clMAGMA: Heterogeneous High-Performance Linear Algebra with OpenCL

Stan Tomov
w/ C. Cao, J. Dongarra, P. Du, M. Gates, P. Luszczek

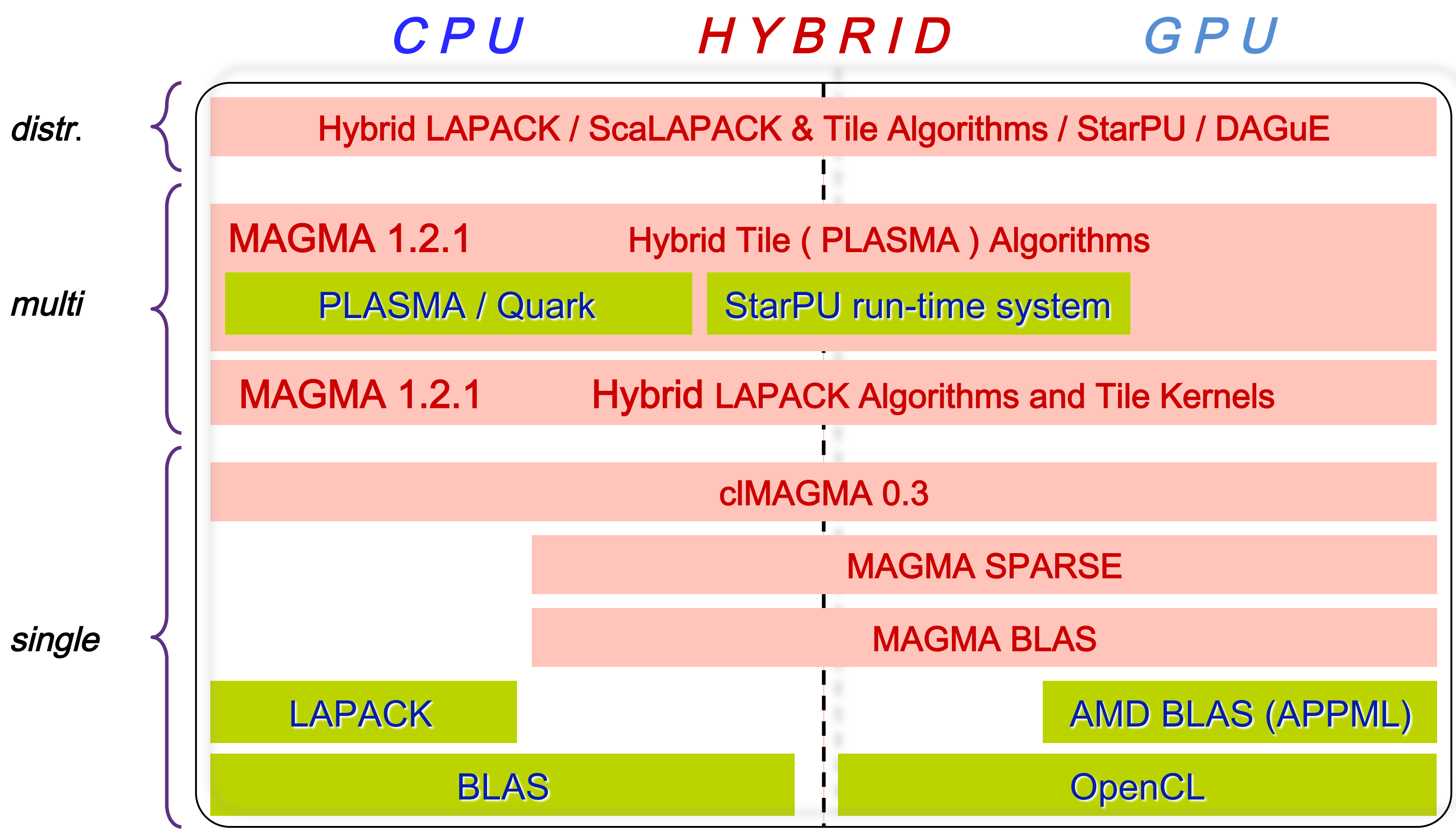
Innovative Computing Laboratory
University of Tennessee, Knoxville

