MAGMA MIC
Optimizing Linear Algebra for Intel Xeon Phi

Stan Tomov

Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

ISC’15, Frankfurt, Germany
Intel Booth Presentation
July 13, 2015
LAPACK and ScaLAPACK

- Standard dense linear algebra (DLA) libraries
- Many applications rely on DLA
- Designed in 80/90’s for cache-based architectures

Must be redesigned for modern heterogeneous systems with multi/many-core CPUs and coprocessors.
IPCC at ICL

• Develop
  – Next generation LAPACK / ScaLAPACK
  – Programming models, and
  – Technologies
for heterogeneous
Intel Xeon Phi-based platforms

• Disseminate developments
through the MAGMA MIC library

• High value proposition
MAGMA MIC enables ease of use and adoption of Intel Xeon Phi architectures
in applications as linear algebra is fundamental to scientific computing
## A New Generation of Dense Linear Algebra Libraries

Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th><strong>Software/Algorithm</strong></th>
<th><strong>(Evolutionary Period)</strong></th>
<th><strong>Rely on</strong></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</td>
<td>- a DAG/scheduler</td>
</tr>
<tr>
<td></td>
<td>(many-core friendly)</td>
<td>- block data layout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- some extra kernels</td>
</tr>
<tr>
<td><strong>MAGMA</strong></td>
<td>Hybrid Algorithms</td>
<td>- hybrid scheduler</td>
</tr>
<tr>
<td></td>
<td>(heterogeneity friendly)</td>
<td>- hybrid kernels</td>
</tr>
</tbody>
</table>
MAGMA MIC
LAPACK for heterogeneous systems

• MAGMA MIC
  – Project on the development of a new generation of HP Linear Algebra Libraries
  – To provide LAPACK/ScaLAPACK on heterogeneous Intel Xeon Phi-based systems
  – Well established project with product disseminated through the MAGMA MIC libraries:

<table>
<thead>
<tr>
<th>MAGMA MIC</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>(2012-11-13)</td>
</tr>
<tr>
<td>1.0</td>
<td>(2013-05-03)</td>
</tr>
<tr>
<td>1.1</td>
<td>(2014-01-07)</td>
</tr>
<tr>
<td>1.2</td>
<td>(2014-09-17)</td>
</tr>
<tr>
<td>1.3</td>
<td>(2014-11-15)</td>
</tr>
<tr>
<td>1.4</td>
<td>(2015-07-12)</td>
</tr>
</tbody>
</table>
  • For heterogeneous, shared memory systems
  • Included are the main factorizations, linear system and eigen-problem solvers
  • Open Source Software ( [http://icl.cs.utk.edu/magma](http://icl.cs.utk.edu/magma) )

• Collaborators
  – Intel MKL Team
  – UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  – Community effort, similar to LAPACK/ScaLAPACK
Key Features of MAGMA MIC

HYBRID ALGORITHMS

MAGMA MIC uses hybrid algorithms where the computation is split into tasks of varying granularity and their execution scheduled over the hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level 3 BLAS, are scheduled on the MICs.

PERFORMANCE & ENERGY EFFICIENCY

MAGMA MIC on KNC
LU factorization in double precision arithmetic

<table>
<thead>
<tr>
<th>MIC</th>
<th>KNC 7120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>60 cores @ 1.23 GHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Xeon ES-2670 (Sandy Bridge)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2x6 cores @ 2.60 GHz</td>
</tr>
</tbody>
</table>

- 3 GFlop / Watt
- 3 x faster for 3 x less energy

FEATURES AND SUPPORT

MAGMA MIC 1.4

- Linear system solvers
- Eigen-problem solvers
- SVD
- CPU/AO interface
- MIC/Native interface
- Multiple precision support
- Mixed-precision iter. refinement solvers
- Multicore and multi-MIC support
- Sparse LA
- LAPACK testing
- Linux
A methodology to use all available resources:

- **MAGMA MIC** uses **hybrid algorithms**
  - 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 eigensolvers

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

Left-looking hybrid Cholesky

to parallel hybrid

```c
for (j=0, j<n; j+=nb) {
    jb = min(nb, n-j);
    magma_zherk( MagmaUpper, MagmaConjTrans, jb, j, one, dA(0,j), ldda, one, dA(j,j), ldda);
    magma_zgetmatrix_async( jb, jb, dA(j,j), ldda, work, jb, queue, &event);
    if (j+jb < n) {
        magma_zgemm( MagmaConjTrans, MagmaNoTrans, jb, n-j-jb, one, dA(0,j), ldda, dA(0,j+jb), ldda, one, dA(j,j+jb), ldda);
        magma_event_sync( event );
    }
}
```

MAGMA runtime environment

- Scheduling can be static or dynamic
- Dynamic is based on QUARK
- Uses MAGMA streams to offload computation to the MIC

Note:
- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences can be hidden in a dynamic scheduler making the top level representation of MAGMA MIC algorithms almost identical to LAPACK
A Hybrid Algorithm Example

