MAGMA 1.0 – LAPACK for GPUs

Stan Tomov

ICL Lunch Talk
January 07, 2011
Outline

• Project motivation
  • Enable new architectures for efficient LA
  • Challenges of emerging architectures

• MAGMA 1.0
  • Outline
  • Results

• Current & future work directions

• Conclusions
Hardware Trends

- Power consumption and the move towards multicore
- Hybrid architectures
- GPU
- Hybrid GPU-based systems
  - CPU and GPU to get integrated (NVIDIA to make ARM CPU cores alongside GPUs)

![Graph showing hardware trends over time](image)

**x86 host**

**host memory**

**DMA**

**7.5 GB/s**

**PCI-e 3.0**

Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanović

Slide from Kathy Yelick
Challenges to software

- **Increase in parallelism**
  Tesla C2050 (Fermi): 448 CUDA cores @1.15 GHz
  SP peak is 1075 GFlop/s, DP peak is 515 Gflop/s

- **Increase in communication cost [vs computation]**
  Processor speed improves ~59% / year
  memory bandwidth by only 23%

- **Heterogeneity**
Matrix Algebra on GPU and Multicore Architectures (MAGMA)

- **MAGMA**: a new generation linear algebra (LA) libraries to achieve the fastest possible time to an accurate solution on hybrid/heterogeneous architectures
  
  **Homepage**: [http://icl.cs.utk.edu/magma/](http://icl.cs.utk.edu/magma/)

- **MAGMA & LAPACK**
  - **MAGMA** uses LAPACK (on the CPUs) and extends its functionality to hybrid systems (GPU support);
  - **MAGMA** is designed to be similar to LAPACK in functionality, data storage and interface
  - **MAGMA** leverages years of experience in developing open source LA software packages like LAPACK, ScaLAPACK, BLAS, ATLAS, and PLASMA

- **Support**
  - NSF, DOE
  - Microsoft, MathWorks
  - NVIDIA [CUDA Center of Excellence at UTK on the development of Linear Algebra Libraries for CUDA-based Hybrid Architectures]

- **MAGMA developers/collaborators**
  - University of Tennessee, Knoxville; University of California, Berkeley; University of Colorado, Denver
  - INRIA Bordeaux - Sud Ouest, France; University of Coimbra, Portugal
MAGMA 1.0

• 32 routines are developed (next)
  ▶ Every routines is in 4 precisions (s/c/d/z, denoted by X)
  ▶ There are 3 mixed precision routines (zc and ds, denoted by XX)
  ▶ These are **hybrid algorithms**, extending the sequential **LAPACK algorithms** for hybrid systems (multicore + GPUs)

• Support is for single CUDA-enabled NVIDIA GPU, either Tesla or Fermi

• MAGMA BLAS
  ▶ A subset of GPU BLAS, highly optimized for Tesla and Fermi GPUs
MAGMA 1.0

One-sided factorizations

<table>
<thead>
<tr>
<th></th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Xgetrf</td>
<td>LU factorization; CPU interface</td>
</tr>
<tr>
<td>2</td>
<td>Xgetrf_gpu</td>
<td>LU factorization; GPU interface</td>
</tr>
<tr>
<td>3</td>
<td>Xgetrf_mc</td>
<td>LU factorization on multicore (no GPUs)</td>
</tr>
<tr>
<td>4</td>
<td>Xpotrf</td>
<td>Cholesky factorization; CPU interface</td>
</tr>
<tr>
<td>5</td>
<td>Xpotrf_gpu</td>
<td>Cholesky factorization; GPU interface</td>
</tr>
<tr>
<td>6</td>
<td>Xpotrf_mc</td>
<td>Cholesky factorization on multicore (no GPUs)</td>
</tr>
<tr>
<td>7</td>
<td>Xgeqrf</td>
<td>QR factorization; CPU interface</td>
</tr>
<tr>
<td>8</td>
<td>Xgeqrf_gpu</td>
<td>QR factorization; GPU interface; with T matrices stored</td>
</tr>
<tr>
<td>9</td>
<td>Xgeqrf2_gpu</td>
<td>QR factorization; GPU interface; without T matrices</td>
</tr>
<tr>
<td>10</td>
<td>Xgeqrf_mc</td>
<td>QR factorization on multicore (no GPUs)</td>
</tr>
<tr>
<td>11</td>
<td>Xgeqrf2</td>
<td>QR factorization; CPU interface</td>
</tr>
<tr>
<td>12</td>
<td>Xgeqlf</td>
<td>QL factorization; CPU interface</td>
</tr>
<tr>
<td>13</td>
<td>Xgelqf</td>
<td>LQ factorization; CPU interface</td>
</tr>
</tbody>
</table>
## MAGMA 1.0