Friday Lunch ICL Talk
August 3, 2012

Outline

- **Methodology overview**
  - ***Hybridization* of Linear Algebra Algorithms**
  - **Use both GPUs and multicore CPUs**
- **clMAGMA**
  - **OpenCL port of MAGMA**
  - **clMAGMA 0.3**
  - **Performance results**
  - **Challenges and future directions**
- **Conclusions**

clMAGMA Software Stack



Linux, Windows, Mac OS X | *C/C++, Fortran* | *Matlab, Python*

[**AMD APPML** -- Accelerated Parallel Processing Math Libraries
http://developer.amd.com/libraries/appmathlibs/]

MAGMA Methodology

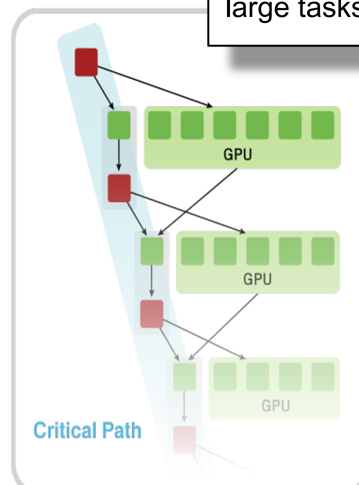
A methodology to use all available resources:

- 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 eigen-solvers

- Productivity
  - 1) High-level; 2) Leveraging prior developments; 3) Exceeding in performance homogeneous solutions

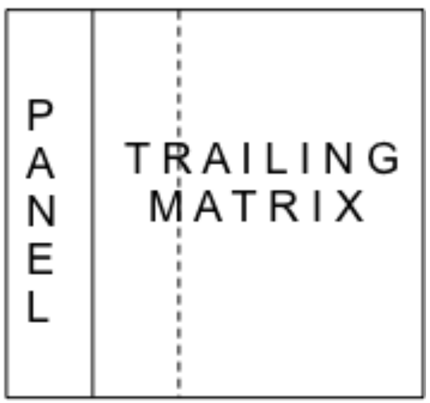
Hybrid CPU+GPU algorithms (small tasks for multicores and large tasks for GPUs)



Hybrid Algorithms

One-sided factorizations (LU, QR, Cholesky)

- **Hybridization**
  - **Panels (Level 2 BLAS) are factored on CPU using LAPACK**
  - **Trailing matrix updates (Level 3 BLAS) are done on the GPU using "look-ahead"**



A Hybrid Algorithm Example

- Left-looking hybrid Cholesky factorization in clMAGMA

```
1   for ( j=0; j<n; j += nb)  {
2        jb = min(nb, n – j);
3        magma_zherk( MagmaUpper, MagmaConjTrans, jb, j, m_one, dA(0, j), ldda, one, dA(j, j), ldda, queue );
4        magma_zgetmatrix_async( jb, jb, dA(j,j), ldda, work, 0, jb, queue, &event );
5        if ( j+jb < n )
6            magma_zgemm( MagmaConjTrans, MagmaNoTrans, jb, n-j-jb, j, mz_one,
7                                   dA(0, j ), ldda, dA(0, j+jb), ldda, z_one,  dA(j, j+jb), ldda, queue );
8        magma_event_sync( event );
9        lapackf77_zpotrf( MagmaUpperStr, &jb, work, &jb, info );
10       if ( *info != 0 )
11            *info += j;
12       magma_zsetmatrix_async( jb, jb, work, 0, jb, dA(j,j), ldda, queue, &event );
13       if ( j+jb < n ) {
12           magma_event_sync( event );
13           magma_ztrsm( MagmaLeft, MagmaUpper, MagmaConjTrans, MagmaNonUnit,
14                                jb, n-j-jb, z_one, dA(j, j), ldda, dA(j, j+jb), ldda, queue );
15        }
16   }
```

- The difference with LAPACK – the 4 additional lines in red
- Line 9 (done on CPU) is overlapped with work on the GPU (from line 6)

Programming model

Host program

```
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 );
    }
}
```

