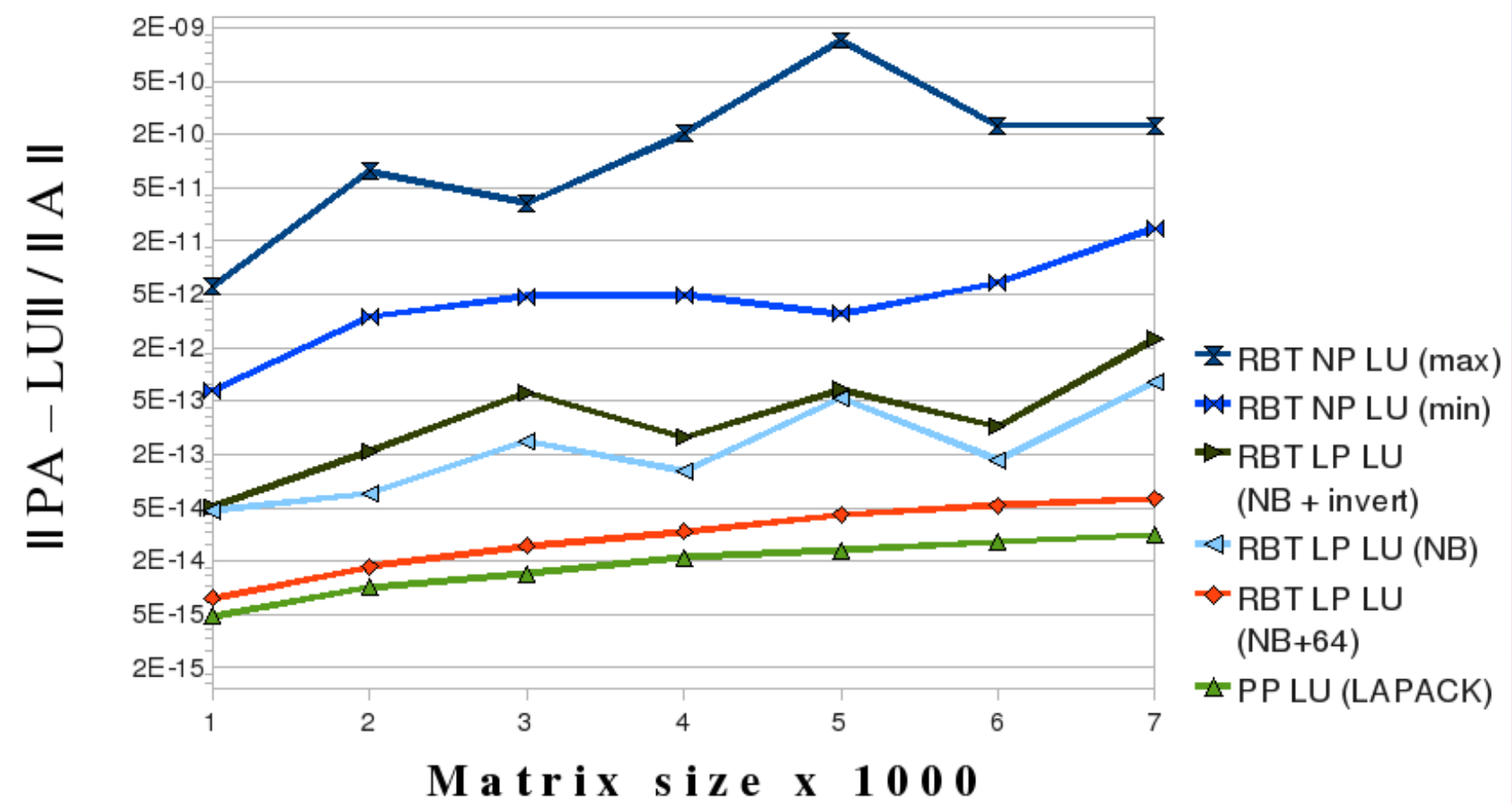
- $Prob(\rho_U > n^{2.5}) \rightarrow 0$ as $n \rightarrow \infty$
- $Prob(\rho_L > n^3) \rightarrow 0$ as $n \rightarrow \infty$