### Linear solvers

<table>
<thead>
<tr>
<th></th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Xgetrs_gpu</td>
<td>Work precision; using LU factorization; GPU interface</td>
</tr>
<tr>
<td>15</td>
<td>Xpotrs_gpu</td>
<td>Work precision; using Cholesky factorization; GPU interface</td>
</tr>
<tr>
<td>16</td>
<td>Xgels_gpu</td>
<td>Work precision LS; GPU interface</td>
</tr>
<tr>
<td>17</td>
<td>XXgetrs_gpu</td>
<td>Mixed precision iterative refinement solver; Using LU factorization; GPU interface</td>
</tr>
<tr>
<td>18</td>
<td>XXpotrs_gpu</td>
<td>Mixed precision iterative refinement solver; Using Cholesky factorization; GPU interface</td>
</tr>
<tr>
<td>19</td>
<td>XXgeqrsv_gpu</td>
<td>Mixed precision iterative refinement solver; Using QR on square matrix; GPU interface</td>
</tr>
</tbody>
</table>
## MAGMA 1.0

### Two-sided factorizations

<table>
<thead>
<tr>
<th>20. Xgehrd</th>
<th>Reduction to upper Hessenberg form; with T matrices stored; CPU interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>21. Xgehrd2</td>
<td>Reduction to upper Hessenberg form; Without the T matrices stored; CPU interface</td>
</tr>
<tr>
<td>22. Xhetrd</td>
<td>Reduction to tridiagonal form; CPU interface</td>
</tr>
<tr>
<td>23. Xgebrd</td>
<td>Reduction to bidiagonal form; CPU interface</td>
</tr>
</tbody>
</table>
# MAGMA 1.0

Generating/applying orthogonal matrices

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.</td>
<td>Xungqr</td>
</tr>
<tr>
<td>25.</td>
<td>Xungqr_gpu</td>
</tr>
<tr>
<td>26.</td>
<td>Xunmtr</td>
</tr>
<tr>
<td>27.</td>
<td>Xunmqr</td>
</tr>
<tr>
<td>28.</td>
<td>Xunmqr_gpu</td>
</tr>
<tr>
<td>29.</td>
<td>Xunghr</td>
</tr>
</tbody>
</table>
**MAGMA 1.0**

Eigen/singular-value solvers

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30. Xgeev</td>
<td>Solves the non-symmetric eigenvalue problem; CPU interface</td>
</tr>
<tr>
<td>31. Xheevd</td>
<td>Solves the Hermitian eigenvalue problem; Uses divide and conquer; CPU interface</td>
</tr>
<tr>
<td>32. Xgesvd</td>
<td>SVD; CPU interface</td>
</tr>
</tbody>
</table>

- Currently, these routines have GPU-acceleration for the
  - two-sided factorizations used and the
  - Orthogonal transformation related to them (matrix generation/application from slide 9)
# MAGMA BLAS

## Level 2 BLAS

1. **Xgemv_terraform**  
   General matrix-vector product for Tesla

2. **Xgemv_fermi**  
   General matrix-vector product for Fermi

3. **Xsymv_terraform**  
   Symmetric matrix-vector product for Tesla

4. **Xsymv_fermi**  
   Symmetric matrix-vector product for Fermi
### MAGMA BLAS

#### Level 3 BLAS

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.</td>
<td>Xgemm_tesla</td>
</tr>
<tr>
<td>6.</td>
<td>Xgemm_fermi</td>
</tr>
<tr>
<td>7.</td>
<td>Xtrsm_tesla</td>
</tr>
<tr>
<td>8.</td>
<td>Xtrsm_fermi</td>
</tr>
<tr>
<td>9.</td>
<td>Xsyrk_tesla</td>
</tr>
<tr>
<td>10.</td>
<td>Xsyr2k_tesla</td>
</tr>
</tbody>
</table>

