MAGMA: Matrix Algebra on GPU and Multicore Architectures
MAGMA: LAPACK for GPUs

• MAGMA
  – Matrix algebra for GPU and multicore architecture
  – To provide LAPACK/ScaLAPACK on hybrid architectures
  – http://icl.cs.utk.edu/magma/

• MAGMA BLAS
  – A subset of BLAS for GPUs, highly optimized for NVIDIA GPGPUs
  – Fast GEMM for Fermi

• MAGMA developers & collaborators
  – UTK, UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  – Community effort, similarly to LAPACK/ScaLAPACK
A New Generation of DLA Software

Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Software/Algorithms</th>
<th>(70’s)</th>
<th>(80’s)</th>
<th>(90’s)</th>
<th>(00’s)</th>
<th>(00’s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (Vector operations)</td>
<td>Level-1</td>
<td>Level-3</td>
<td>PBLAS</td>
<td>hybrid</td>
<td>hybrid</td>
</tr>
<tr>
<td>LAPACK (Blocking, cache friendly)</td>
<td>Level-3</td>
<td>Level-3</td>
<td>PBLAS</td>
<td>hybrid</td>
<td>hybrid</td>
</tr>
<tr>
<td>ScaLAPACK (Distributed Memory)</td>
<td>Level-3</td>
<td>Level-3</td>
<td>PBLAS</td>
<td>hybrid</td>
<td>hybrid</td>
</tr>
<tr>
<td>PLASMA (New Algorithms)</td>
<td>Level-3</td>
<td>Level-3</td>
<td>PBLAS</td>
<td>hybrid</td>
<td>hybrid</td>
</tr>
<tr>
<td>MAGMA Hybrid Algorithms</td>
<td>Level-3</td>
<td>Level-3</td>
<td>PBLAS</td>
<td>hybrid</td>
<td>hybrid</td>
</tr>
</tbody>
</table>

MAGMA
Hybrid Algorithms
(heterogeneity friendly)
MAGMA Software Stack

Linux, Windows, Mac OS X  |  C/C++, Fortran  |  Matlab, Python
MAGMA Functionality

• **80+ hybrid algorithms** have been developed (total of 320+ routines)
  – Every algorithm is in 4 precisions (s/c/d/z)
  – There are 3 mixed precision algorithms (zc & ds)
  – These are hybrid algorithms, expressed in terms of BLAS

• **MAGMA BLAS**
  – A subset of GPU BLAS, optimized for Tesla and Fermi GPUs

<table>
<thead>
<tr>
<th>MAGMA 1.1 ROUTINES &amp; FUNCTIONALITIES</th>
<th>SINGLE GPU</th>
<th>MULTI-GPU STATIC</th>
<th>MULTI-GPU DYNAMIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sided Factorizations (LU, QR, Cholesky)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Linear System Solvers</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Linear Least Squares (LLS) Solvers</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Matrix Inversion</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Singular Value Problem (SVP)</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Non-symmetric Eigenvalue Problem</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Symmetric Eigenvalue Problem</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Generalized Symmetric Eigenvalue Problem</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

**SINGLE GPU**
- Hybrid LAPACK algorithms with static scheduling and LAPACK data layout

**MULTI-GPU STATIC**
- Hybrid LAPACK algorithms with 1D block cyclic static scheduling and LAPACK data layout

**MULTI-GPU DYNAMIC**
- Tile algorithms with StarPU scheduling and tile matrix layout
Methodology overview

A methodology to use all available resources:

- **MAGMA** uses **hybridization** methodology based on
  - Representing linear algebra algorithms as collections of **tasks** and **data dependencies** among them
  - Properly **scheduling** tasks’ execution over multicore and GPU hardware components

- Successfully applied to fundamental linear algebra algorithms
  - One- and two-sided factorizations and solvers
  - Iterative linear and eigensolvers

- **Productivity**
  - 1) High level; 2) Leveraging prior developments; 3) Exceeding in performance homogeneous solutions
Hybrid Algorithms

