Accelerating linear algebra computations with hybrid GPU-multicore systems.

Marc Baboulin

INRIA/Université Paris-Sud

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

4th Workshop INRIA-Illinois

11/23/2010

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

- LAPACK, ScaLAPACK, sparse solvers, iterative solvers...have to be rethought and rewritten
- Need for fast Dense Linear Algebra (DLA) kernels in scientific calculations

Outline

1. Taking advantage of new parallel architectures
   - Towards hybrid GPU-multicore algorithms
   - Mixed precision algorithms

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

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

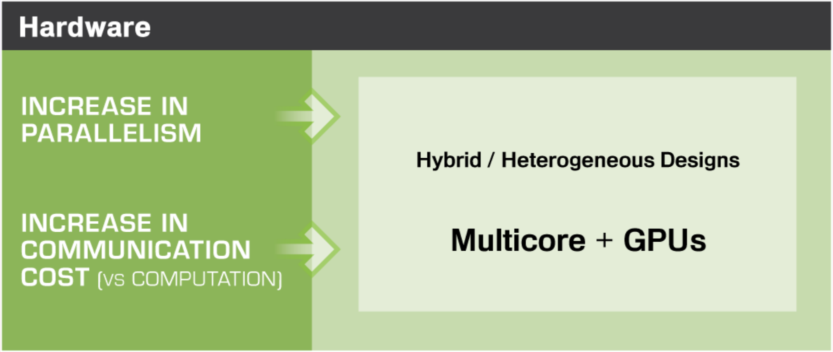
3. Conclusion

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
3. Conclusion

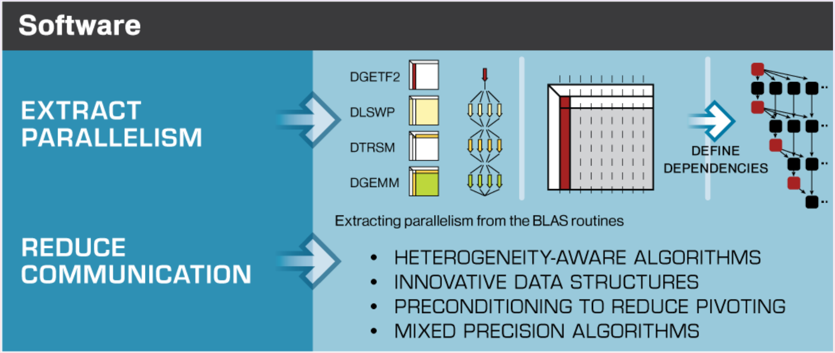
Outline

Hardware to software trends



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

Hardware to software trends



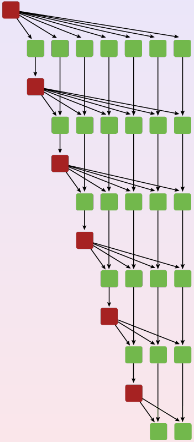
Motivation for heterogeneity-aware algorithms

- GPUs evolution: applications far beyond graphics, high bandwidth, programmability (CUDA), memory hierarchy, double precision arithmetic...
- Architectural trends have moved towards heterogeneous (CPU+GPU) designs
- Fully exploit the computational power that each of the hybrid components offers
- Need for linear algebra routines for hybrid systems: there is no self-contained library like LAPACK

Designing algorithms for Multicore+GPU

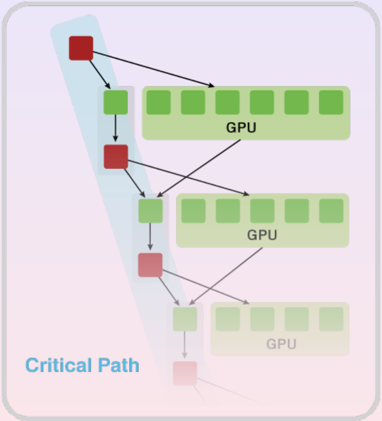
- Represent LAPACK algorithms as a collection of BLAS-based tasks and dependencies among them
  $\rightarrow$ rely on high performance of (CU)BLAS
- Abstract us from specificities in programming a GPU
- Properly schedule the tasks execution over the multicore and the GPU
- **MAGMA**: **Matrix Algebra on GPU and Multicore Architectures**
  DLA library for heterogeneous/hybrid architectures starting with current Multicore+GPU systems
  LAPACK-style interface
  U. Tennessee, U. California Berkeley, INRIA

Task splitting and scheduling



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

Task splitting and scheduling

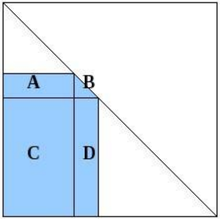


DAGs for hybrid systems
(both small and large tasks)

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
- More details in [Dongarra, Tomov, Baboulin, 2010]

