MAGMA: Evolution and Revolution

Stan Tomov
Innovative Computing Laboratory
University of Tennessee, Knoxville

ICL Lunch Talk Seminar
July 2, 2021
Matrix Algebra on GPU and Multicore Architectures

- Shared memory systems
- BLAS/LAPACK on GPUs
- Hybrid CPU-GPU Algorithms
  - Linear system solvers (+ mixed precision)
  - Eigenvalue problem solvers
- Batched LA
  - BLAS-3 (fixed/variable), LU, QR, Cholesky
- Sparse LA
  - Solvers: BiCG, BiCGSTAB, GMRES
  - Preconditioners: ILU, Jacobi,
  - SPMV, SPMM (CSR, ELL, … etc.)

**PERFORMANCE & ENERGY EFFICIENCY**

MAGMA LU factorization in double-precision arithmetic

- Support CUDA and ROCM/HIP
- [https://icl.utk.edu/magma](https://icl.utk.edu/magma)
GPUs: excelling in graphics rendering

Scene model → Graphics pipelined computation → Final image

Repeated fast over and over: e.g. TV refresh rate is 30 fps; limit is 60 fps

This type of computation:
- Requires enormous computational power
- Allows for high parallelism
- Needs high bandwidth

GPU Evolution from 1999 to 2007 (GeForce1 to GeForce8 Series)

- Some applications were exploring GPUs use
- Some LA was developed but was “slow”
- No memory hierarchy for data reuse
- E.g., even a dot product would be computed using the graphics pipeline with multiple passes through the data to reduce it to single number
CUDA first appears in 2007
CUDA GPUs have memory hierarchy
- Enables development of more efficient LA
- BLAS can be efficiently implemented
- More applications start to explore GPU use
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

LAPACK for GPUs (00’s)?

Hybrid algorithms (CPU + GPU)

Performance (SP) on Quadro FX5600 GPU

- 1st hybrid CPU+GPU LA algorithms
- LU, QR, and Cholesky
- LU to avoid pivoting (RBT)

---

Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Software/Algorithms</th>
<th>Year</th>
<th>GB/s</th>
<th>SP (Gflop/s)</th>
<th>CUDA GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (70’s)</td>
<td>2007</td>
<td>62</td>
<td>416</td>
<td>GeForce 8800 GTS (Tesla)</td>
</tr>
<tr>
<td></td>
<td>2008</td>
<td>70</td>
<td>470</td>
<td>GeForce 9800 GTX (Tesla)</td>
</tr>
</tbody>
</table>

**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

**LAPACK for GPUs (00’s)?**
- Hybrid algorithms (CPU + GPU)

---


- 1st hybrid CPU+GPU LA algorithms
- LU, QR, and Cholesky
- LU to avoid pivoting (RBT)

- Very efficient sgemm (61% of peak) vs. CUBLAS 1.1 (37% of peak)
- Volkov and Demmel made the code code available and we used it to further accelerate LU, QR, and Cholesky
Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Year</th>
<th>GB/s</th>
<th>SP (Gflop/s)</th>
<th>CUDA GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>62</td>
<td>416</td>
<td>GeForce 8800 GTS (Tesla)</td>
</tr>
<tr>
<td>2008</td>
<td>70</td>
<td>470</td>
<td>GeForce 9800 GTX (Tesla)</td>
</tr>
</tbody>
</table>

- 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

LAPACK for GPUs (00’s)?
- Hybrid algorithms (CPU + GPU)
  - Rely on BLAS tasking
  - Hybrid scheduling

LU and RBT LU performance (SP) on GTX 280 GPU

Performance in DP

- 1st hybrid CPU+GPU LA algorithms
- LU, QR, and Cholesky
- LU to avoid pivoting (RBT)
- Look-ahead scheduling to overlap CPU work with GPU
- Computing and performance in DP

---


Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Year</th>
<th>GB/s</th>
<th>SP (Gflop/s)</th>
<th>CUDA GPUs</th>
<th>MAGMA 2.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>62</td>
<td>416</td>
<td>GeForce 8800 GTS (Tesla)</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>70</td>
<td>470</td>
<td>GeForce 9800 GTX (Tesla)</td>
<td></td>
</tr>
</tbody>
</table>

**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

**MAGMA (00’s)**
- Hybrid algorithms (CPU + GPU)
  - Rely on - BLAS tasking
  - Hybrid scheduling

---


---

1. **1st hybrid CPU+GPU LA algorithms**
2. **LU, QR, and Cholesky**
3. **LU to avoid pivoting (RBT)**
4. **Look-ahead scheduling** to overlap CPU work with GPU
5. **Computing and performance in DP**
6. **First mention of MAGMA project**
Matrix Algebra on GPU and Multicore Architectures

The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.

