Improving multicore capabilities in hybrid CPUs/GPUs applications (Case of MAGMA)

DONFACK Simplice

Joint work with
Stanimire Tomov

Contributors: Jack Dongarra, Mathieu Faverge, Mark Gates, Jakub Kurzak, Piotr Luszczek, Asim YarKhan

Innovative Computing Laboratory, University of Tennessee

February 25, 2014
### Motivation

**Hybrid CPU/GPUs:**
- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton'11]. **Release:** magma/exp
Motivation

Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton'11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak'12].
Motivation

**Hybrid CPU/GPUs:**

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton'11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak'12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song'12].
Motivation

Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton’11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak’12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song’12].
- Multi-GPU Implementation of LU Factorization [Yulu’12].
Motivation

Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton’11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak’12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song’12].
- Multi-GPU Implementation of LU Factorization [Yulu’12].
- Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs [Donfack’13]
Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton'11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak'12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song'12].
- Multi-GPU Implementation of LU Factorization [Yulu'12].
- Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs [Donfack’13]
- Unified Development for Mixed Multi-GPU and Multi-Coprocessor Environments using a Lightweight Runtime Environment [Haidar’13].
Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton’11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak’12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song’12].
- Multi-GPU Implementation of LU Factorization [Yulu’12].
- Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs [Donfack’13]
- Unified Development for Mixed Multi-GPU and Multi-Coprocessor Environments using a Lightweight Runtime Environment [Haidar’13].
- MORSE. **Previous release:** magma/multigpu-dynamic ?.
Motivation

Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton’11]. **Release**: magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak’12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song’12].
- Multi-GPU Implementation of LU Factorization [Yulu’12].
- Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs [Donfack’13]
- Unified Development for Mixed Multi-GPU and Multi-Coprocessor Environments using a Lightweight Runtime Environment [Haidar’13].
- MORSE. **Previous release**: magma/multigpu-dynamic ?.
- PARSEC.
Motivation

Hybrid CPU/GPUs:

- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton'11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak'12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song'12].
- Multi-GPU Implementation of LU Factorization [Yulu'12].
- Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs [Donfack’13]
- Unified Development for Mixed Multi-GPU and Multi-Coprocessor Environments using a Lightweight Runtime Environment [Haidar’13].
- MORSE. **Previous release:** magma/multigpu-dynamic ?.
- PARSEC.
- StarPU.
**Motivation**

**Hybrid CPU/GPUs:**
- A Class of Hybrid LAPACK Algorithms for Multicore and GPU Architectures [Horton’11]. **Release:** magma/exp
- LU Factorization with Partial Pivoting for a Multicore System with Accelerators [Kurzak’12].
- Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems [Song’12].
- Multi-GPU Implementation of LU Factorization [Yulu’12].
- Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs [Donfack’13]
- Unified Development for Mixed Multi-GPU and Multi-Coprocessor Environments using a Lightweight Runtime Environment [Haidar’13].
- MORSE. **Previous release:** magma/multigpu-dynamic ?.
- PARSEC.
- StarPU.
- ...
Motivation (2)

What makes MAGMA highly competitive?

- It is simple to use and easy to maintain.
- Its multicore CPUs use relies on vendor multithreaded LAPACK, which is usually well optimized.
- It takes fully advantage of GPUs.
What makes MAGMA highly competitive?

- It is simple to use and easy to maintain.
- Its multicore CPUs use relies on vendor multithreaded LAPACK, which is usually well optimized.
- It takes fully advantage of GPUs.

What makes CPU-based development cumbersome?

- CPU codes require usually lot of tuning (cache hierarchy, granularity, data layout, scheduling ...).
- Combining CPU and GPU requires not just software capabilities but also hardware components consideration (as any heterogeneous systems).
- CPU/GPU communication is costly and requires pre-allocated pinned memory (very costly).
- Is CPU code performance unpredictable?
Importantly, goal:

- Bring hybrid CPU/GPU development into a standard (e.g. LAPACK/ScaLAPACK).
- Schedule properly works between CPUs and GPUs.
- Propose a theoretical model to guide performance and avoid tuning-effect.
## Goal

**Important goal:**
- Bring hybrid CPU/GPU development into a standard (e.g. LAPACK/ScaLAPACK).
- Schedule properly works between CPUs and GPUs.
- Propose a theoretical model to guide performance and avoid tuning-effect.

