Doing Linear Algebra in Parallel

Stan Tomov
Research Director
Innovative Computing Laboratory
University of Tennessee, Knoxville

NICS Seminar Series on High-Performance Computing
University of Tennessee, Knoxville, TN
April 3, 2014
Outline

• Introduction
• Methodology
• Dense linear system and eigen-problem solvers
• Multi-GPU algorithms
  – Dynamic scheduling
  – Distributed MAGMA
• Sparse linear algebra
• Tutorial
Major Change to Software

• Must rethink the design of our software for heterogeneous architectures

  ➢ Another disruptive technology
    • Similar to what happened with cluster computing and message passing

  ➢ Rethink and rewrite the applications, algorithms, and software

• Numerical libraries for example will change

  ➢ For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this
### Next Generation of DLA Software

**Software/Algorithms follow hardware evolution in time**

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

**MAGMA**

Hybrid Algorithms (heterogeneity friendly)

Rely on
- hybrid scheduler (of DAGs)
- hybrid kernels (for nested parallelism)
- existing software infrastructure
Challenges on modern architectures

• The explosion of parallelism
e.g., single K40 has 2,880 CUDA cores; algorithms must account for these levels of parallelism

• The growing gap of compute vs. data-movement capabilities

K40 GPU computing efficiency on

Compute intensive (dense LU) vs. Memory-bound computation (SpMV)

Focus is on architecture-aware algorithms of high-parallelism and improved data access patterns
MAGMA: LAPACK for GPUs

• MAGMA
  – Matrix algebra for GPU and multicore architecture
  – The LAPACK/ScaLAPACK on hybrid architectures

• MAGMA 1.4.1, clMAGMA 1.1, MAGMA MIC 1.1
  – For correspondingly NVIDIA CUDA GPUs, AMD GPUs, and Intel Xeon Phi coprocessors (MICs) on shared memory systems
  – Hybrid dense linear algebra (for CPUs and accelerators)
    • One-sided factorizations and linear system solvers
    • Two-sided factorizations and eigenproblem solvers
    • A subset of BLAS and auxiliary routines in CUDA

• MAGMA developers & collaborators
  – UTK, UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  – Community effort, similarly to LAPACK/ScaLAPACK
Key Features of MAGMA

• High performance
• Multiple precision support (Z, C, D, S, and MP)
• Hybrid algorithms
• Out-of-GPU memory algorithms
• MultiGPU support
HYBRID ALGORITHM

MAGMA uses a hybridization methodology where algorithms of interest are split into tasks of varying granularity and their execution scheduled over the available 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 GPU.

PERFORMANCE

MAGMA on Kepler K40
LU factorization in double precision arithmetic

<table>
<thead>
<tr>
<th>GPU</th>
<th>NVIDIA K40 (Atlas) 15 MP x 192 @ 0.88 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon ES-2670 (Sandy Bridge) 2 x 8 cores @ 2.60 GHz</td>
</tr>
</tbody>
</table>

2 GPUs
MAGMA

1 GPU
MAGMA

CPU MKL

FEATURES AND SUPPORT

- **MAGMA 1.4.1** for CUDA
- **cIMAGMA 1.1** for OpenCL
- **MAGMA MIC 1.1** for Intel Xeon Phi

- Linear system solvers
- Eigenvalue problem solvers
- Auxiliary BLAS
- CPU Interface
- GPU Interface
- Multiple precision support
- Non-GPU-resident factorizations
- Multicore and multi-GPU support
- LAPACK testing
- Linux
- Windows
- Mac OS
Multiple precision support

Performance of the LU factorization in various precisions

Keeneland
GPU  M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
CPU  Intel Xeon X5660 (2x6 cores @2.8GHz)
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
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)
- How to use it? [see the short MAGMA tutorial]
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

Keeneland
GPU  M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
CPU  Intel Xeon X5660@2.80GHz (2 x 6 cores)
Out of GPU Memory Algorithms

Solving large problems that do not fit in the GPU memory

Matrices of size that do not fit in a specified GPU memory

Keeneland
- **GPU** M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
- **CPU** Intel Xeon X5660@2.80GHz (2 x 6 cores)
Out of GPU Memory Algorithms

Solving large problems that do not fit in the GPU memory

Out-of-GPU-memory Algorithms can now solve large problems

Keeneland
GPU  M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
CPU  Intel Xeon X5660@2.80GHz (2 x 6 cores)
Out of GPU Memory Algorithm