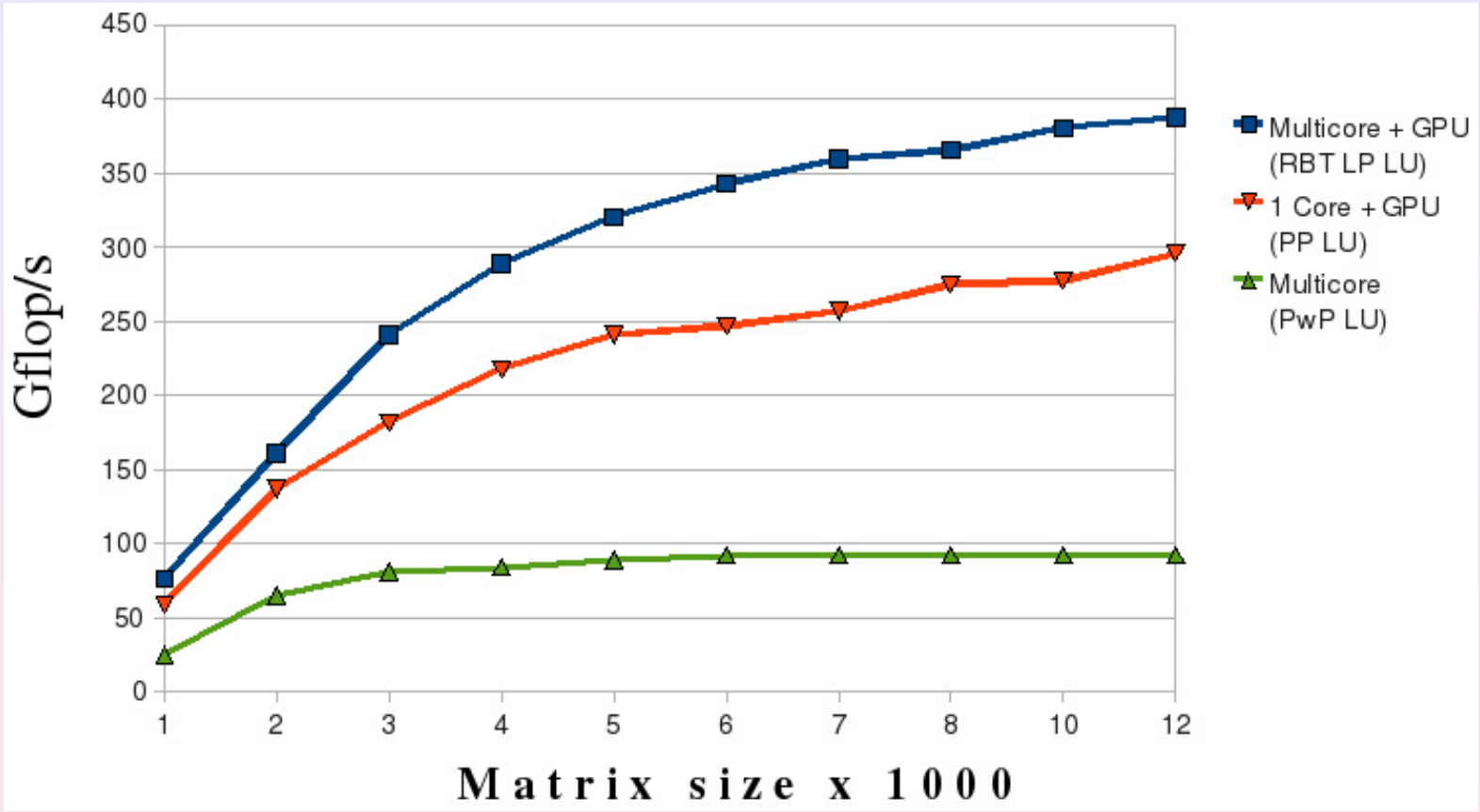
Hybrid RBT LU factorization



Load splitting for a hybrid LU factorization (8 cores+GPU)