**Previous work:**
- Combine the main idea of communication avoiding algorithm with good balancing of work between CPUs and GPUs.
- Provide a simple and accurate model to guide the distribution of the workload among CPUs and GPUs.
Goal

**Important goal:**
- Bring hybrid CPU/GPU development into a standard (e.g. LAPACK/ScaLAPACK).
- Schedule properly works between CPUs and GPUs.
- Propose a theoretical model to guide performance and avoid tuning-effect.

**Previous work:**
- Combine the main idea of communication avoiding algorithm with good balancing of work between CPUs and GPUs.
- Provide a simple and accurate model to guide the distribution of the workload among CPUs and GPUs.

**Current work:**
- Integrate our previous work in MAGMA using QUARK, which is a better scheduling strategy.
- Enable and extend in MAGMA for:
  - Recursive parallel panel, QR, CAQR, SVD reduction and more.
  - Multi-GPUs.
- Propose a programming model.
1 Introduction
   - Motivation
   - Goal

2 MAGMA approach

3 Asynchronous and optimized multicore capabilities in MAGMA (AMC)
   - New scheduling approach (AMC)
   - Performance Model
   - Programming model

4 Experiments
   - Performance
   - Profiling
   - Testing
Hybrid LU factorization in MAGMA

- Panel are factorized on the CPUs.
- Update of the trailing submatrices are performed on the GPUs.

**Figure**: Example of execution of magma_dgetrf on a square matrix.
MAGMA standard approach

Hybrid LU factorization in MAGMA
- Panel are factorized on the CPUs.
- Update of the trailing submatrices are performed on the GPUs.

Figure: Example of execution of magma_dgetrf on a square matrix.

- Efficient updates and optimal use of the GPUs.
- Load imbalance between CPUs and GPUs.
- Poor multicore scalability.
Asynchronous and optimized multicore capabilities in MAGMA (AMC)

**New approach in 3 phases:**

- A better scheduling strategy to keep both CPU and GPU busy.
- A theoretical model to guide the distribution of the workload among CPUs and GPUs.
- A programming model.
The matrix is partitioned into two parts for the CPUs and the GPU. Panel factorization and the corresponding trailing submatrix update in the first part are performed on the CPUs, while update of the trailing submatrices in the second part are performed on the GPU. Each factorized panel is asynchronously sent to the GPU. Some block columns are dynamically sent to the CPUs during the runtime to balance work.
Performance Model

Use of a model to determine the amount of work for the CPUs part.

We consider the parameters:
- $d$, the number of block column in the CPUs part.
- $P$, the number of processors for the CPUs part.
- $g_1$ and $g_2$, the peak performance of one CPU and one GPU respectively.

At each step of the factorization $K$, we consider:
- $N_K$, the number of block column of the remaining matrix.
- $W_{CPUs}$ and $W_{GPU}$, the amount of work required to compute the CPUs part and GPU part respectively.
- $T_{CPUs}$ and $T_{GPU}$, the time required to complete $W_{CPUs}$ and $W_{GPU}$.

Figure: Initial matrix decomposition.
Performance Model (2)

Figure: Initial matrix decomposition.

\[ W_{CPUs} = W_{panel} + (d-1)W_{update} \quad \text{and} \quad T_{CPUs} = \frac{W_{CPUs}}{P \times g_1} \]
\[ W_{GPU} = (N_K - d)W_{update}\quad \text{and} \quad T_{GPU} = \frac{W_{GPUs}}{g_2} \]

By solving \( T_{CPUs} = T_{GPU} \), we obtain:

\[
\frac{d}{N_K} = \frac{P g_1}{P g_1 + g_2}
\]

\( \frac{d}{N_K} \) represents the percentage of the matrix to assign to the CPUs.
Programming model

**Purpose of our programming model:**

- Express the algorithm as a set of CPUs and GPUs tasks.

- Use a scheduling wrapper to insert tasks in the execution queue. This allows the use of any scheduling software.

- Allow any thread to control the GPUs using their device ID. e.g `magma_dev_dgemm(DeviceID, ...)`

- Then, keep the **code simple**. Provide an interface friendly for LAPACK and MAGMA users.
Algorithm 1 MAGMA LU using multithreaded MKL

1: **Input**: $m \times n$ matrix $dA$, block size $b$, number of processors $P$, $M = m / b$, $N = n / b$
2: **workspace**: $A$, dimension $(m, b)$
3: magma_getMatrix($dA(0 : M - 1, 0), A(0 : M - 1, 0)$)
4: for $K = 1$ to $N = n / b$ do
5:   lapack_dgetrf($A(K, K)$, ...)
6:   if $N - K > 0$ then
7:     magma_setMatrix($A(K, K)$, $dA(K, K)$)
8:     magma_dtrsm($dA(K, K + 1)$, $dA(K, K)$)
9:     magma_dgemm($dA(K + 1, K)$, $dA(K, K + 1)$, $dA(K + 1, K + 1)$)
10:    magma_getMatrix($dA(K + 1, K + 1)$, $A(K + 1, K + 1)$)
11:  end if
12: end for