- 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
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 on multiGPUs in DP

- **1 GPU**
- **CPU (MKL)**

Keeneland
- **GPU** M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
- **CPU** Intel Xeon X5660@2.80GHz (2 x 6 cores)
LU on multiGPUs in DP

- **2 GPUs**
- **1 GPU**
- **CPU (MKL)**

**Keeneland**
- **GPU** M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
- **CPU** Intel Xeon X5660@2.80GHz (2 x 6 cores)
LU on multiGPUs in DP

- **3 GPUs**
- **2 GPUs**
- **1 GPU**
- **CPU (MKL)**

Keeneland
- **GPU** M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
- **CPU** Intel Xeon X5660@2.80GHz (2 x 6 cores)
LU on Kepler in DP

- Kepler (K20X)
- 3 GPUs
- 2 GPUs
- 1 GPU
- CPU (MKL)

Keeneland
GPU M2090 (14 MP @1.3 GHz, peak 583 GFlop/s)
CPU Intel Xeon X5660@2.80GHz (2 x 6 cores)
Eigenproblem Solvers in MAGMA

\[ A \mathbf{x} = \lambda \mathbf{x} \]

- Quantum mechanics (Schrödinger equation)
- Quantum chemistry
- Principal component analysis (in data mining)
- Vibration analysis (of mechanical structures)
- Image processing, compression, face recognition
- Eigenvalues of graph, e.g., in Google's page rank

• Need to solve it fast

Current MAGMA results:

MAGMA with 1 GPU can be 12x faster vs. vendor libraries on state-of-art multicore systems


Toward fast Eigensolvers for Non-symmetric Matrices

- $A$ is $n \times n$, nonsymmetric
- $Ax = \lambda x$
- Three phases:
  - Hessenberg reduction $H = Q_1^T A Q_1$
  - QR iteration to triangular form $T = Q_2^T H Q_2$
  - Compute eigenvectors $Z$ of $T$ and back-transform to eigenvectors $X$ of $A$

Characteristics

- Too many Blas-2 op,
- Relies on panel factorization,
- Bulk sync phases,
- Memory bound algorithm.

Toward fast Eigensolvers

Characteristics

- Blas-2 GEMV moved to the GPU,
- Accelerate the algorithm by doing all BLAS-3 on GPU,
- ➔ Bulk sync phases,
- ➔ Memory bound algorithm.

Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

**Toward fast Eigensolvers**

[Graph showing performance of different eigensolvers with varied matrix sizes.]

**flops formula:** $n^3/3 \times \text{time}$  
Higher is faster

Keeneland system, using one node  
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)  
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

**Acceleration w/ 3 GPUs:**  
15 X vs. 12 Intel cores

**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.

Current work

• Schedule task execution using Dynamic Runtime Systems

48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000, tile size is 200 x 200
High-productivity w/ Dynamic Runtime Systems
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], ...);
  }
}
Current work

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

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

Various runtime systems can be used:
- StarPU
  http://icl.cs.utk.edu/projectsdev/morse
- PaRSEC
  https://icl.cs.utk.edu/parsec/
- QUARK
  http://icl.cs.utk.edu/quark/
Scalability and efficiency:

- Snapshot of the execution trace of the Cholesky factorization on System A for a matrix of size 40K using six GPUs K20c.

- As expected the pattern of the trace looks compressed which means that our implementation is able to schedule and balance the tasks on the GPU devices (six).
Dynamic MAGMA with QUARK

![Graph showing the performance of DPOTRF on a K20c GPU across different matrix sizes.]
Dynamic MAGMA with QUARK
Dynamic MAGMA with QUARK

Matrix size

Gflop/s

DPOTRF 3 K20c
DPOTRF 2 K20c
DPOTRF 1 K20c
Dynamic MAGMA with QUARK

Matrix size

Gflop/s

DPOTRF 4 K20c
DPOTRF 3 K20c
DPOTRF 2 K20c
DPOTRF 1 K20c
Dynamic MAGMA with QUARK

![Graph showing performance of different matrix sizes for MAGMA with QUARK]
Distributed MAGMA

- Preliminary work on distributed memory systems
- Extensions of the Dynamic MAGMA
  - ScaLAPACK 2D block-cyclic data distribution
  - Lightweight “local” (node) scheduling with QUARK + MPI communications
  - Match in performance previous results using “tile” algorithms

