Enhancing the Performance of Dense Linear Algebra Solvers on GPUs in the MAGMA Project

Marc Baboulin
University of Coimbra (Portugal)

Jim Demmel
University of California, Berkeley (USA)

Jack Dongarra
University of Tennessee, Knoxville (USA)
Oak Ridge National Laboratory (USA)
University of Manchester (UK)

Stanimire Tomov
University of Tennessee, Knoxville (USA)

Vasily Volkov
University of California, Berkeley (USA)

Matrix Algebra on GPU and Multicore Architectures (MAGMA)

The MAGMA project, headed by the linear algebra research groups at University of Tennessee, UC Berkeley, and UC Denver, aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current 'Multicore+GPU' systems. This transition cannot be done automatically as in many cases new algorithms that significantly differ from algorithms for conventional architectures, will be needed. Preliminary studies – on a new class of 'heterogeneity-aware' algorithms of 'reduced communication' and 'high-parallelism', as shown in this poster – confirm that this is the case.

1 INTRODUCTION TO DENSE LINEAR ALGEBRA (DLA) FOR GPUs

DLA Algorithms, due to a high ratio of floating point calculations to data required, have been of high performance

Therefore, special purpose architectures have not been able to significantly accelerate them up until recently

- Fatahalian et al. study SGEMM (in 2004) to conclude CPUs almost always outperform GPUs (only ATI X800XT produced 12 Gflop/s in single precision, comparable with a 3 GHz Pentium 4)
- Galoppo et al. (in 2005) had similar results on LU (5.7 Gflop/s in single precision on an NVIDIA 7800, compared to 3.4GHz Pentium 4 at the time)

This has changed as CPUs move to multi/manycores with an exponentially growing gap between processor speed and memory (and bandwidth shared between cores), while GPUs have consistently outpaced them both in performance and memory bandwidth

First CUDA GPU results to significantly outperform CPUs on DLA started appearing at the beginning of 2008 (illustrated also on Figure 1 for the GEMM operation)

- In January 2008 V.Volkov and J. Demmel [1] reported on SGEMM kernel (among others) to significantly outperform the CUBLAS library (125 Gflop/s vs more then 180 Gflop/s in their implementation) and an LU factorization running at up to 140 Gflop/s in single precision arithmetic (SP)
- In March S. Tomov et al. [2] presented at PPSC08 Cholesky factorization running at up to 160 Gflop/s in SP using Volkov's SGEMM kernel (also described in [3])
- In May V.Volkov and J. Demmel [4] described LU, QR, and Cholesky running at up to 180 Gflop/s in SP
- In May, Dongarra et al. [5] reported on SP Cholesky running at 325 Gflop/s on a pre-release NVIDIA card

GPU GEMM ON CURRENT MULTICORES vs GPUs

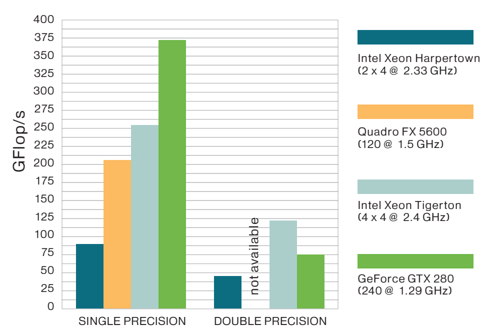


Figure 1

Peak measured GEMM performance on current multicore (from Intel) and GPU (from NVIDIA) architectures.

Note that in SP the GTX 280 is **10 x faster** than a quad-core processor (at 2.33GHz) and still **75 GFlop/s** faster than an entire quad-socket quad-core Intel Xeon Tigerton system (cores running at 2.4 GHz)

HARDWARE TO SOFTWARE TRENDS

Note: For the trends in Multicore architectures, we refer to the PLASMA project [8] and to new algorithms minimizing communication [9, 10, 11]

Hardware

INCREASE IN PARALLELISM

INCREASE IN COMMUNICATION COST (VS COMPUTATION)