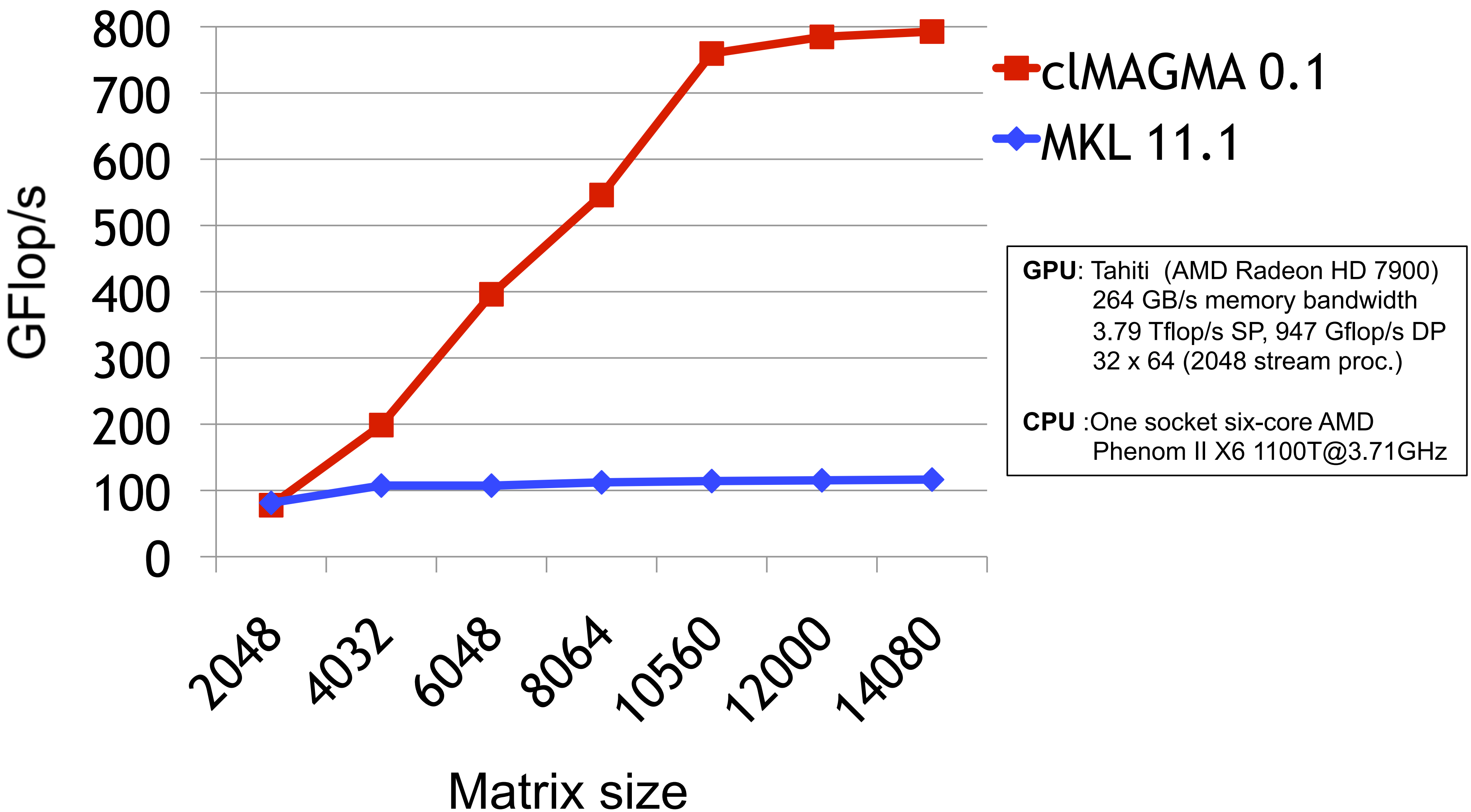
OpenCL interface – communications

```
magma_err_t
magma_zgetmatrix_async(
    magma_int_t m, magma_int_t n,
    magmaDoubleComplex_const_ptr dA_src, size_t dA_offset, magma_int_t
    magmaDoubleComplex*          hA_dst, size_t hA_offset, magma_int_t
    magma_queue_t queue, magma_event_t *event )
{
    size_t buffer_origin[3] = { dA_offset*sizeof(magmaDoubleComplex),
    size_t host_orig[3]     = { 0, 0, 0 };
    size_t region[3]        = { m*sizeof(magmaDoubleComplex), n, 1 };
    cl_int err = clEnqueueReadBufferRect(
        queue, dA_src, CL_FALSE,  // non-blocking
        buffer_origin, host_orig, region,
        ldda*sizeof(magmaDoubleComplex), 0,
        ldha*sizeof(magmaDoubleComplex), 0,
        hA_dst, 0, NULL, event );
```