![Graph](image_url)
Sparse Solvers
Sparse HPC on modern architectures

- Important scientific applications rely on sparse linear algebra

- **HPCG** – a new benchmark proposal to complement **Top500 (HPL)**
  - To solve $Ax = b$, where $A$ is large and sparse
  - To show essential communication & computation patterns in solving PDEs
  - To encourage the focus on architecture features and application needs
  - In collaboration with Sandia National Laboratory

- **MAGMA Sparse**
  - Develop GPU-aware Sparse Solvers
  - Support from DOE, DOD, and Nvidia
MAGMA Sparse

• Recently added MAGMA component for sparse linear algebra
• Under evaluation for release (MathWorks, friendly users, and collaborators)
• Current MAGMA Sparse functionality:
  – Krylov subspace iterative linear system and eigen-problem solvers
  – Support for various matrix formats
  – Sparse BLAS GPU kernels
  – Dense LA building blocks for preconditioners
Challenges on modern architectures

- The explosion of parallelism
e.g., single K40 has 2,880 CUDA cores; algorithms must account for these levels of parallelism

- The growing gap of compute vs. data-movement capabilities

K40 GPU computing efficiency on Compute intensive (dense LU) vs. Memory-bound computation (SpMV)

Focus is on architecture-aware algorithms of high-parallelism and improved data access patterns
GPU-aware Sparse Solvers
[ braking the memory bound performance limitation ! ]

• Reference as well as optimized (kernels & reduced communication) implementations

[included are SpMV / SpMM in various formats, e.g., DENSE, CSR, Block-CSR, ELLPACK, ELLPACKRT, HYB, COO, CSC, SELLC/SELLC-σ; and other kernels/building blocks ...]

Performance of SpMM for various matrices and a block of 32 vectors on a K40 GPU (GPU MAGMA & CUBLAS)

## GPU-aware Sparse Solvers

- Reference as well as optimized (kernels & reduced communication) implementations

[included are CG, BiCGSTAB, GMRES, preconditioned versions, CA-GMRES, and LOBPCG]

<table>
<thead>
<tr>
<th>BiCGSTAB Method implementation</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>while</strong> (( k &lt; maxiter ) &amp;&amp; ( res &gt; \tau ))</td>
<td><strong>while</strong> ( ( k &lt; \text{maxiter} ) &amp;&amp; ( \text{res_host} &gt; \text{epsilon} ) ){</td>
</tr>
<tr>
<td>( k := k + 1 )</td>
<td>( \text{magma_dbicgmerge_p_update}(&lt;&lt;\text{Gs}, \text{Bs}, 0&gt;&gt;) )</td>
</tr>
<tr>
<td>( \rho_k := -\hat{r}<em>0^Tr</em>{k-1} )</td>
<td>( ( \text{n, skp, v, r, p} ) ; )</td>
</tr>
<tr>
<td>( \beta_{k+1} := \rho_{k-1}/\omega_{k-1} )</td>
<td>( \text{magma_dbicgmerge_spmv1}(&lt;&lt;\text{Gs}, \text{Bs}, \text{Ms1}&gt;&gt;) )</td>
</tr>
<tr>
<td>( p_k := r_{k-1} + \beta(\rho_{k-1} - \omega_{k-1}v_{k-1}) )</td>
<td>( (\text{n, valA, rowA, coLA, p, r, v, d1} ) ; )</td>
</tr>
<tr>
<td>( v_k := Ap_k )</td>
<td>( \text{magma_dbicgmerge_reduce1}(\text{n, Gs, Bs, d1, d2, skp}) ; )</td>
</tr>
<tr>
<td>( \alpha_k := \frac{\rho_k}{\beta_{k+1}} )</td>
<td>( \text{magma_dbicgmerge_spmv2}(&lt;&lt;\text{Gs}, \text{Bs}, \text{Ms2}&gt;&gt;) )</td>
</tr>
<tr>
<td>( s_k := r_{k-1} - \alpha_kv_k )</td>
<td>( (\text{n, valA, rowA, coLA, s, t, d1} ) ; )</td>
</tr>
<tr>
<td>( t_k := As_k )</td>
<td>( \text{magma_dbicgmerge_reduce2}(\text{n, Gs, Bs, d1, d2, skp}) ; )</td>
</tr>
<tr>
<td>( \omega_k := \frac{\beta_{k+1}t_k}{\rho_k} )</td>
<td>( \text{magma_dbicgmerge_xr_update}(&lt;&lt;\text{Gs}, \text{Bs}, 0&gt;&gt;) )</td>
</tr>
<tr>
<td>( x_{k+1} := x_k + \alpha_kp_k + \omega_k s_k )</td>
<td>( (\text{n, skp, r_hat, r, p, s, t, x, d1}) ; )</td>
</tr>
<tr>
<td>( r_k := s_k - \omega_tv_k )</td>
<td>( \text{magma_dbicgmerge_reduce3}(\text{n, Gs, Bs, d1, d2, skp}) ; )</td>
</tr>
<tr>
<td>( \text{res} = r_k^T r_k )</td>
<td>( \text{magma_memcpy}(1, \text{skp+5}, \text{res_host}) ; )</td>
</tr>
<tr>
<td><strong>end</strong></td>
<td>( \text{k}++ ; )</td>
</tr>
</tbody>
</table>

