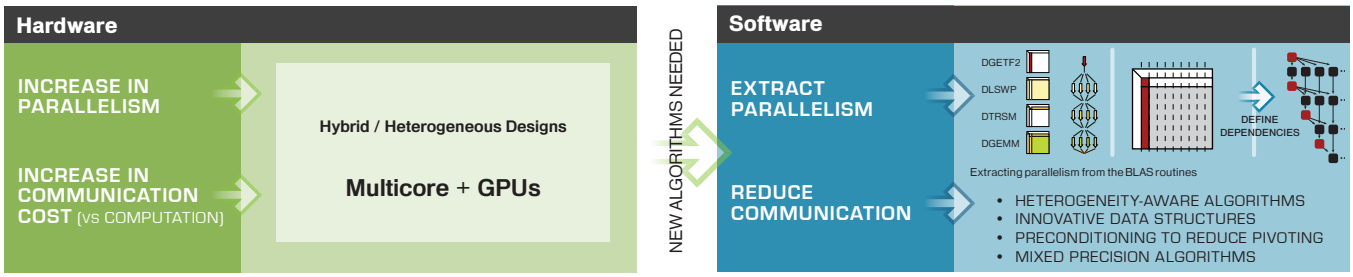
MAGMA

http://icl.cs.utk.edu/magma/

MATRIX ALGEBRA ON GPU AND MULTICORE ARCHITECTURES

The MAGMA project, led by the linear algebra research groups at University of Tennessee, UC Berkeley, and UC Denver, aims to develop a 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" confirm that this is the case.

HARDWARE TO SOFTWARE TRENDS

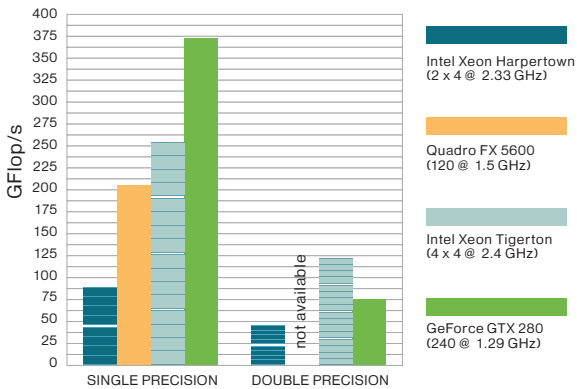


DENSE LINEAR ALGEBRA (DLA) FOR GPUs

DLA Algorithms, due to high ratio of floating point calculations to data required, have been of high performance on standard architectures. Therefore, special purpose architectures have not been able to significantly accelerate them up until recently. This has changed as CPUs move to multi/manycores with an exponentially growing gap between processor speed and memory, 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 below for the GEMM operation and on the reverse page for the main factorizations and solvers).

PEAK GEMM ON CURRENT MULTICORES vs GPUs



HARDWARE TRENDS AND CURRENT WORK ON MAGMA SHOW:

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 on that. MAGMA addresses 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.**

SPONSORED BY The MathWorks Microsoft NVIDIA NSF

University of California Berkeley
University of Coimbra
University of Colorado Denver Downtown Campus
The University of Tennessee
The Parallel Computing Laboratory

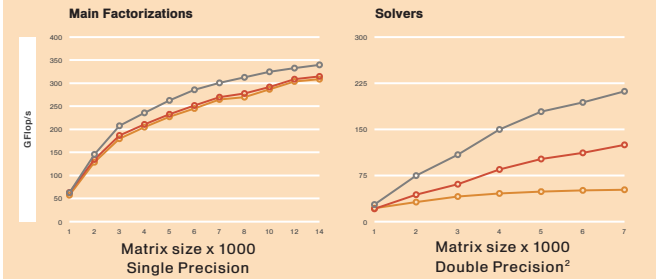
INNOVATIVE COMPUTING LABORATORY
CENTER for INFORMATION TECHNOLOGY RESEARCH

MAGMA

http://icl.cs.utk.edu/magma/

PERFORMANCE RESULTS[1]

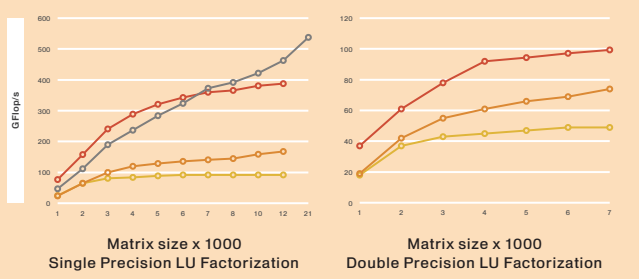
Single Core + Single GPU



QR
Cholesky
PP LU

Iterative Mixed Precison (GPU Iterations)
Iterative Mixed Precision (CPU Iterations)
Direct Solver

Multicores / Multi-GPUs



2 Cores + 2 GPUs [Partial Pivoting (PP) LU]
8 Cores + 1 GPU [Preconditioned Limited Pivoting (LP) LU(NB)[3]]
16 Cores / Tigerton [Pairwise Pivoting (PwP) LU from PLASMA]
8 Cores / Harpertown [PwP LU from PLASMA]

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

[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 three steps of iterative refinement (on symmetric and positive definite matrices using Cholesky).

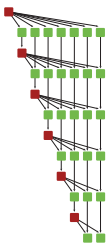
[3] Limited amount of pivoting (within the block size NB or more) is justified by a specially designed unitary transformation: experiments with random matrices show that LP LU(NB+64) for example is comparable in accuracy to PP LU, and LP LU(NB) loses only from 1 to 2 digits of accuracy to gain up to 30% in speed compared to PP LU.

GPU ALGORITHMS ≠ TRADITIONAL ALGORITHMS

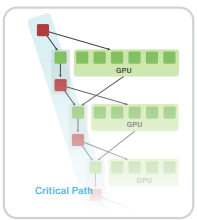
EXTRACTING PARALLELISM

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)

Algorithms as DAGs
(small tasks/tiles for multicore)



Current hybrid CPU-GPU algorithms
(small and large tasks)

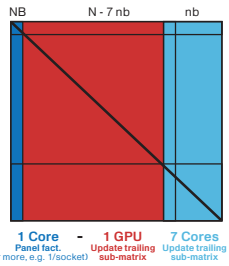


REDUCE COMMUNICATION

A. HETEROGENEITY-AWARE ALGORITHMS

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. Larger 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)
   - Two-level blocking (as in Cholesky)
5. Work partitioning (specific) for hybrid GPU + Multicore



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

B. INNOVATIVE DATA STRUCTURES

Non-traditional data layouts may be beneficial in reducing communication costs, e.g. 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 (to often double the performance).

C. PRECONDITIONING FOR REDUCED PIVOTING

D. MIXED PRECISION ALGORITHMS

E. TILED ALGORITHMS

F. COMMUNICATION-OPTIMAL ALGORITHMS