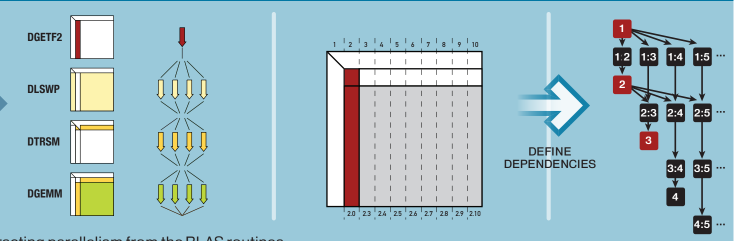
Hybrid / Heterogeneous Designs

Multicore + GPUs

New Algorithms Needed

Software

EXTRACT PARALLELISM
Section 2.1



Extracting parallelism from the BLAS routines

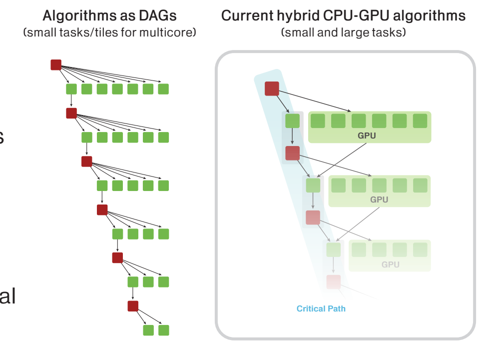
REDUCE COMMUNICATION
Sections 2.2-2.5

- HETEROGENEITY-AWARE ALGORITHMS
- INNOVATIVE DATA STRUCTURES
- PRECONDITIONING TO REDUCE PIVOTING
- MIXED PRECISION ALGORITHMS

2 GPU ALGORITHMS ≠ TRADITIONAL ALGORITHMS

2.1 EXTRACTING PARALLELISM

1. Splitting Algorithms into tasks
   - The concept of representing algorithms as Directed Acyclic Graphs (DAGs) where the nodes represent the sub-tasks and the edges the dependencies
   - Heterogeneity-aware splitting
2. Scheduling task execution
   - Crucial for performance, for example scheduling tasks on the critical path 'as soon as possible' frees more parallelism
3. GPU triangular solvers through explicitly inverting the triangular matrix
   - Significantly accelerates both TRSM (up to 3 times) and TRSV (order of magnitude)



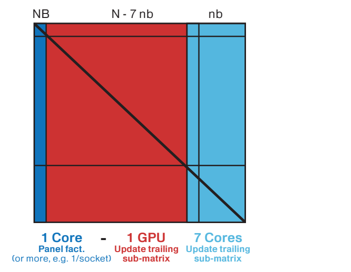
2.2 HETEROGENEITY-AWARE ALGORITHMS

In particular, algorithms for hybrid GPU + multicore computing should split the computation to fully exploit the power that each of the hybrid components offers.

1. 'Small' tasks of low parallelism to be executed on the CPU (for example tasks on the critical path)
2. Bigger tasks of high parallelism to be executed on the GPU
3. Proper scheduling should explore asynchronicity between CPU and GPU
4. Blocking strategies
   - Varying block sizes (as in QR [4])
   - Two-level blocking (as in Cholesky [4])

Note: see [4] for other heterogeneity-related techniques and detail on 1..4

5. Work partitioning (specific) for hybrid GPU + Multicore [6]



EXAMPLE

An LU factorization work splitting for Single GPU + 8 cores CPU host

The first N – 7nb columns reside on the GPU and are processed by 1 GPU + 1 core, the rest resides and is processed by the remaining cores

2.3 INNOVATIVE DATA STRUCTURES

Non-traditional data layouts may be beneficial in reducing communication costs.

For example, to avoid severely penalized strided memory access in pivoting on the GPU, the matrix is laid out in the GPU memory in row-major order - this reduces the pivoting overhead from 50% (of the total computation) to only 1% (on NVIDIA's GTX280) in SP!

2.4 PRECONDITIONING FOR REDUCED PIVOTING

1. Here Preconditioning is a Random Butterfly Transformation (RBT) with cost of applying negligible compared to the factorization, and meant to transform the original matrix into a "sufficiently random" matrix s.t. with probability close to 1 pivoting would not be needed [3]
   - We use unitary RBT (does not change the condition number): can be represented as 2 x 2 block matrix where the blocks are diagonal matrices
   - To solve Ax = b using butterfly matrices U and V we solve (U* A V) y = U*b, followed by x = V y
2. RBT helps but in general accuracy is reduced. Two ways to improve it (if needed) are [3,6]:
   - Add limited pivoting (LP) within the block size (denoted by NB) or more (for example: LP LU(NB+64) is a LU with pivoting limited to the 1st NB+64 rows of the current panel)
   - Add iterative refinement in the working precision

2.5 MIXED PRECISION ALGORITHMS

Accelerating solvers using mixed precision arithmetic:

The algorithm: (e.g. in solving A x = b, A is SPD)

```
1  Factorization A = LL^T     (SP)
2  Solve L y = b              (SP)
3  Solve L^T x = y            (SP)
   do k = 1, 2, . . .
4     r = b – A x             (DP)
5     Solve L y = r           (SP)
6     Solve L^T z = y         (SP)
7     x += z                  (DP)
      check convergence
   done
```

This technique is feasible because it uses much faster SP arithmetic for the bulk of the computation and DP iterations to raise the accuracy (if needed).