One-Sided Factorizations (LU, QR, and Cholesky)

• Hybridization
  – Panels (Level 2 BLAS) are factored on CPU using LAPACK
  – Trailing matrix updates (Level 3 BLAS) are done on the GPU using “look-ahead”
A Hybrid Algorithm Example

• Left-looking hybrid Cholesky factorization in MAGMA

```c
for (j=0; j<n; j += nb) {
    jb = min(nb, n - j);
    magma_zherk( MagmaUpper, MagmaConjTrans,
                 jb, j, m_one, dA(0, j), ldda, one, dA(j, j), ldda, queue );
    magma_zgetmatrix_async( jb, jb, dA(j,j), ldda, work, 0, jb, queue, &event );
    if (j+jb < n)
    {
      magma_zgemm( MagmaConjTrans, MagmaNoTrans, jb, n-j-jb, j, mz_one,
                    dA(0, j), ldda, dA(0, j+jb), ldda, z_one, dA(j, j+jb), ldda, queue );
      magma_event_sync( event );
    }
    lapackf77_zpotrf( MagmaUpperStr, &jb, work, &jb, info );
    if (*info != 0)
    {
      *info += j;
    }
    magma_zsetmatrix_async( jb, jb, work, 0, jb, dA(j,j), ldda, queue, &event );
    if (j+jb < n)
    {
      magma_event_sync( event );
      magma_ztrsm( MagmaLeft, MagmaUpper, MagmaConjTrans, MagmaNonUnit,
                   jb, n-j-jb, z_one, dA(j, j), ldda, dA(j, j+jb), ldda, queue );
    }
}
```

• The difference with LAPACK – the 4 additional lines in red

• Line 8 (done on CPU) is overlapped with work on the GPU (from line 6)
LU Factorization (single GPU)

**Graph: MAGMA LU in double precision on single GPU (C2050)**

- **GPU MAGMA**
  - 1,090 MFlop/W *
  - Performance increases with matrix size.

- **CPU LAPACK**
  - 55 MFlop/W *
  - Performance also increases with matrix size.

**Specifications:**

**GPU**
- Fermi C2050 (448 CUDA Cores @ 1.15 GHz)
- Intel Q9300 (4 cores @ 2.50 GHz)
- DP peak: 515 GFlop/s
- Power: ~220 W

**CPU**
- AMD Istanbul (8 sockets x 6 cores (48 cores) @ 2.8GHz)
- DP peak: 538 GFlop/s
- Power: ~1,022 W

* Computation consumed power rate (total system rate minus idle rate), measured with KILL A WATT PS, Model P430.
From Single to MultiGPU Support

- Data distribution
  - 1-D block-cyclic distribution

- Algorithm
  - GPU holding current panel is sending it to CPU
  - All updates are done in parallel on the GPUs
  - Look-ahead is done with GPU holding the next panel
LU Factorization (multiGPUs)

Keeneland system, using one node
3 NVIDIA GPUs (M2070 @ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)
LU Factorization (multiGPUs)

MAGMA LU in double precision on multi-GPUs (Fermi C2070)

Keeneland system, using one node
3 NVIDIA GPUs (M2070 @ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)
Out of GPU Memory Algorithms

- Perform left-looking factorizations on sub-matrices that fit in the GPU memory (using existing algorithms)
- The rest of the matrix stays on the CPU
- Left-looking versions minimize writing on the CPU

1) Copy $A_2$ to the GPU
2) Update $A_2$ using $A_1$ (a panel of $A_1$ at a time)
3) Factor the updated $A_2$ using existing hybrid code
4) Copy factored $A_2$ to the CPU

Trivially extended to multiGPUs:
- $A_2$ is “larger” with 1-D block cyclic distribution, again reusing existing algorithms
Hybrid Algorithms

Two-Sided Factorizations (to bidiagonal, tridiagonal, and upper Hessenberg forms) for eigen- and singular-value problems