The MAGMA research is based on the idea that, to address the complex challenges of the emerging hybrid environments, optimal software solutions will themselves have to hybridize, combining the strengths of different algorithms within a single framework. Building on this idea, we aim to design linear algebra algorithms and frameworks for hybrid manycore and GPU systems that can enable applications to fully exploit the power that each of the hybrid components offers.

Please use any of the following publications to reference MAGMA.

MAGMA version 0.1

This release is intended for a single CUDA enabled NVIDIA GPU and includes the 3 one-sided factorizations - LU, QR, and Cholesky in single and double precision arithmetic.

Download View License
Algorithmic view of template code for GEMM

- GEMM performance is crucial for performance of LA (GEMM is about 12x faster than GEMV at the time)
- Parameterize and template code and use empirical auto-tuning
- Obtained substantial, up to 27%, speedup compared to CUBLAS 2.0

Auto-tuning rank-k update DGEMM (k=64)
(on GeForce GTX 280 GPU)

Hybrid algorithms for two-sided factorizations

- Eigen-solvers need two-sided factorizations but are notoriously slow on CPUs due to need of Level 2 BLAS computations
- Develop Level 2 BLAS to benefit GPU’s high-bandwidth
- Developed hybrid CPU+GPU two-sided factorizations (Hessenberg for general eigen-solvers, bidiagonal for SVD, and tridiagonal for symmetric eigen-solvers)

Performance in DP of Hessenberg reduction on GTX 280 GPU

Algorithmic view of template code of GEMM for Fermi

Performance of xSYMV on a GTX 280

- **Best GEMM performance** at the time (58% of peak in DP and 63% of peak in SP)
- **Added register blocking** to previous state-of-art
- Incorporated in CUBLAS
- More BLAS in MAGMA BLAS

---


Scaling of hybrid Cholesky on multiple C1060 GPUs (double precision)

- Best GEMM performance at the time (58% of peak in DP and 63% of peak in SP)
- Added register blocking to previous state-of-art
- Incorporated in CUBLAS
- More BLAS in MAGMA BLAS
- Multi-GPU
- Mixed-precision solvers

Mixed-precision LU solver on GTX280 GPU (double precision accuracy)

References:

- Auto-tuning GEMMs for Fermi (J. Kurzak, S. Tomov, J. Dongarra, 2011)
- Hybrid LAPACK algorithms (M. Horton, S. Tomov, J. Dongarra, 2011)

- Power-aware computing on GPUs (Kasichayanula, K., H. You, S. Moore, S. Tomov, H. Jagode, and M. Johnson, 2011)

- Multi-core and multi-GPU algorithms (F. Song, S. Tomov, J. Dongarra, 2011)
- Performance portability with the DAGuE framework (Bosllica, G., A. Bouteiller, T. Herault, P. Lemariner, N. Ohm Saengpatsa, S. Tomov, and J. Dongarra)

- Two major releases per year
  - MAGMA 1.0 released in August, 2011
Hybrid algorithms

- clMAGMA – OpenCL port of MAGMA used to add support to AMD GPUs
- clMAGMA 0.1 (April), clMAGMA 0.2 (May), clMAGMA 0.3 (June), clMAGMA 1.0 (December)
  - LU, QR, and Cholesky factorization and solvers
  - Two-sided factorizations, eigen-solvers and SVD
  - Orthogonal transformation routines

“From CUDA to OpenCL: Towards a Performance-portable Solution for
Multi-platform GPU Programming,”

- Communication-avoiding algorithms
  (Baboulin, M., S. Donfack, J. Dongarra, L. Grigori, A. Remi, and S. Tomov)
- Asynchronous methods (H. Anzt et al.)
- Non-GPU resident algorithms (I. Yamazaki et al.)
Toward fast Eigen solver

- Characteristics
  - Stage 1: BLAS-3, increasing computational intensity,
  - Stage 2: BLAS-1.5, new cache friendly kernel,
  - 4X/12X faster than standard approach,
  - Bottleneck: if all Eigenvectors are required, it has 1 back transformation extra cost.