- Our GEMMs for Fermi are used in CUBLAS 3.2
### MAGMA BLAS

#### Other routines

<table>
<thead>
<tr>
<th></th>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.</td>
<td>Xswap</td>
<td>LU factorization; CPU interface</td>
</tr>
<tr>
<td>12.</td>
<td>Xlacpy</td>
<td>LU factorization; GPU interface</td>
</tr>
<tr>
<td>13.</td>
<td>Xlange</td>
<td>LU factorization on multicore (no GPUs)</td>
</tr>
<tr>
<td>14.</td>
<td>Xlanhe</td>
<td>Cholesky factorization; CPU interface</td>
</tr>
<tr>
<td>15.</td>
<td>Xtranspose</td>
<td>Cholesky factorization; GPU interface</td>
</tr>
<tr>
<td>16.</td>
<td>Xinplace_transpose</td>
<td>Cholesky factorization on multicore (no GPUs)</td>
</tr>
<tr>
<td>17.</td>
<td>Xpermute</td>
<td>QR factorization; CPU interface</td>
</tr>
<tr>
<td>18.</td>
<td>Xauxiliary</td>
<td>QR factorization; GPU interface; with T matrices stored</td>
</tr>
</tbody>
</table>
Results – one sided factorizations

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

- Similar results for Cholesky & QR
- Fast solvers (several innovations)
  - in working precision, and
  - mixed-precision iter. refinement based on the one-sided factor.
Performance of MAGMA Cholesky on Fermi (C2050)

- GFlop/s
- Matrix size
- CP
- SP
- ZP
- DP
Results – linear solvers

MAGMA LU-based solvers on Fermi (C2050)

- Single Prec
- Double Prec
- Iter Ref

Fermi
Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 GFlop/s
**Results – two sided factorizations**

Hessenberg Factorization in double precision
[ for the general eigenvalue problem ]

- Similar accelerations for the bidiagonal factorization [for SVD] & tridiagonal factorization [for the symmetric eigenvalue problem]

- Similar acceleration (exceeding 10x) compared to other top-of-the-line multicore systems (including Nehalem-based) and libraries (including MKL, ACML)

---

**FERMI**

- Tesla C2050: 448 CUDA cores @ 1.15GHz
- SP/DP peak is 1030 / 515 Gflop/s
- [ system cost ~ $3,000 ]

**ISTANBUL**

- AMD 8 socket 6 core (48 cores) @2.8GHz
- SP/DP peak is 1075 / 538 Gflop/s
- [ system cost ~ $30,000 ]

---

**Graph Details**

- **GFlop/s** vs. **Matrix Size**
- **FERMI MAGMA**
- **LAPACK + GOTO BLAS**

---

21/29
**Results - BLAS**

**SGEMM**

- **MAGMA**
- **M3 CGEMM**
- **MAGMA**
- **CUBLAS 3.1**

**DGEMM**

- **MAGMA**
- **M3 ZGEMM**
- **MAGMA**
- **CUBLAS 3.1**

Tesla C2050 (Fermi): 448 CUDA cores @ 1.15GHz, theoretical SP peak is 1.03 Tflop/s, DP peak 515 GFlop/s)
Performance of MAGMA xUNGQR on Fermi (C2050)

- **GFlop/s** vs. **Matrix size**

- Lines represent different data types:
  - **CP**
  - **SP**
  - **ZP**
  - **DP**

- Legend indicates the color and symbol for each data type.
MAGMA Software Stack

- **CPU**
  - Tile & LAPACK Algorithms with DAGuE

- **HYBRID**
  - MAGNUM / Rectangular / PLASMA Tile Algorithms
  - PLASMA / Quark
  - LAPACK Algorithms and Tile Kernels
  - MAGMA 0.2
  - MAGMA SPARSE
  - MAGMA BLAS

- **GPU**
  - StarPU
  - BLAS
  - CUDA

**Operating Systems:**
- Linux, Windows, Mac OS X
- C/C++, Fortran
- Matlab, Python
Current and future work

- Software engineering, improvements, and bug fixes
- MAGMA BLAS for Fermi
- Tuning
- Portability to other hardware
  - Through OpenCL and auto-tuning
- Algorithms for multiGPU and multicore
  - Various Hybrid Tile algorithms
  - Hybrid LAPACK algorithms
  - Distributed Hybrid Tile and LAPACK algorithms
Magnum tile algorithms with StarPU

- We have finished the LU, QR, and Cholesky factorizations on a node (with Hatem, Mathieu and the StarPU team including Cedric and Emmanuel)
Research on Portability Across Platforms through OpenCL

- Performance portability of OpenCL implementations – through auto-tuning
  - Collecting best kernel versions
  - Generating multiple kernel versions to explore the kernel parameter space
  - Find best performing kernel versions on particular architecture using empirical-based search enhanced with heuristic models
Conclusions

- **Linear and eigenvalue solvers can be significantly accelerated on systems of multicore and GPU architectures**

- Many-core architectures with accelerators (e.g., GPUs) are the future of high performance scientific computing

- Challenge: Fundamental libraries will need to be redesigned/rewritten to take advantage of the emerging many-core architectures