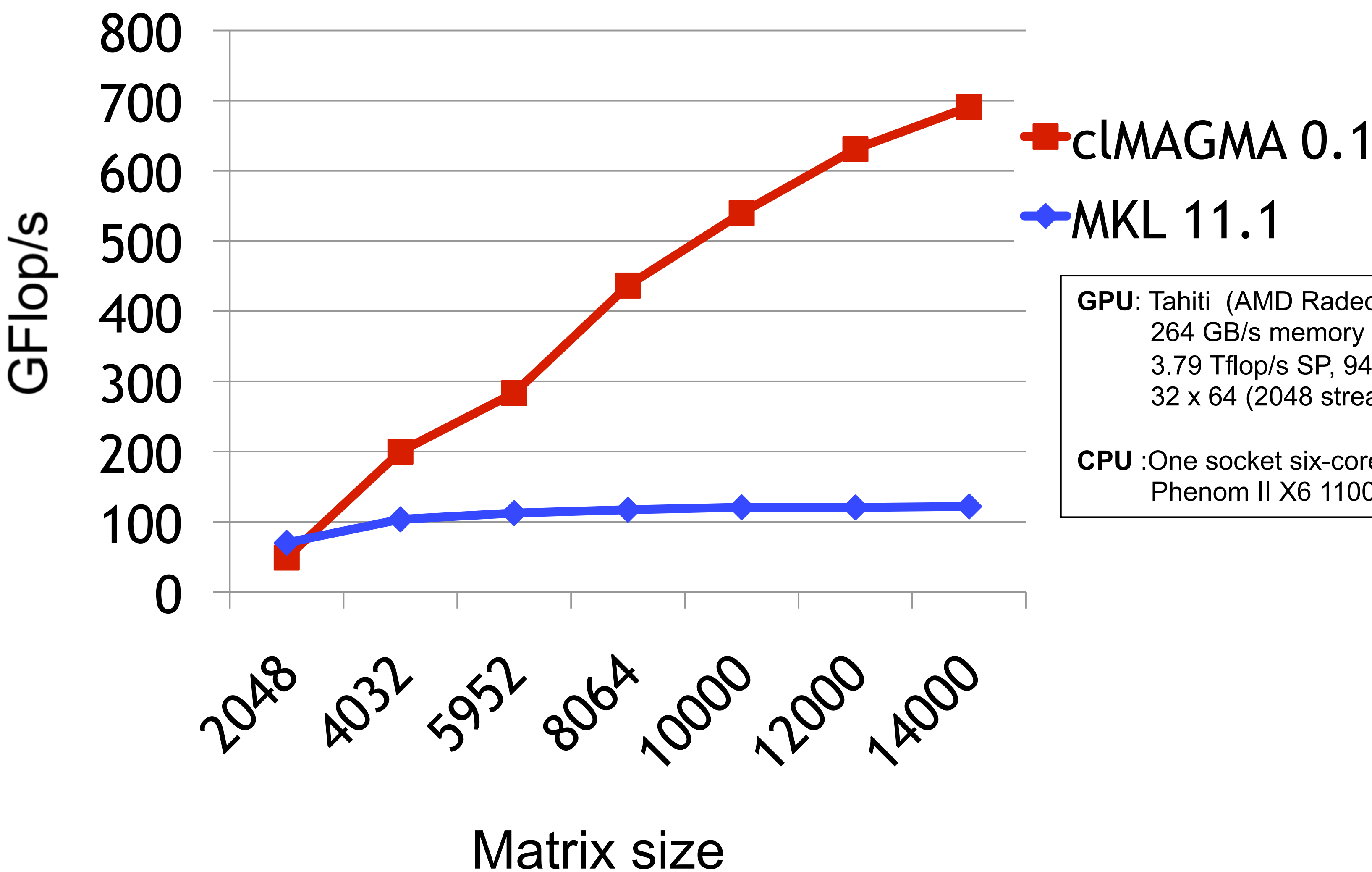
OpenCL interface – AMD APPML BLAS

```
magma_zherk(
    magma_uplo_t uplo, magma_trans_t trans,
    magma_int_t n, magma_int_t k,
    double alpha, magmaDoubleComplex_const_ptr dA, size_t dA_offset,
    double beta,  magmaDoubleComplex_ptr       dC, size_t dC_offset,
    magma_queue_t queue )
{
    cl_int err = clAmdBlasZherk(
        clAmdBlasColumnMajor,
        amdblas_uplo_const( uplo ),
        amdblas_trans_const( trans ),
        n, k,
        alpha, dA, dA_offset, lda,
        beta,  dC, dC_offset, ldc,
        1, &queue, 0, NULL, NULL );
    return err;
}
```

