MAGMA MIC: Linear Algebra Library for Intel Xeon Phi Coprocessors

Jack Dongarra w/ M. Gates, Y. Jia, K. Kabir, P. Luszczek, and S. Tomov
University of Tennessee, Knoxville

11 / 13 / 2012
MAGMA: LAPACK for hybrid systems

• MAGMA
  • A new generation of HP Linear Algebra Libraries
  • To provide LAPACK/ScaLAPACK on hybrid architectures

• MAGMA MIC 0.3
  • For hybrid, shared memory systems featuring Intel Xeon Phi coprocessors
  • Included are one-sided factorizations
  • Open Source Software ([http://icl.cs.utk.edu/magma](http://icl.cs.utk.edu/magma))

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

Software/Algorithms follow hardware evolution in time:

<table>
<thead>
<tr>
<th>Software/Algorithm</th>
<th>Hardware Evolution</th>
<th>Rely on</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (70’s)</td>
<td>(Vector operations)</td>
<td>- Level-1 BLAS operations</td>
</tr>
<tr>
<td>LAPACK (80’s)</td>
<td>(Blocking, cache friendly)</td>
<td>- Level-3 BLAS operations</td>
</tr>
<tr>
<td>ScaLAPACK (90’s)</td>
<td>(Distributed Memory)</td>
<td>- PBLAS Mess Passing</td>
</tr>
<tr>
<td>PLASMA (00’s)</td>
<td>New Algorithms (many-core friendly)</td>
<td>- a DAG/scheduler - block data layout - some extra kernels</td>
</tr>
<tr>
<td><strong>MAGMA</strong></td>
<td></td>
<td>- hybrid scheduler - hybrid kernels</td>
</tr>
</tbody>
</table>
MAGMA Software Stack

Support: 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 MIC provides support for Intel Xeon Phi Coprocessors
Methodology overview

A methodology to use all available resources:

- MAGMA MIC uses **hybridization** methodology based on
  - Representing linear algebra algorithms as collections of **tasks** and **data dependencies** among them
  - Properly **scheduling** tasks' execution over multicore CPUs and manycore coprocessors

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

- **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 MIC using “look-ahead”
Programming LA on Hybrid Systems

- Algorithms expressed in terms of BLAS
- Use vendor-optimized BLAS
- Algorithms expressed in terms of tasks
- Use some scheduling/run-time system
Intel Xeon Phi specific considerations

• Intel Xeon Phi coprocessors (vs GPUs) are less dependent on host
  [can login on the coprocessor, develop, and run programs in native mode]

• There is no high-level API similar to CUDA/OpenCL facilitating Intel Xeon Phi’s use from the host *

• There is pragma API but it may be too high-level for HP numerical libraries

• We used Intel Xeon Phi’s Low Level API (LLAPI) to develop MAGMA API
  [allows us to uniformly handle hybrid systems]

* OpenCL 1.2 support is available for Intel Xeon Phi as of Intel SDK XE 2013 Beta
Intel Xeon Phi acts as coprocessor

On the Intel Xeon Phi, MAGMA runs a “server”

Communications are implemented using LLAPI
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 MIC (from line 6)
MAGMA MIC programming model

Host program

```
for ( j=0; j<n; j += nb ) {
    jb = min(nb, n - j);
    magma_zherk( MagmaUpper, MagmaConjTrans,
                 j, j, m_one, dA(0, j), ldda, one, dA(j, j), ldda, queue);
    if ( j+jb < n )
        magma_zgemm( MagmaConjTrans, MagmaNoTrans, jb, n-j+jb, dA(0, j), ldda, dA(0, j+jb), ldda, z_one, dA(j, j), 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, MagmaNoTrans,
                 jb, n-j-jb, z_one, dA(j, j), ldda, dA(j, j+jb), ldda, queue);
}
```

Intel Xeon Phi interface

Send asynchronous requests to the MIC; Queued & Executed on the MIC.
MAGMA MIC Performance (QR)