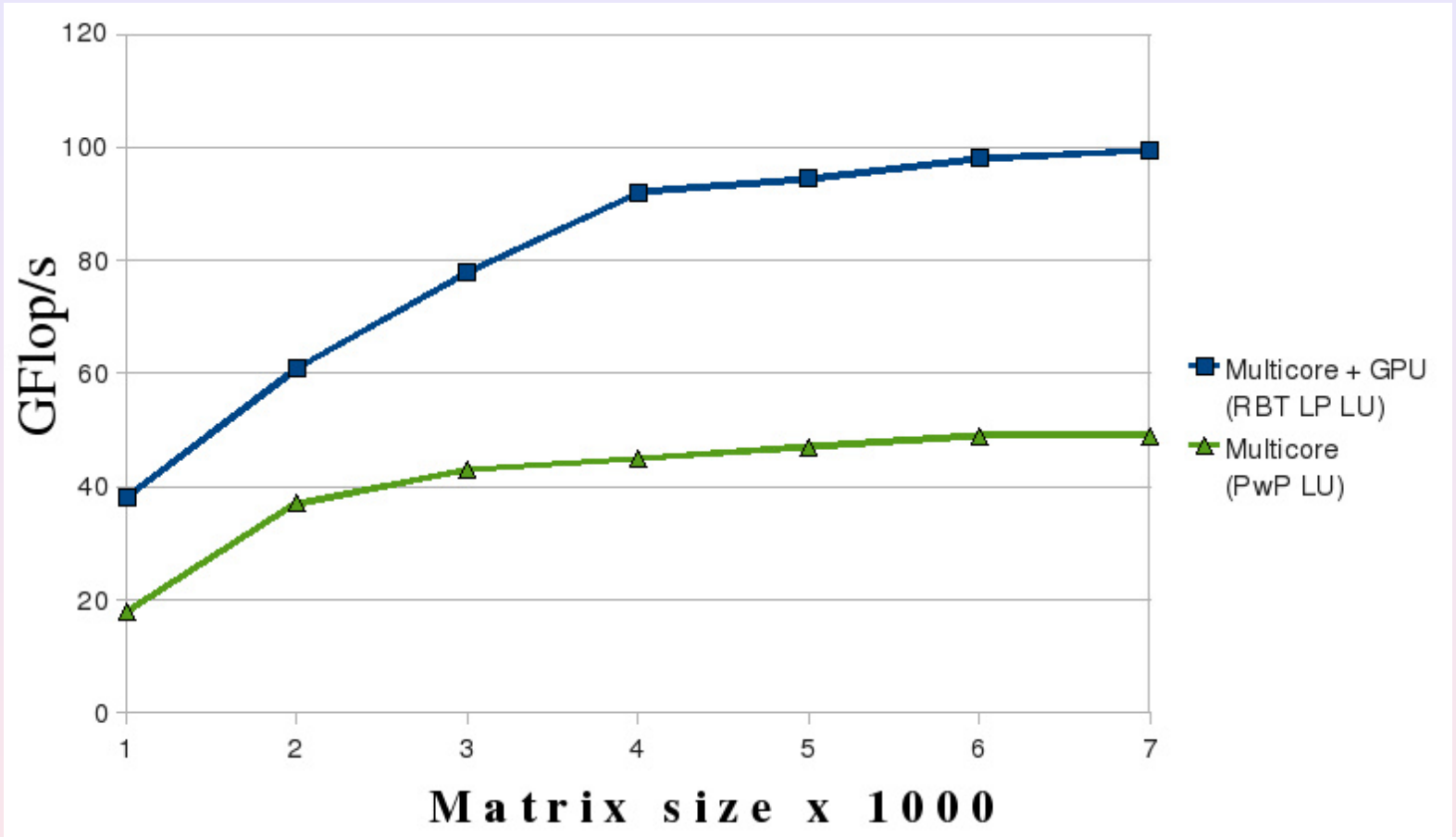
Accuracy of RBT





Performance of RBT LU factorization (single precision)

Intel Xeon (2 x 4 cores @ 2.33 GHz) - GeForce GTX 280 (240 Cores @ 1.30 GHz).



Performance of RBT LU factorization (double precision)

Intel Xeon (2 x 4 cores @ 2.33 GHz), GeForce GTX 280 (240 Cores @ 1.30 GHz).

- MAGMA version 0.2 available with linear system solvers (single and double precision): LU, QR, Cholesky factorizations, including mixed precision iterative refinement
- Statistical approach very promising for accelerating linear algebra computations on multicore-GPU architectures (linear systems, condition number estimates)
- Future work: least squares (with mixed precision) and eigenvalue solvers, condition estimates
- More details at `http://icl.cs.utk.edu/magma/`

Some references for this talk

[1] 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).

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

[3] S. Tomov, J. Dongarra,
Accelerating the reduction to upper-Hessenberg form through hybrid GPU-based computing.
LAPACK Working Note 219 (2009).

[4] M. Baboulin, J. Demmel, J. Dongarra, S. Tomov, V. Volkov,
Enhancing the performance of dense linear algebra on GPUs.
Supercomputing (SC'08), Austin, USA, Nov. 15-21, 2008.

[5] 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).