Performance of clMAGMA 0.3
Cholesky Factorization in single precision



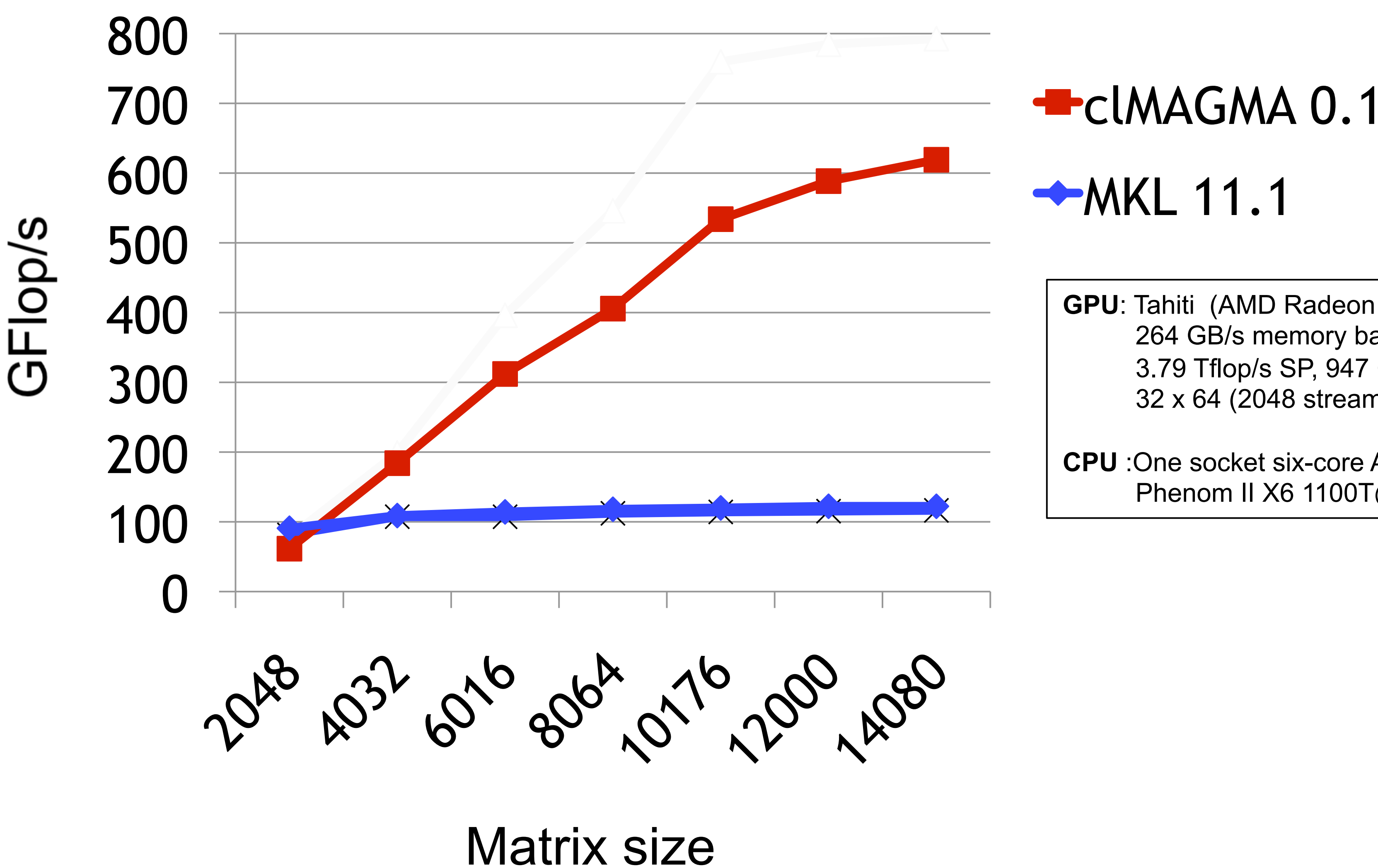
Performance of clMAGMA 0.3
LU Factorization in single precision



GPU: Tahiti (AMD Radeon HD 7900)
264 GB/s memory bandwidth
3.79 Tflop/s SP, 947 Gflop/s DP
32 x 64 (2048 stream proc.)