- **Host**: Sandy Bridge (2 x 8 @ 2.6 GHz)
  - DP Peak: 332 GFlop/s
- **Coprocessor**: Intel Xeon Phi (60 @ 1.09 GHz)
  - DP Peak: 1046 GFlop/s
- **System DP Peak**: 1378 GFlop/s
- **MPSS**: 2.1.4346-16
- **Compiler**: xe_2013.1.117

Matrix size vs. GFlop/s graph showing performance comparison between MAGMA DGEQRF and MKL DGEQRF.
MAGMA MIC Performance (Cholesky)

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
Intel Xeon Phi (60 @ 1.09 GHz)
DP Peak 1046 GFlop/s

System DP Peak 1378 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117

Matrix size

GFlop/s

960 3072 4992 7104 9024 12596 16064 20160 22596 24364 26604
MAGMA MIC Performance (LU)

Matrix size

GFlop/s

MAGMA DGETRF
MKL DGETRF

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
Intel Xeon Phi (60 @ 1.09 GHz)
DP Peak 1046 GFlop/s

System DP Peak 1378 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117
MAGMA MIC Performance

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
Intel Xeon Phi (60 @ 1.09 GHz)
DP Peak 1046 GFlop/s

System DP Peak 1378 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117
From Single to MultiMIC Support

• Data distribution
  - 1-D block-cyclic distribution

• Algorithm
  - MIC holding current panel is sending it to CPU
  - All updates are done in parallel on the MICs
  - Look-ahead is done with MIC holding the next panel

![Diagram of MICs and nb]
MAGMA MIC Scalability
LU Factorization Performance in DP

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
4 Intel Xeon Phi (60 @1.09 GHz)
DP Peak 4 x 1046 GFlop/s

System DP Peak 4516 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117
MAGMA MIC Scalability
LU Factorization Performance in DP

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
4 Intel Xeon Phi (60 @1.09 GHz)
DP Peak 4 x 1046 GFlop/s

System DP Peak 4516 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117

Matrix size
GFlop/s
0 200 400 600 800 1000 1200 1400 1600 1800 2000 2200
8064 16064 24364 31784 37800

2 MICs
1 MIC
MAGMA MIC Scalability
LU Factorization Performance in DP

Matrix size

GFlop/s

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
4 Intel Xeon Phi (60 @1.09 GHz)
DP Peak 4 x 1046 GFlop/s

System DP Peak 4516 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117

3 MICs
2 MICs
1 MIC

8064 16064 24364 31784 37800
MAGMA MIC Scalability
LU Factorization Performance in DP

Matrix size

GFlop/s

Host
Sandy Bridge (2 x 8 @2.6 GHz)
DP Peak 332 GFlop/s

Coprocessor
4 Intel Xeon Phi (60 @1.09 GHz)
DP Peak 4 x 1046 GFlop/s

System DP Peak 4516 GFlop/s
MPSS 2.1.4346-16
compiler_xe_2013.1.117
Current and Future MIC-specific directions

- Explore the use of tasks of lower granularity on MIC
  [ GPU tasks in general are large, data parallel]
- Reduce synchronizations
  [ less fork-join synchronizations ]
- Scheduling of less parallel tasks on MIC
  [ on GPUs, these tasks are typically offloaded to CPUs]
  [ to reduce CPU-MIC communications ]
- Develop more algorithms,
  porting newest developments in LA,
  while discovering further MIC-specific optimizations
Current and Future MIC-specific directions

• Synchronization avoiding algorithms using Dynamic Runtime Systems

48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000, tile size is 200 x 200
Current and Future MIC-specific directions

High-productivity w/ Dynamic Runtime Systems
From Sequential Nested-Loop Code to Parallel Execution

```c
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], ...);
    }
}
```
Current and Future MIC-specific directions

High-productivity w/ Dynamic Runtime Systems
From Sequential Nested-Loop Code to Parallel Execution

```c
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, ...);
    }
}
```
Contact

Jack Dongarra
dongarra@eecs.utk.edu
Innovative Computing Laboratory
University of Tennessee, Knoxville

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