MAGMA MIC – MAGMA port providing support for Intel Xeon Phi Coprocessors

- MAGMA MIC 0.3 (November 2012), MAGMA MIC 1.0 (May 2013)
- LU, QR, and Cholesky factorization and solvers
- Two-sided factorizations, eigen-solvers and SVD
- Orthogonal transformation routines

- MAGMA Sparse available since MAGMA 1.3 (November 2012)
- Extended continuously improved up to current MAGMA 1.5 release (September 2014)
- Krylov Subspace Solvers (H. Anzt, J. Dongarra, P. Luszczek, W. Sawyer, S. Tomov, I. Yamazaki)
- CA Krylov methods (I. Yamazaki et al.)
- LOBPCG and Blocked SpMM (H. Anzt et al.)
- Mixed-precision orthogonalization and adaptive CA-GMRES (I. Yamazaki et al.; best paper at VECPAR)
- Self-adaptive multiprecision preconditioners (H. Anzt et al.)
- CA-GMRES with multiple GPUs (I. Yamazaki et al.)
- Sparse Matrix Vector Products for GPUs (H. Anzt et al.)
- MAGMA Batched available at least since MAGMA 1.5 (September 2014)
  - Continuously developed, extended, and improved

- Batched BLAS
- Batched LAPACK
- Leading Batched BLAS and LAPACK standard development
- Batched LA is needed in many applications
- MAGMA provides most complete batched BLAS and LAPACK
  - Vendors have also started to provide some in their numerical libraries

---

**Data Analytics and associated with it Linear Algebra on small LA problems are needed in many applications:**

- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, etc.

---

**Applications using high-order FEM**

- Matrix-free basis evaluation needs efficient tensor contractions,
  \[ C_{i,j,k} = \sum A_{i,k} B_{j,k} \]
  - Within ECP CEED Project, designed MAGMA batched methods to split the computation in many small high-intensity GEMMs, grouped together (batched) for efficient execution:
    \[ \text{Batch}_{\{ C_{i,j,k} = A^T B_{i,j,k}, \text{for range of } i3 \}} \]

---

**References**


- Performance, design, and autotuning of batched GEMM for GPUs (A Abdelfattah et al.)
- High-performance matrix-matrix multiplications of very small matrices (I Masliah et al.)
- High-performance tensor contractions for GPUs (A Abdelfattah et al.)
- Accelerating Tensor Contractions for High-Order FEM on CPUs, GPUs, and KNLs (A. Haidar et al.)
- Performance Tuning and Optimization Techniques of Fixed and Variable Size Batched Cholesky Factorization on GPUs (A Abdelfattah et al.)
- A proposed API for batched basic linear algebra subprograms (J Dongarra et al.)

![Graph showing performance comparison of tensor contraction versions using batched $C = \alpha AB + \beta C$ on 100,000 square matrices of size $n$ on a K40c GPU and 16 cores of Intel Xeon E5-2670, 2.60 GHz CPUs.](image1)

![Graph showing effect of a Thread Block Concurrency (tbc) techniques where several DGEMMs are performed on one TB simultaneously.](image2)
- Performance, design, and autotuning of batched GEMM for GPUs (A. Abdelfattah et al.)
- High-performance matrix-matrix multiplications of very small matrices (I. Masliah et al.)
- High-performance tensor contractions for GPUs (A. Abdelfattah et al.)
- Accelerating Tensor Contractions for High-Order FEM on CPUs, GPUs, and KNLs (A. Haidar et al.)
- Performance Tuning and Optimization Techniques of Fixed and Variable Size Batched Cholesky Factorization on GPUs (A. Abdelfattah et al.)
- A proposed API for batched basic linear algebra subprograms (J. Dongarra et al.)
- MagmaDNN high-performance data analytics for manycore GPUs and CPUs
- ASGD training
- Sampling algorithms to update truncated SVD
- Out of memory algorithms (SVD and symmetric and indefinite problems)
- Symmetric Indefinite Solvers
- Convex optimizations
- Tensor contractions and other batched routines
- Mixed-precision iterative refinement using GPU Tensor Cores and FP16
  - “Investigating half precision arithmetic to accelerate dense linear system solvers”,
- MATEDOR – Matrix, Tensor, and Deep-learning Optimized Routines
- Tensor Contractions using Optimized Batch GEMM Routines
- Harnessing GPU's Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solvers and Achieve 74 Gflops/Watt on Nvidia V100 (Haidar et al., GTC18)
- The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques (A Haidar et al., ICCS18)
- Accelerating the SVD two stage bidiagonal reduction and divide and conquer using GPUs (M. Gates et al., J. Parallel Computing 2018)
Towards Half-Precision Computation for Complex Matrices: A Case Study for Mixed Precision Solvers on GPUs (A. Abdelfattah et al.)

Fast batched matrix multiplication for small sizes using half-precision arithmetic on GPUs (A. Abdelfattah et al.)

MagmaDNN: towards high-performance data analytics and machine learning for data-driven scientific computing (D Nichols et al., ISC19)

MagmaDNN 1.1 release
- Bug fixes and performance improvements;
- Distributed training;
- Hyperparameter optimization framework improvements;
- Benchmarks using MagmaDNN;
- Performance comparisons, accuracy validations, etc. (w/ TensorFlow, Theano, and PyTorch).