Left-looking hybrid Cholesky

to parallel hybrid

MAGMA

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

Note:
- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences can be hidden in a dynamic scheduler making the top level representation of MAGMA MIC algorithms almost identical to LAPACK
Programming models

• We developed two APIs for offloading work to MIC:

  ![Diagram showing Host and Intel Xeon Phi connected through LLAPI or pragma based on PCIe]

  Both APIs have the same interface and abstract low level programming details

  **LLAPI based**
  - A server runs on the MIC
  - Communications are implemented through LLAPI using SCIF

  **Compiler pragma offload based**
  - API is using Phi-specific offload directives
  - Enhancements for CPU-MIC communications
Scheduling strategies

High-productivity with Dynamic Runtime Systems

From sequential code

```
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], ...);
    }
}
```

Parallel execution

```
for (k = 0; k < min(MT, NT); k++){
    Insert_Task(&zgeqrt, k, k, ...);
    for (n = k+1; n < NT; n++)
        Insert_Task(&zunmqr, k, n, ...);
    for (m = k+1; m < MT; m++){
        Insert_Task(&ztsqrt, m, k, ...);
        for (n = k+1; n < NT; n++)
            Insert_Task(&ztsmqr, m, n, k, ...);
    }
}
```

No need to explicitly code data dependencies and data transfers. This is hidden in the runtime system.
Performance on single MIC QR AO with static and dynamic MAGMA

Algorithms are scalable using all available hardware, e.g. CPU cores

- 71% of KNC peak
- 90% of dgemm peak

Host
Ivytown (2 x 12 @2.7 GHz)
DP Peak 518 GFlop/s

Coprocessor
Intel Xeon Phi (60 @ 1.23 GHz)
DP Peak 1180 GFlop/s
Scalability on multiple MICs

MAGMA DGETRF Performance (Multiple Card)

- 4 MIC
- 3 MIC
- 2 MIC
- 1 MIC

Performance scales well in spite of PCI's bandwidth limitations

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

76% of peak
MAGMA MIC Sparse

FEATURES AND SUPPORT in MAGMA MIC 1.4

- ROUTINES: CG, GMRES, BiCGSTAB, Iterative Refinement
- PRECONDITIONERS: Jacobi, user defined
- KERNELS: SpMV, SpMM
- DATA FORMATS: CSR, CPU converters from/to ELL and SELL-P
- BENCHMARKS: CG, GMRES, BiCGSTAN, SpMV, SpMM, BLAS

PERFORMANCE (in double precision)

P_{max} \sim 26 \text{ GFlop/s}

[ based on achievable bandwidth limit of about 160 \text{ GB/s} ]

85\% \text{ of peak}

14\% \text{ vs. MKL}

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

FEATURES AND SUPPORT in MAGMA MIC 1.4

<table>
<thead>
<tr>
<th>ROUTINES</th>
<th>CG, GMRES, BiCGSTAB, Iterative Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRECONDITIONERS</td>
<td>Jacobi, user defined</td>
</tr>
<tr>
<td>KERNELS</td>
<td>SpMV, SpMM</td>
</tr>
<tr>
<td>DATA FORMATS</td>
<td>CSR, CPU converters from/to ELL and SELL-P</td>
</tr>
<tr>
<td>BENCHMARKS</td>
<td>CG, GMRES, BiCGSTAN, SpMV, SpMM, BLAS</td>
</tr>
</tbody>
</table>

PERFORMANCE (in double precision)

- **SpMV**
  - 90% of SpMV

<table>
<thead>
<tr>
<th>Coprocessor</th>
<th>Intel Xeon Phi (60 @ 1.09 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP Peak</td>
<td>1046 GFlop/s</td>
</tr>
</tbody>
</table>

Flops / iter.

- **CG** 14n + SpMV
- **BiCGSTAB** 20n + 2 SpMV
- **GMRES(30)** 4(i+1) + SpMV
Future directions

- Important scientific applications rely on sparse LA ...
  - SpMV has low computational intensity & achieves only ~2% of the Xeon Phi’s compute peak
  - To increase the intensity, many applications/solvers organize the computation to be on small dense matrices, usually data-independent, and many, that can be “batched”
    - Tiled linear algebra algorithms
    - Multifrontal methods
    - Preconditioners (using DLA) in sparse iterative solvers
    - Tensor contractions (in high-order FEM, etc.)

Sparse / Dense Matrix System

To capture main LA patterns needed in a numerical library for Batched LA

- LU, QR, or Cholesky on small diagonal matrices
- TRSMs, QRs, or LUs
- TRSMs, TRMMs
- Updates (Schur complement) GEMMs, SYRKs, TRMMs

And many other BLAS/LAPACK, e.g., for application specific solvers, preconditioners, and matrices
Future directions

- Batched LA to provide support for various applications
- Communication-avoiding algorithms in dense and sparse LA applications
- Mixed-precision methods
- Benchmarks
Collaborators and Support

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

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

Intel MKL team

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