Example: Cholesky factorization



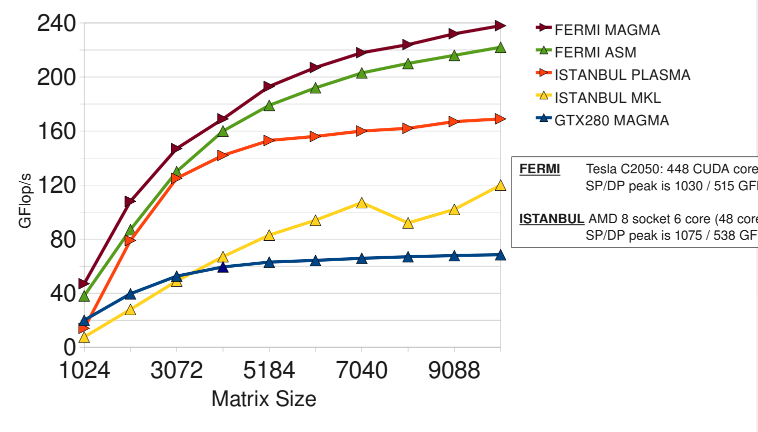
(1) $B = B - A A^T$	ssyrk	(GPU)
(2) $B = LL^T$	spotrf	(CPU)
(3) $D = D - CA^T$	sgemm	(GPU)
(4) $D = D \backslash B$	strsm	(GPU)

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

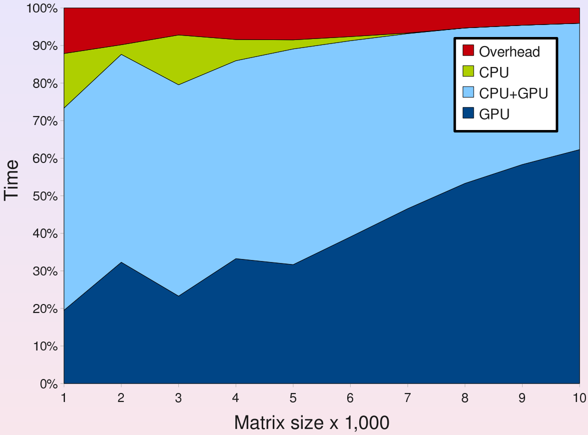
Standard LAPACK pseudo-code	**Hybrid Single Core-GPU code**
ssyrk_("L", "N", &jb, &i__3, &c_b13, a_ref(j,1), ...)	cublasSsyrk('L', 'N', jb, i__3. c_b13, a_ref(j,1), ...)
spotrf_("L", &jb, a_ref(j, j), lda, info)	cublasGetMatrix(jb, jb, 4, a_ref(j, j), *lda, work, jb)
sgemm_("N", "T", &i__3, ...)	cublasSgemm('N', 'T', i__3, ...)
	spotrf_("L", &jb, work, &jb, info)
	cublasSetMatrix(jb, jb, 4, work, jb, a_ref(j, j), *lda)
strsm_("R", "L", "T", "N", &i__3, ...)	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

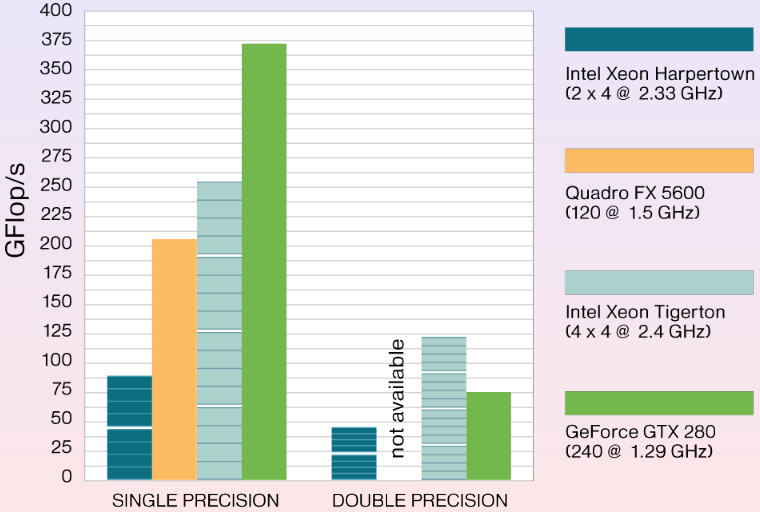


Time breakdown for MAGMA QR (single precision)