MAGMA 2.5 release
- Mixed-precision using Nvidia Tensor Cores
- LU and Cholesky native (GPU-only)

MAGMA LU factorization in DP arithmetic

- IBM Power 8: 2x10 cores @ 2.094 GHz
- NVIDIA Pascal GPU: 56 MP x 64 @ 1.33 GHz
- Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems (A. Haidar et al.)
- Matrix multiplication on batches of small matrices in half and half-complex precisions (A. Abdelfattah et al.)
- Investigating the benefit of fp16-enabled mixed-precision solvers for symmetric positive definite matrices using GPUs (A. Abdelfattah et al.)

- Integrating Deep Learning in Domain Sciences at Exascale (R. Archibald et al.; SMC20)
- MagmaDNN 1.2 release
  - oneDNN (MKL DNN) support including fully connected, convolutional, and pooling layers;
  - CMake build system;
  - Added NN examples including CNN, ResNet, AlexNet, LeNet, MNIST and CIFAR interactive, and VGG16;
  - Added examples for Tensor Math; C++ style formatter;
  - Modularized distributed optimizer;
  - CIFAR10, CIFAR100, and MNIST data loaders;
  - CUDA streams added;
  - Model summary prinout;
  - Spack package manager installation support.

- Design, Optimization, and Benchmarking of Dense Linear Algebra Algorithms on AMD GPUs (C. Brown, A. Abdelfattah, S. Tomov, J. Dongarra, HPEC’20)
- hipMAGMA 1.0 (March 2020) and hipMAGMA 2.0 (July 2020)
  - MAGMA BLAS and LAPACK
  - Batched BLAS and Batched LAPACK are ported to HIP
  - Working with AMD
MAGMA Today

- MAGMA has grown significantly in functionalities over the years
- Provided are around 3,000 routines (per architecture)
- Ports become challenge due to volume
- Still, there is functional and performance portability
  - Due to use of standards, e.g., BLAS for performance portability
  - Abstractions that keep base of code unchanged
  - Auto-source translation tools, when needed
    - The challenge are BLAS and low level kernels that may benefit architecture-specific tuning
  - Must add some auto-tuning framework to help performance portability

for architectures in

- CPUs + Nvidia GPUs (CUDA),
- CPUs + AMD GPUs (HIP & OpenCL),
- CPUs + Intel Xeon Phis,
- manycore (native: GPU or KNL/CPU),
- embedded systems, combinations, and software stack, e.g., since CUDA x

for precisions in

- s, d, c, z,
- half-precision (FP16),
- mixed, ...

for interfaces

- heterogeneous CPU/GPU, native, ...

- LAPACK
- BLAS
- Batched LAPACK
- Batched BLAS
- Sparse
- Tensors
- MAGMA-DNN
- Templates
- ...

for architectures in

- CPUs + Nvidia GPUs (CUDA),
- CPUs + AMD GPUs (HIP & OpenCL),
- CPUs + Intel Xeon Phis,
- manycore (native: GPU or KNL/CPU),
- embedded systems, combinations, and software stack, e.g., since CUDA x
MAGMA 2.6 Release (July 1, 2021)

- Added HIP support for AMD GPUs (former hipMAGMA) as part of MAGMA
- The CUDA and HIP functionalities are in sync
- MAGMA Sparse added for AMD GPUs with this release
- Support for makefile, cmake, or spack installation
- Added inertia computational routines for GPUs
- Performance improvements for AMD GPUs
- Performance improvement for magma_Xgesv_batched for small sizes
- Added Bunch-Kaufman GPU-only solver using BLAS calls (magma_zhetsr_gpu)
- Added include/magma_config.h file storing the configuration for a particular magma installation (CUDA vs. HIP, etc.)
- Added expert interfaces for magma_Xgetrf_gpu and magma_Xpotrf_gpu. These interfaces allow the user to specify the factorization mode; hybrid (CPU+GPU) vs. native (GPU only), as well as the blocking size (nb)
- Added tuning for small size LU, QR, and Cholesky factorizations
Next

- Mixed-precision solvers, e.g., to harness TC hardware for fast FP16
- Batched LA, e.g., SVD, solvers
- Performance optimizations
  - Tuning for new architectures
  - Kernels/BLAS
  - More GPU-only factorizations
  - Optimize multi-GPU algorithms
- User requests, e.g., inertia computations, LDLT factorizations
- Tensor contractions, e.g., for high-order FEM solvers
- Data analytics & AI
  - Kernels (GEMMs, SVD, batched routines, convolutions, FFT, FFT variants, etc.)
  - DNN framework
  - Users calling MAGMA through PyTorch, etc.
- Porting to other programming models
  - Intel GPUs (using MKL oneAPI & DPC++)