---

GPU-aware Sparse Solvers

- Communication avoiding GMRES (CA-GMRES)
  - Replacing GMRES’ SpMV $\rightarrow$ Matrix Powers Kernel (MPK):
    
    $$v_{k+1} = A v_k \quad \text{for } k = j, \ldots, j+s$$

    BLAS-2 $\rightarrow$ BLAS-3 based orthogonalization (next ...)

MPK to generate 100 vectors for various $s$
Overall performance improvement on up to 3 GPUs

Orthogonalization procedures

- Mixed-precision Cholesky QR
  - CholQR obtains BLAS-3 performance, but error is bounded by $\varepsilon \kappa(V)^2$
  - Remove the “square” by selectively using double-double (doubled) precision

Step 1  Gram-matrix formation $B := V^T V$
  on GPUs in doubled-precision.

Step 2  Cholesky factorization $R^T R := B$
  on CPUs in doubled-precision.

Step 3  Backward-substitution $Q := VR^{-1}$
  on GPUs in standard-precision.

GPU-aware Sparse Eigen-Solvers

- Locally Optimal Block PCG (LOBPCG)
  - Find a set of smallest eigen-states of a sparse SPD matrix \( Ax = \lambda x \)
  - Replace finding the states one-by-one by a block algorithm
    - finds them simultaneously; needs fast SpMM, re-orthogonalization, and GEMM of particular sizes

<table>
<thead>
<tr>
<th>Performance of SpMM with various matrices (x 32 vec.)</th>
<th>Overall speedup vs. LOBPCG from BLOPEX on CPUs</th>
</tr>
</thead>
</table>

**Graphs:**
- **MAGMA** vs. **CUSPARSE**
- **BLOPEX LOBPCG:** uses CPU
- **MAGMA Sparse:** uses CPU
Batched DLA & Other Building Blocks

- Many small DLA problems solved in parallel
  - Needed in preconditioners [5,6], orthogonalization routines [4,5], some sparse direct solvers, high-order FEMs [8], and batched higher-level DLA [7]

Future Directions

• Distributed multi-GPU solvers
  – Scheduling
  – Reproducibility

• Further tuning
  – partitioning, comm/comp overlaps
  – autotuning

• Extended functionality
  – Other mixed precision techniques
  – Other communication-avoiding techniques
  – Sparse direct multifrontal solvers & preconditioners
  – Other techniques of bringing BLAS-3 performance to sparse solvers
MAGMA Tutorial

• Installing MAGMA
• Using MAGMA
  – Documentation, naming conventions, and functionality
• Solving a linear system of equations
  \[ Ax = b \]
• Solving an eigenvalue problem
  \[ Ax = \lambda x \]
• DGEMM example
• Writing a hybrid algorithm
Install MAGMA

• Get MAGMA 1.4
  > wget http://icl.cs.utk.edu/projectsfiles/magma/downloads/magma-1.4.1.tar.gz

• Unpack the library
  > tar zxfv magma-1.4.1.tar.gz

• Prerequisites: LAPACK, BLAS, and CUDA toolkit
  > module load cudatoolkit intel

• Modify ‘make.inc’ (where are CUDA & LAPACK, and for what GPU)
  See examples in make.inc.{mkl-gcc | mkl-icc | mkl-ilp64 | mkl-shared | acml | atlas | goto | shared},
  > cp make.inc.mkl-gcc make.inc
  and modify make.inc by setting GPU_TARGET, MKLROOT, and CUDADIR, e.g.,
  GPU_TARGET ?= Kepler
  MKLROOT = $(INTEL_PATH)/mkl
  CUDADIR = $(CRAY_CUDATOOLKIT_DIR)