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

Outline

PEAK GEMM ON CURRENT MULTICORES vs GPUs



Mixed precision algorithms

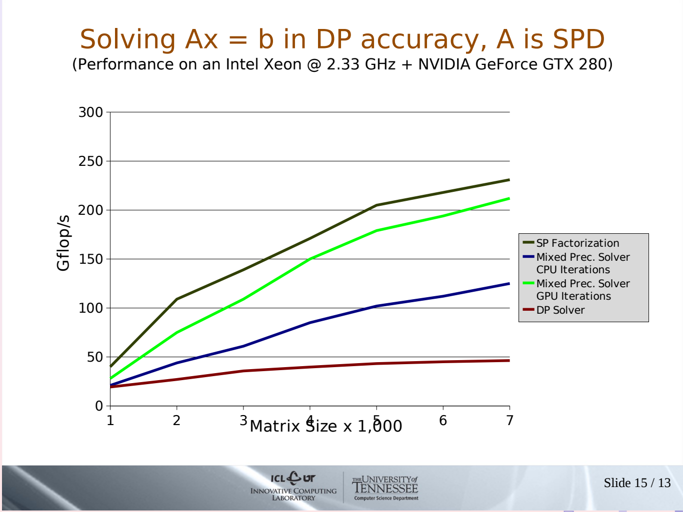
- Bulk of the computation in 32-bit arithmetic
- Postprocess the 32-bit solution by refining it into a solution that is 64-bit accurate
  Can be performed on the GPU
- Problem must be ”not ill-conditioned”
- Software details in:
  M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, 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

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

Mixed precision Cholesky factorization



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)

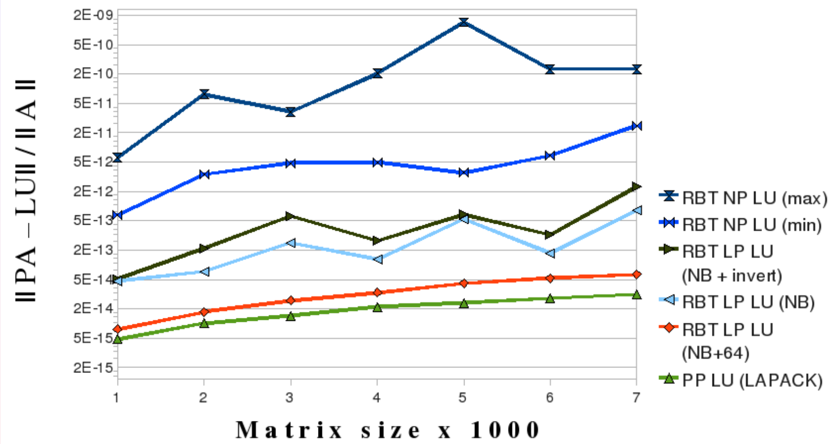
- [Parker,1995] proposed to make the matrix sufficiently "random" so that, with probability close to 1, pivoting is not needed
- **Precondition A with random matrices:** UAV to solve $Ax = b$, we instead solve $(UAV)y = Ub$ followed by $x = Vy$
- Random matrices U and V are chosen among a particular 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)

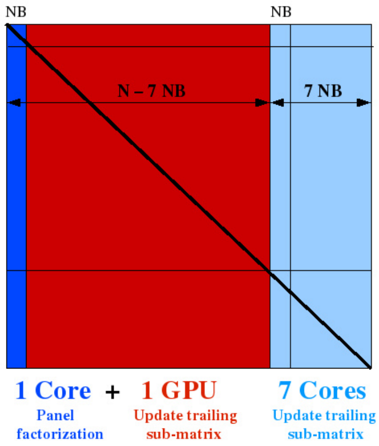
- Method:**LU with no pivoting on a preconditioned matrix**
- The preconditioning is "cheap" ($\mathcal{O}(n^2)$ operations)
- We do not pivot (RBT NP) or just within the first few rows of the panel (RBT LP)
  $\rightarrow$ we have a fully BLAS 3 algorithm
- RBT may require some steps of iterative refinement 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

Accuracy of RBT

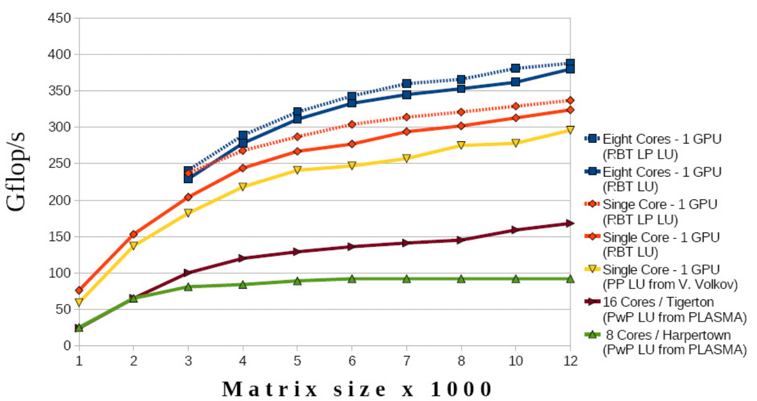


Hybrid RBT LU factorization



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

Performance (DP)



Performance of RBT LU factorization (double precision)

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

- **Hybrid CPU/GPU library** available (MAGMA 0.2) with main linear system solvers, including mixed precision iterative refinement
  More details at `http://icl.cs.utk.edu/magma/`
- **Randomization** is very promising for accelerating linear algebra computations on multicore and/or GPU architectures
- **Some ongoing work**:
  -Hybrid implementation for SVD and eigensolvers
  -Apply statistical techniques to estimate condition number for very big problems
- Starting collaboration with Wen-mei Hwu (UIUC) and student Liwen Chang

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).