CPU :One socket six-core AMD
Phenom II X6 1100T@3.71GHz

Performance of clMAGMA 0.3
QR Factorization in single precision



GPU: Tahiti (AMD Radeon HD 7900)
264 GB/s memory bandwidth
3.79 Tflop/s SP, 947 Gflop/s DP
32 x 64 (2048 stream proc.)

CPU :One socket six-core AMD
Phenom II X6 1100T@3.71GHz

clMAGMA 0.3

MAGMA

News

- Home
- Overview
- News
- Software
- Publications
- People
- Partners
- Documentation
- User Forum

clMAGMA 0.3 Released

2012-06-29

clMAGMA 0.3 is now available. This release provides OpenCL support for the main two-sided matrix factorizations -
basis for eigenproblem and SVD solvers. In particular, clMAGMA 0.3 adds the following new functionalities:

- Reduction to upper Hessenberg form by similarity orthogonal transformations (routines magma_{z|c|d|s}gehrd);
- Reduction to upper/lower bidiagonal form by similarity orthogonal transformations (routines magma_{z|c|d|s}gebrd);
- Reduction to tridiagonal form by similarity orthogonal transformations (routines magma_{zhe|che|dsy|ssy}trd);

Further detail can be found in the ReleaseNotes.

See the MAGMA software homepage for a **download link**.

Sponsored By: Industry Support From:

clMAGMA Roadmap

Software Releases	Functionality
clMAGMA 0.1 Beta April 1 ✔	One-sided factorizations (LU, QR, and Cholesky)
clMAGMA 0.2 May 1 ✔	Add linear solvers and least squares
clMAGMA 0.3 July 1 ✔	Add two-sided factorizations
clMAGMA 0.4 September 1	Add matrix inversion, SVD, eigen-problem solvers
clMAGMA 1.0 November 1	Add multiGPU support for the one-sided factorizations

Current work

- **Add functionality following the roadmap**
  - **This was the priority so far**
- **Dynamic scheduling**
  - **MultiGPU and distributed environment**
- **Efficiency improvements**
  - **To match in performance the underlying BLAS**
  - **Discover and eliminate performance bottlenecks [using tracing, performance models, etc.]**
  - **OpenCL-specific optimizations**

Dynamic Scheduling

- **Conceptually similar to out-of-order processor scheduling because it has:**
  - Dynamic runtime DAG scheduler
  - Out-of-order execution flow of fine-grained tasks
  - Task scheduling as soon as dependencies are satisfied
  - Producer-Consumer

- **Data Flow Programming Model**
  - The DAG approach
  - Scheduling is data driven
  - Inherently parallel

High Level of Productivity

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

High Level of Productivity

From Sequential Nested-Loop Code to Parallel Execution:

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

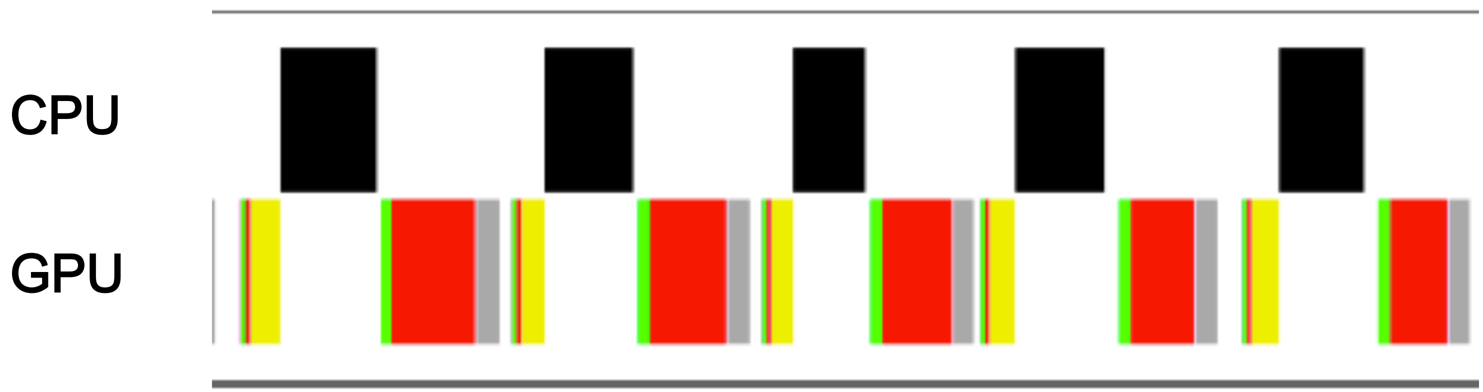
StarPU run-time: http://runtime.bordeaux.inria.fr/StarPU/