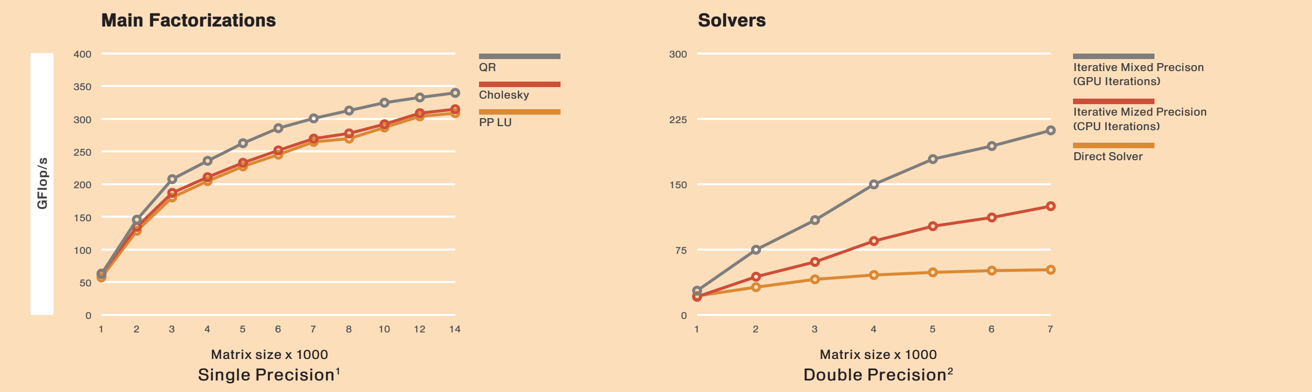
Ratios between SP and DP on the GTX 280 are much higher than those on traditional CPUs:

Theoretical peak: ~ 10
SGEMM/DGEMM: ~ 5
One sided factorizations: ~ 4 (to 5)

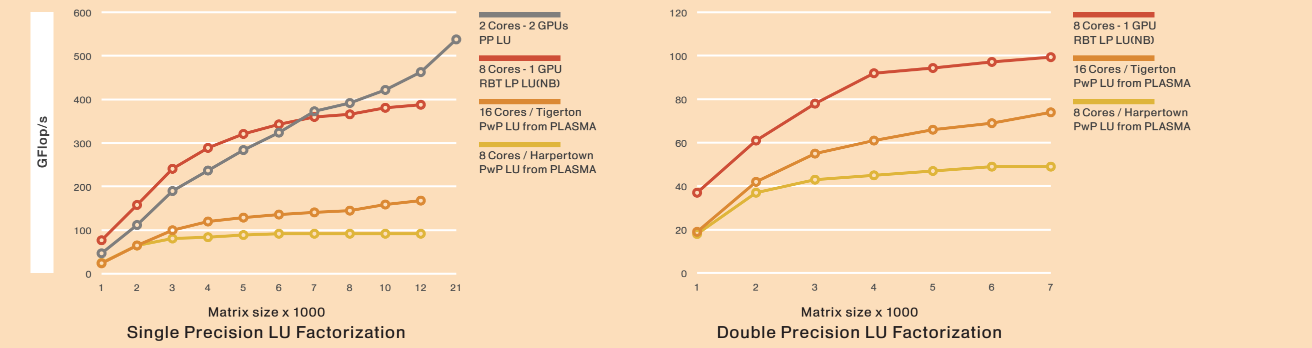
often leading to speedup factors of 4 !

3 PERFORMANCE RESULTS

Single Core + Single GPU



Multicores / Multi-GPUs



HARDWARE USED
GPU: GeForce GTX 280 (240 Cores @ 1.30 GHz)
Host: Intel Xeon (2 x 4 Cores @ 2.33 GHz)
Tigerton: Intel Xeon (4 x 4 Cores @ 2.4 GHz)
Harpertown: Intel Xeon (2 x 4 Cores @2.33 GHz)

[1] Note that QR runs at a higher MFlop rate than Cholesky: Cholesky has less thread-level parallelism in GEMM, as it deals with triangular matrices.