• Create libmagma.a in ‘lib’ and testing drivers in ‘testing’
  > make

• For more information on installation, read file ‘README’
Using **MAGMA**

- Support is provided through the MAGMA user forum

http://icl.cs.utk.edu/magma/forum/

<table>
<thead>
<tr>
<th>Title</th>
<th>Views</th>
<th>Replies</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN errors with dpotrf and dpotrf_gpu</td>
<td>9</td>
<td>3095</td>
</tr>
<tr>
<td>by fletchjp » Tue Dec 28, 2010 7:08 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMRES on magma</td>
<td>4</td>
<td>1064</td>
</tr>
<tr>
<td>by nitinb60 » Sun Aug 12, 2012 8:15 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Help please with choice of forums</td>
<td>4</td>
<td>1628</td>
</tr>
<tr>
<td>@ by fletchjp » Sat Apr 13, 2013 2:44 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the error when I compile magma1.4.0 in vs2010</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>by Eva Joo » Mon Sep 09, 2013 4:57 am</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undefined reference to cuda functions within libmagma</td>
<td>1</td>
<td>63</td>
</tr>
<tr>
<td>by Matt Phillips » Mon Sep 09, 2013 9:15 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>crash testing strs for OpenBLAS</td>
<td>0</td>
<td>101</td>
</tr>
<tr>
<td>by Matt Phillips » Thu Sep 05, 2013 9:15 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGMA Installation: CLAPACK reference BLAS problem</td>
<td>0</td>
<td>84</td>
</tr>
<tr>
<td>by psrivas2 » Thu Sep 03, 2013 4:07 am</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Running MAGMA across several GPUs on several nodes</td>
<td>1</td>
<td>310</td>
</tr>
<tr>
<td>by hshasra » Tue Aug 13, 2013 1:32 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>magma_init/finalize missing in fortran interface</td>
<td>1</td>
<td>116</td>
</tr>
<tr>
<td>@ by stachon » Wed Aug 28, 2013 4:02 am</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error: BLAS/LAPACK routine 'magma_' gave error code -7</td>
<td>2</td>
<td>197</td>
</tr>
<tr>
<td>@ by christianHEL » Thu Aug 22, 2013 2:37 pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problems testing dsevd with magma-1.4.0</td>
<td>1</td>
<td>173</td>
</tr>
<tr>
<td>by dougrabson » Thu Aug 15, 2013 5:44 am</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using MAGMA

- Doxygen documentation
  http://icl.cs.utk.edu/magma/docs/

---

MAGMA  magma-1.4.0
Matrix Algebra on GPU and Multicore Architectures

File List
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp
- dgeqrf_gpu.cpp

Purpose
-----
DGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

\[ A = P \cdot L \cdot U \]

where \( P \) is a permutation matrix, \( L \) is lower triangular with unit diagonal elements (lower trapezoidal if \( m > n \)), and \( U \) is upper triangular (upper trapezoidal if \( m < n \)).

This is the right-looking Level 3 BLAS version of the algorithm. If the current stream is NULL, this version replaces it with user defined stream to overlap computation with communication.

Arguments
-----

\( M \)  (input) INTEGER
The number of rows of the matrix \( A \). \( M \geq 0 \).

\( N \)  (input) INTEGER
The number of columns of the matrix \( A \). \( N \geq 0 \).

\( A \)  (input/output) DOUBLE PRECISION array on the GPU, dimension...
Using MAGMA

• Naming conventions (e.g., `magma_dgesv_gpu`)
  
  – Prefix `magma_` or `magmablas_`
  
  – Followed by precision
    
    s single, d double, c single complex, or z double complex
    ds mixed double-single, or zc double complex-single complex
  
  – Matrix type
    
    general symmetric hermetian positive definite
    orthogonal unitary triangular
  
  – Operation
    
    sv solve ev eigenvalue problem
    trf triangular factorization gv generalized eigenvalue
    qrf QR factorization
  
  – Suffix `_gpu` if input and output are on the GPU memory
    (no suffix indicates CPU interface – use GPUs to accelerate where input and output are on the CPU)
Using MAGMA