in collaboration w/ **INRIA, France** and **KAUST, Saudi Arabia**

Efficiency improvements

- **Tracing**

  A dgetrf trace example

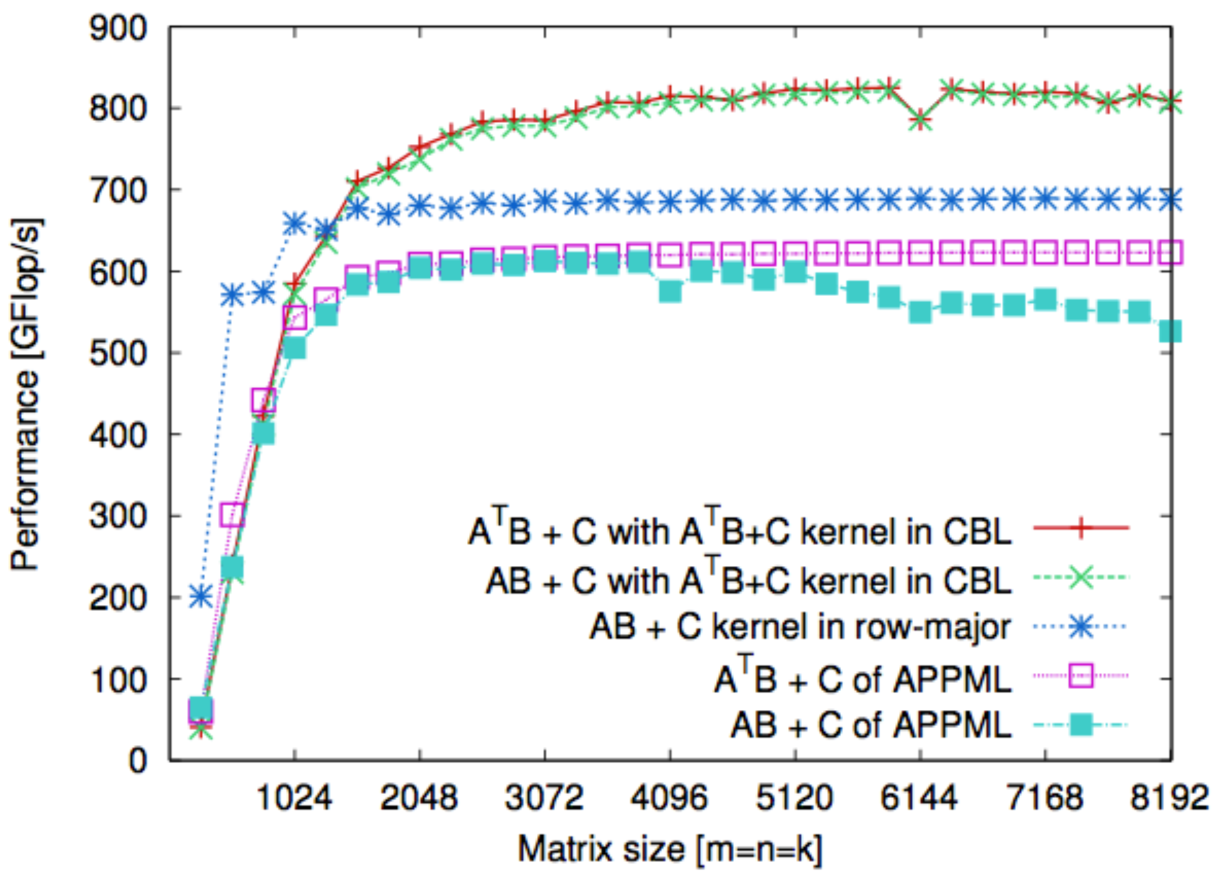


Black – panel　　Red – dgemm　　Green – dtrsm　　Gray / Yellow – dset / dget

- Shows several opportunities for improvements
  - overlap panel w/ update ✔
  - overlap GPU computation and communication (not clear if it is yet possible) ?
    [leads to about 30% performance loss compared to BLAS]
  - Improvements in the panel factorization on the CPU ✔

DGEMM in OpenCL

Kazuya Matsumoto, Naohito Nakasato, Stanislav G.Sedukhin, *Implementing a Code Generator for Fast Matrix Multiplication in OpenCL on the GPU*, University of Aizu, Japan, July 2, 2012.