• Hybridization
  - Trailing matrix updates (Level 3 BLAS) are done on the GPU (similar to the one-sided factorizations)
  - Panels (Level 2 BLAS) are hybrid
    - operations with memory footprint restricted to the panel are done on CPU
    - The time consuming matrix-vector products involving the entire trailing matrix are done on the GPU
MultiGPU Two-Sided Factorizations

- Need HP multiGPU Level 2 BLAS (e.g., 50% of flops in the tridiagonal reduction)

Performance of DSYMV on multi M2090s

Hybrid Two-Sided Factorizations

Task Splitting & Task Scheduling

![Diagram showing task splitting and scheduling]

- Task scheduling:
  - Multicore+GPU
  - GPU
  - Multicore

![Diagram showing work and dWork]

- 1. Copy dP to CPU
- 2. Copy y to GPU
- 3. Copy y to CPU
- 4. Copy to CPU

![Diagram showing critical path]

Critical path:
- P0
- M0
- G0

- CPU
- GPU
- Multicore

Innovative Computing Laboratory
Department of Electrical Engineering and Computer Science
From Fast BLAS to Fast Tridiagonalization

- 50 % of the flops are in SYMV
- Memory bound, i.e. does not scale well on multicore CPUs
- Use the GPU’s high memory bandwidth and optimized SYMV
- 8 x speedup over 12 Intel cores (X5660 @2.8 GHz)

Keeneland system, using one node
3 NVIDIA GPUs (M2070@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

An Additional 4x Speedup!

- 12 x speedup over 12 Intel cores (X5660 @2.8 GHz)
- A two-stage approach leading to increased computational intensity

Keeneland system, using one node
3 NVIDIA GPUs (M2070@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

From Static to Dynamic Scheduling ...

- Static may stall in situations where work is available
- Hand tuned optimizations
- Hardware heterogeneity
- Kernel heterogeneity
- Separation of concerns
- Dynamic Runtime System
Matrices Over Runtime Systems at Exascale (MORSE)

- Mission statement: "Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems”
- Runtime challenges due to the ever growing hardware complexity
- Algorithmic challenges to exploit the hardware capabilities at most
- Integrated into MAGMA software stack
MAGMA-MORSE: x86 + multiGPUs

- Lessons Learned from PLASMA!
- New high performance numerical kernels
- StarPU Runtime System (Augonnet et. Al, INRIA, Bordeaux)
- Both: x86 and GPUs => Hybrid Computations
- Similar to LAPACK in functionality
High Productivity

From Sequential Nested-Loop Code to Parallel Execution

for (k = 0; k < min(MT, NT); k++){
  
zgeqrt(A[k;k], ...);
  
  for (n = k+1; n < NT; n++)
    
zunmqr(A[k;k], A[k;n], ...);
  
  for (m = k+1; m < MT; m++)
    
ztsqrt(A[k;k], A[m;k], ...);
  
  for (n = k+1; n < NT; n++)
    
ztsmqr(A[m;k], A[k;n], A[m;n], ...);

}
High Productivity

From Sequential Nested-Loop Code to Parallel Execution

for (k = 0; k < min(MT, NT); k++){
    starpu_Insert_Task(&cl_zgeqrt, k, k, ...);
    for (n = k+1; n < NT; n++)
        starpu_Insert_Task(&cl_zunmqr, k, n, ...);
    for (m = k+1; m < MT; m++)
        starpu_Insert_Task(&cl_ztsqrt, m, k, ...);
    for (n = k+1; n < NT; n++)
        starpu_Insert_Task(&cl_ztsmqr, m, n, k, ...);
}
}
Collaborators and Support

**MAGMA team**
http://icl.cs.utk.edu/magma

**PLASMA team**
http://icl.cs.utk.edu/plasma

**Collaborating partners**
University of Tennessee, Knoxville
University of California, Berkeley
University of Colorado, Denver
INRIA, France (StarPU team)
KAUST, Saudi Arabia