- Difference with LAPACK in red.
- Line 5 (done on CPU) is overlapped with Line 9 (done on GPU).
Introduction
MAGMA approach
Asynchronous and optimized multicore capabilities in MAGMA (AMC)
Experiments

New scheduling approach (AMC)
Performance Model
Programming model

Programming model: LU example

Algorithm 3 MAGMA LU using multithreaded MKL

1: Input: \( m \times n \) matrix \( dA \), block size \( b \), number of processors \( P, M = m/b, N = n/b \)
2: workspace: \( A, \text{dimension}(m, b) \)
3: \text{magma}\_getMatrix(\( dA(0 : M - 1, 0), A(0 : M - 1, 0), \) )
4: for \( K = 1 \) to \( N = n/b \) do
5: lapack\_dgetrf(\( A(K, K), \) ...) 
6: if \( N - K > 0 \) then
7: \text{magma}\_setMatrix(\( A(K, K), dA(K, K) \))
8: magma\_dtrsm(\( dA(K + 1, K), dA(K, K) \))
9: magma\_dgemm(\( dA(K + 1, K), dA(K + 1, K + 1) \))
10: \text{magma}\_getMatrix(\( dA(K+1, K+1), A(K+1, K+1) \))
11: end if
12: end for

Difference with LAPACK in red.
Line 5 (done on CPU) is overlapped with Line 9 (done on GPU).

Algorithm 4 MAGMA LU expressed as tasks (no optimization)

1: Input: \( m \times n \) matrix \( dA \), block size \( b \), number of processors \( P, M = m/b, N = n/b \)
2: workspace: \( A, \text{dimension}(m, b) \)
3: \text{magma}\_getMatrix(\( dA(0 : M - 1, 0), A(0 : M - 1, 0), \) )
4: \text{magma}\_schedule\_init(\( P \) )
5: for \( K = 1 \) to \( N = n/b \) do
6: \text{magma}\_insert\_core\_dgetrf(\( A(K, K), P, \) ...)
7: if \( N - K - d > 0 \) then
8: \text{magma}\_insert\_setMatrix(\( A(K, K), dA(K, K), \) )
9: \text{magma}\_insert\_dtrsm(\( dA(K + 1, K + d), dA(K, K) \))
10: \text{magma}\_insert\_dgemm(\( dA(K + 1, K + d), dA(K, K + d), dA(K + 1, K + d) \))
11: \text{magma}\_insert\_getMatrix(\( dA(K+1, K+d), A(K+1, K+d) \))
12: end if
13: end for
14: \text{magma}\_schedule\_barrier(\( P \) )

MAGMA LU using the new programming model (no optimization)
- Easy adaptation.
- Blocking operations or synchronizations converted to parallel tasks with dependencies among them.
- Dependencies internally and automatically tracked using data pointer.
- Same performance as MAGMA using multithreaded MKL.
Algorithm 5  MAGMA LU using multithreaded MKL

1: Input: \( m \times n \) matrix \( dA \), block size \( b \), number of processors \( P, M = m/b, N = n/b \)
2: workspace: \( A, \text{dimension}(m, b) \)
3: magma_getMatrix(\( dA(0: M - 1, 0), A(0: M - 1, 0), \) )
4: for \( K = 1 \) to \( N = n/b \) do
5: \( \text{lapack_dgetrf}(A(K, K), ...) \)
6: if \( N - K > 0 \) then
7: \( \text{magma_setMatrix}(A(K, K), dA(K, K)) \)
8: \( \text{magma_dtrsm}(dA(K, K + 1), dA(K, K)) \)
9: \( \text{magma_dgemm}(dA(K + 1, K), dA(K + 1, K + 1)) \)
10: end if
11: end for

- Difference with LAPACK in red.
- Line 5 (done on CPU) is overlapped with Line 9 (done on GPU).