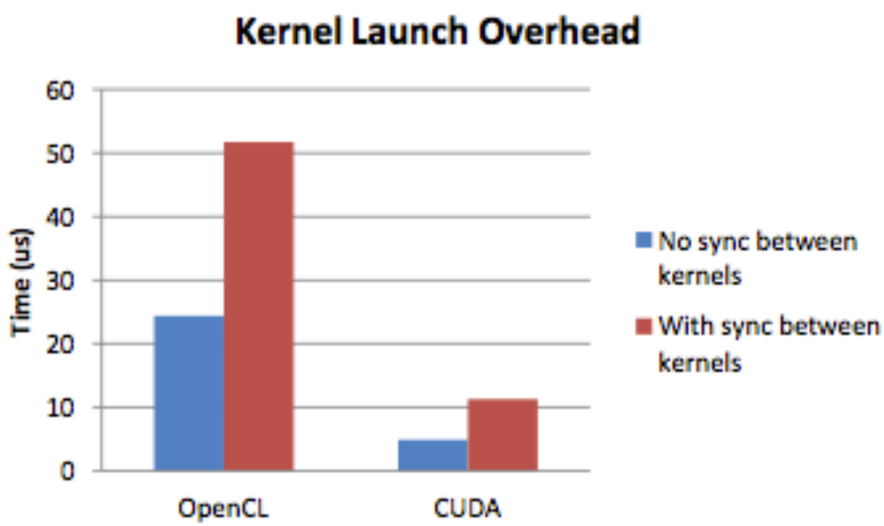


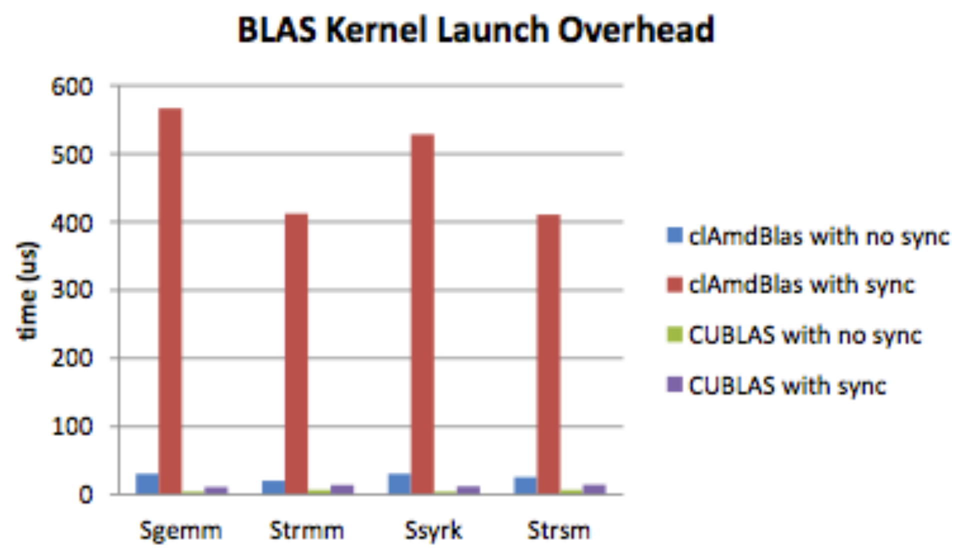
DGEMM

GPU: Tahiti (AMD Radeon HD 7900)
264 GB/s memory bandwidth
3.79 Tflop/s SP, 947 Gflop/s DP
32 x 64 (2048 stream proc.)

OpenCL-specific optimizations

- **"Latencies" are in general higher in OpenCL**





Summary and Future Directions

- **A hybrid methodology and its application to DLA using OpenCL**

- **clMAGMA: LAPACK for heterogeneous computing**
  - Achieving high-performance linear algebra using OpenCL
  - clMAGMA 0.3 includes the main one- and two-sided factorizations and linear solvers

- **What is next?**
  - Eigen/singular-value problem solvers
  - Further performance/efficiency improvements
  - MultiGPU and distributed environments

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