[2] Mixed precision solvers often achieve 4 x speedup compared to DP solvers but the speed depends on the conditioning of the matrix. In these performance results, we considered 3 steps of iterative refinement (on symmetric and positive definite matrices using Cholesky).

2.6 SPEED VS ACCURACY TRADE-OFFS

The new techniques often gain in speed for the price of relaxed accuracy. Understanding this trade-off of speed vs accuracy can lead to very efficient algorithms, for example

1. Using technique 2.1.3 within stable algorithms like LU with partial pivoting (PP LU) results in stable LU of practically the same accuracy [4].
2. Numerical experiments show that technique 2.4 is stable in practice and can be used in combination with 2.1.3 as well. Experiments with random matrices show that
   - RBT LU(NB+64) is comparable in accuracy to PP LU[6]
   - RBT LU(NB) loses from 1 to 2 digits of accuracy to gain up to 30% in speed compared to PP LU[6].
3. Technique 2.5 can be used to obtain solutions of prescribed accuracy as long as the conditioning of the problem does not exceed the reciprocal of the SP accuracy [7].

CONCLUSIONS

1. GPU computing has reached a point to significantly outperform current multicores on DLA (in spite of DLA's traditionally high performance on x86 architectures).
2. Architecture trends have moved towards heterogeneous (GPU + CPU) designs of increased parallelism and communication costs, and software trends have to reflect that: we addressed this with innovative heterogeneity-aware algorithms/techniques on extracting parallelism and reducing communication.
3. There are significant differences between the new algorithms and those for conventional CPUs.
4. The new techniques in many cases present an opportunity for trade-off between speed and accuracy.
5. The need for DLA for hybrid systems will grow

MOTIVATING OUR **FUTURE WORK DIRECTIONS,** AS ENVISIONED IN THE MAGMA PROJECT,

TOWARDS A SELF CONTAINED **DLA LIBRARY** SIMILAR TO LAPACK BUT **FOR HETEROGENEOUS ARCHITECTURES.**

REFERENCES

1 V. Volkov and J. Demmel, Using GPUs to accelerate linear algebra routines, Poster at PAR lab winter retreat, January 9, 2008, http:/www.eecs.berkeley.edu/~volkov/volkov08_parlab.pdf.

2 S. Tomov, M. Baboulin, J. Dongarra, S. Moore, V. Natoli, G. Peterson, and D. Richie, Special-purpose hardware and algorithms for accelerating dense linear algebra, Presentation at PPSC, Atlanta, March 12-14, 2008, http://www.cs.utk.edu/~tomov/PP08_Tomov.pdf.

3 M. Baboulin, J. Dongarra, and S. Tomov, Some issues in dense linear algebra for multicore and special purpose architectures, LAPACK Working Note 200.

4 V. Volkov and J. Demmel, LU, QR and Cholesky factorizations using vector capabilities of GPUs, Tech. Report UCB/EECS-2008-49, EECS Department, University of California, Berkeley, May 2008.

5 J.Dongarra, S.Moore, G.Peterson, S.Tomov, J.Allred, V.Natoli, and D.Richie, Exploring new architectures in accelerating CFD for Air Force applications, HPCMP User Group Conference 2008 (July 14-17, 2008), http://www.cs.utk.edu/~tomov/ugc2008_final.pdf

6 S. Tomov, J. Dongarra, and M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, LAPACK Working Note 210.

7 J.Langou, J.Langou, P.Luszczek, J.Kurzak, A.Buttari, and J.Dongarra, Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems), sc 0 (2006), 50.

8 A.Buttari, J.Langou, J.Kurzak, and J.Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures, LAPACK Working Note 191.

9 J.Demmel, L.Grigori, H.Xiang, Communication-avoiding Gaussian Elimination, Supercomputing '08.

10 J.Demmel, L.Grigori, M.Hoemmen,J.Langou, Communication Optimal Parallel and Sequential QR and LU Factorizations, SIAM J. Sci. Comp (submitted), UC Berkeley Tech Report EECS-2008-89.

11 J.Demmel, L.Grigori, M.Hoemmen, J.Langou, Implementing Communication Optimal Parallel and Sequential QR and LU Factorizations, SIAM J. Sci Comp (submitted).

UNIVERSITY OF CALIFORNIA BERKELEY · UNIVERSITY OF COIMBRA · UNIVERSITY OF COLORADO DENVER · THE UNIVERSITY of TENNESSEE · The Parallel Computing Laboratory · SPONSORED BY The MathWorks · Microsoft · NVIDIA