- MAGMA functionality


### MAGMA 1.4.1 DRIVER ROUTINES

<table>
<thead>
<tr>
<th>MATRIX</th>
<th>OPERATION</th>
<th>ROUTINE</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>Solve using LU</td>
<td>{sdc2}gesv</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Solve using MP</td>
<td>{zdc,dc}gesv</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPD/HPD</td>
<td>Solve using Cholesky</td>
<td>{sdc2}posv</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Solve using MP</td>
<td>{zdc,dc}posv</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LLS</td>
<td>Solve LLS using QR</td>
<td>{sdc2}gesv</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Solve using MP</td>
<td>{zdc,dc}gesv</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GE</td>
<td>Compute e-values,</td>
<td>{sdc2}geev</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>optionally e-vectors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV/HE</td>
<td>Compute all e-values,</td>
<td>{sd}syevd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>optionally e-vectors</td>
<td>{cz}heevd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Range (D&amp;C)</td>
<td>{cz}heevd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Range (B&amp;I lt.)</td>
<td>{cz}heevx</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Range (MRRR)</td>
<td>{cz}heevr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GE</td>
<td>Compute SVD,</td>
<td>{sdc2}gesvd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>optionally s-vectors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPD/HPD</td>
<td>Compute all e-values,</td>
<td>{sd}sygvd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>optionally e-vectors</td>
<td>{cz}hegvd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Range (D&amp;C)</td>
<td>{cz}hegvd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Range (B&amp;I lt.)</td>
<td>{cz}hegvx</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Range (MRRR)</td>
<td>{cz}hegvr</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### MAGMA 1.4.1 COMPUTATIONAL ROUTINES

<table>
<thead>
<tr>
<th>MATRIX</th>
<th>OPERATION</th>
<th>ROUTINE</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>LU</td>
<td>{sdc2}getrf</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Solve</td>
<td>{sdc2}getrs</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPD/HPD</td>
<td>Invert</td>
<td>{sdc2}getri</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Cholesky</td>
<td>{sdc2}potrf</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TR</td>
<td>Invert</td>
<td>{sdc2}trtri</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GE</td>
<td>Multiply matrix by Q</td>
<td>{sdc2}ormqr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{cz}unmqr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>LQ factorization</td>
<td>{sdc2}gelqf</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>QL factorization</td>
<td>{sdc2}geqlf</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Multiply matrix by Q</td>
<td>{sdc2}ormql</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{cz}unmql</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GE</td>
<td>Hessenberg reduction</td>
<td>{sdc2}gehrd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Generate Q</td>
<td>{sdc2}orgqr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SY/HE</td>
<td>Tridiagonalization</td>
<td>{sd}sytrd</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Generate Q</td>
<td>{sd}orgtr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Multiply by Q</td>
<td>{sd}ormtr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{cz}unmtr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SVD</td>
<td>GE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reduction to standard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>form</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ABBREVIATIONS

- GE: General
- SPD/HPD: Symmetric/Hermitian Positive Definite
- TR: Triangular
- D&C: Divide & Conquer
- B&I It: Bisection & Inverse Iteration
- MP: Mixed-precision Iterative Refinement

### NAMING CONVENTION

magma_[routine name]_[gpu]
Examples / Exercises

• Solving a linear system of equations

\[ Ax = b \]

```c
#include "magma.h"
#include "magma_lapack.h"
...
magma_init();
cublasInit();
...
double *hA, *hB;  // A is a typical array on the CPU
magma_malloc_pinned((void **) &hA, lda * N * sizeof(double));  // A can be allocated in pinned memory
magma_malloc_cpu( (void **) &hB, nrhs * M * sizeof(double));
...
init_matrix(M, N, hA, lda);
...
magma_dgetrf( M, N, hA, lda, ipiv, &info);  // LU factorization (using CPU+GPU)
lapackf77_dgetrs( MagmaNoTransStr, &N, &nrhs, hA, &lda, ipiv, hB, &ldb, &info );  // Solve on CPU with LAPACK
```

Alternatively, there is a direct MAGMA function to solve:

see testing_dgesv.cpp and testing_dgetrf.cpp
Examples / Exercises

• Solving an eigenvalue problem
  \[ Ax = \lambda x \]

```c
#include "magma.h"
#include "magma_lapack.h"
...
magma_init();
cublasInit();
...
double *hA;  // A is a typical array on the CPU
magma_malloc_pinned((void **)&hA, lda * N * sizeof(double));  // A can be allocated in pinned memory
...
init_matrix(M, N, hA, lda);
...
magma_dsyevd( opts.jobz, opts.uplo,
  N, hA, lda, w1,
  h_work, lwork,
  iwork, liwork, &info );

see testing_dsyevd.cpp
```
Examples / Exercises