Algorithm 6  MAGMA LU using AMC

1: Input: \( m \times n \) matrix \( dA \), block size \( b \), number of processors \( P = P_r \times P_c, M = m/b, N = n/b \)
2: \( d: \) number of blocks for the CPU parts
3: workspace: \( A, \text{dimension}(m, d = b) \)
4: magma_getMatrix(\( dA(0: M - 1, 0: d - 1), A(0: M - 1, 0: d - 1), \) )
5: magma_copy_1D_col_bycyclic(\( dA(:, d : N), \text{NGPU} \))
6: magma_schedule_init(\( P \))
7: for \( K = 1 \) to \( N = n/b \) do
8: \( \text{magma_insert_core_dgetrf}(A(K, K), P_r, ...) \)
9: for \( J = K + 1 \) to \( K + d - 1 \) do
10: \( \text{magma_insert_core_dtrsm}(A(K, K), A(K, J) ...) \)
11: for \( I = K \) to \( M - 1 \) do
12: \( \text{magma_insert_core_dgemm}(A(I, K), A(K, J), A(I, J) ...) \)
13: end for
14: end for
15: if \( N - K - d > 0 \) then
16: for \( dd = 0 \) to \( \text{NGPU} \) do
17: \( \text{magma_dev_insert_setMatrix}(dd, A(K, K), dA(K, K)) \)
18: \( \text{magma_dev_insert_dtrsm}(dd, dA(K, K + d), dA(K, K)) \)
19: \( \text{magma_dev_insert_dgemm}(dd, dA(K + 1, K), dA(K + 1, K + d)) \)
20: end for
21: \( dd = \text{gpu_owner}(K+d) \)
22: \( \text{magma_dev_insert_getMatrix}(dd, dA(K + 1, K + d), A(K + 1, K + d)) \)
23: end if
24: end for
1 Introduction
   - Motivation
   - Goal

2 MAGMA approach

3 Asynchronous and optimized multicore capabilities in MAGMA (AMC)
   - New scheduling approach (AMC)
   - Performance Model
   - Programming model

4 Experiments
   - Performance
   - Profiling
   - Testing
Experiments (1)

Figure: Performance using 48 cores and 1 GPU.

- `magma_dgetrf`: magma using multithreaded MKL
- `magma_dgetrf_amc`: magma using AMC (pinned memory buffer allocated internally)
- `magma_dgetrf_work_amc`: magma using AMC and workspace (pinned memory buffer preallocated before the execution)

AMD opteron 6180: 4x12 cores @2.5Ghz; Peak performance CPUs: 480.0 Gflops/s, GPU: 504 GFlops/s, Total: 984.0GFlops/s.
Experiments (2)

Figure: Performance using 48 cores and 4 GPUs.

AMD opteron 6180: 4x12 cores @2.5Ghz; Peak performance CPUs: 480 Gflops/s, GPUs: 2016 GFlops/s, Total: 2496GFlops/s.
Experiments (3)

**Figure**: Performance using 16 cores and 1 GPU.

Intel Xeon E5-2670: 2x8 cores @2.6Ghz; Peak performance CPUs: 332.8 Gflops/s, GPU: 665 GFlops/s, Total: 997.8GFlops/s.
Experiments (4)

Figure: Performance using 48 cores and 3 GPUs.

Intel Xeon E5-2670: 2x8 cores @2.6Ghz; Peak performance CPUs: 332.8 Gflops/s, GPUs: 1995 GFlops/s, Total: 2327GFlops/s.
**Profiling (1)**

*Figure*: Multi-GPU Profiling. Every thread communicates with the GPUs.

*Legend*
- **CPU** (first 16th lines): **Red**: panel factorization. **Green**: trailing submatrix update. **Black**: communication.
- **GPU** (last 2th lines): **Light green**: trailing submatrix update. **Blue**: communication.
Asynchronous and optimized multicore capabilities in MAGMA (AMC)

Experiments

Performance

Profiling

Testing

Introduction

MAGMA approach

Profiling (1)

**Figure**: Multi-GPU Profiling. Every thread communicates with the GPUs.

**Figure**: Multi-GPU Profiling. A dedicated thread communicates with the each GPU

**Legend:**

CPU (first 16th lines): **Red**: panel factorization. **Green**: trailing submatrix update. **Black**: communication.

GPU (last 2th lines): **Light green**: trailing submatrix update. **Blue**: communication.
Introduction
MAGMA approach
Experiments

Asynchronous and optimized multicore capabilities in MAGMA (AMC)

Performance
Profiling
Testing

Profiling (2)

Figure: Profiling of LU factorization using 1 GPU

Figure: Profiling of CALU factorization using 1 GPU

Legend:

GPU (last 2th lines): Light green: trailing submatrix update. Blue: communication.
Code available as a module.

> build magma
> cd exp_magma_quark
> make
> numactl –interleave=all ./test/testing_dgetrf_mgpu_amc
-N 10112 –nb 128 –ngpu 4 –nthread 48 –panel_nthread 4 –fraction 0.1
Thank you.