- DGEMM example

.\testing_dgemm -l -c

MAGMA 1.4.1, capability 3.0
device 0: Tesla K20X, 732.0 MHz clock, 5759.6 MB memory, capability 3.5

Usage: .\testing_dgemm [options] [-h|--help]

transA = N, transB = N

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>K</th>
<th>MAGMA Gflop/s (ms)</th>
<th>CUBLAS Gflop/s (ms)</th>
<th>CPU Gflop/s (ms)</th>
<th>MAGMA error</th>
<th>CUBLAS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1088</td>
<td>1088</td>
<td>1088</td>
<td>612.88 ( 4.20)</td>
<td>969.12 ( 2.66)</td>
<td>3.13 ( 823.44)</td>
<td>6.92e-15</td>
<td>6.92e-15</td>
</tr>
<tr>
<td>2112</td>
<td>2112</td>
<td>2112</td>
<td>667.85 ( 28.21)</td>
<td>1100.09 ( 17.13)</td>
<td>6.79 (2775.89)</td>
<td>8.12e-15</td>
<td>7.95e-15</td>
</tr>
<tr>
<td>3136</td>
<td>3136</td>
<td>3136</td>
<td>675.16 ( 91.36)</td>
<td>1144.59 ( 53.89)</td>
<td>11.14 (5538.38)</td>
<td>1.13e-14</td>
<td>1.13e-14</td>
</tr>
</tbody>
</table>
... 

see file testing_dgemm.cpp, sgemm.pdf, dgemm_fermi.cu, and directory testing
Writing a hybrid algorithm

Develop a hybrid CPU-GPU algorithm for this Matlab code

function [Q,R] = chol_qr_it(A)
    i=0;
    cn = 200;
    Q = A;
    G = Q'*Q;
    n = size(A,2);
    R = eye(n);

    while cn > 100, i = i + 1
        [u,s,v]=svd(G);
        [q,r]=qr(sqrt(s)*v');
        R = r * R;
        cn = sqrt(cond(s));
        Q = Q * inv(r);
        if cn>100
            G = Q'*Q;
        end;
    end;
return

Examples / Exercises

- Computations on small data to be done on the CPU
- Computations on large data to be done on the GPU
Examples / Exercises

• Writing a hybrid algorithm

Develop a hybrid CPU-GPU algorithm for this Matlab code

```matlab
function [Q,R] = chol_qr_it(A)
    i=0;
    cn = 200;
    Q = A;
    G = Q'*Q;
    n = size(A,2);
    R = eye(n);

    while cn > 100, i = i + 1
        [u,s,v]=svd(G);
        [q,r]=qr(sqrt(s)*v');
        R = r * R;
        cn = sqrt(cond(s));
        Q = Q * inv(r);
        if cn>100
            G = Q'*Q;
        end;
    end;
    return
end;
```

see testing_dgegqr_gpu.cpp and dgegqr_gpu.cpp
Contributions

The MAGMA program style follows the general guidelines for Sca/LAPACK in terms of interfaces, copyrights and licensing, citing the authors of the software, and documentation:
http://www.netlib.org/lapack-dev/lapack-coding/program-style.html

• Routine Naming and Design
  – Use BLAS
  – Develop in double complex; other precisions are generated
  – Machine parameters are determined at magma_init (as needed for tuning)
  – Block sizes are provided by extern "C" int magma_get_ROUTINE_nb(int m), i.e., not by ilaenv

• Language, source formatting, timing, and testing
  – Written in C/C++; wrappers for Fortran are supported

• See file ‘ContributorsGuige.txt’
Collaborators / Support

- MAGMA [Matrix Algebra on GPU and Multicore Architectures] team
  http://icl.cs.utk.edu/magma/

- PLASMA [Parallel Linear Algebra for Scalable Multicore Architectures] team
  http://icl.cs.utk.edu/plasma

- Collaborating partners
  - University of Tennessee, Knoxville
  - University of California, Berkeley
  - University of Colorado, Denver
  - INRIA, France
  - KAUST, Saudi Arabia