MAGMA

Matrix Algebra on GPU and Multicore Architectures

Innovative Computing Laboratory
Electrical Engineering and Computer Science
University of Tennessee

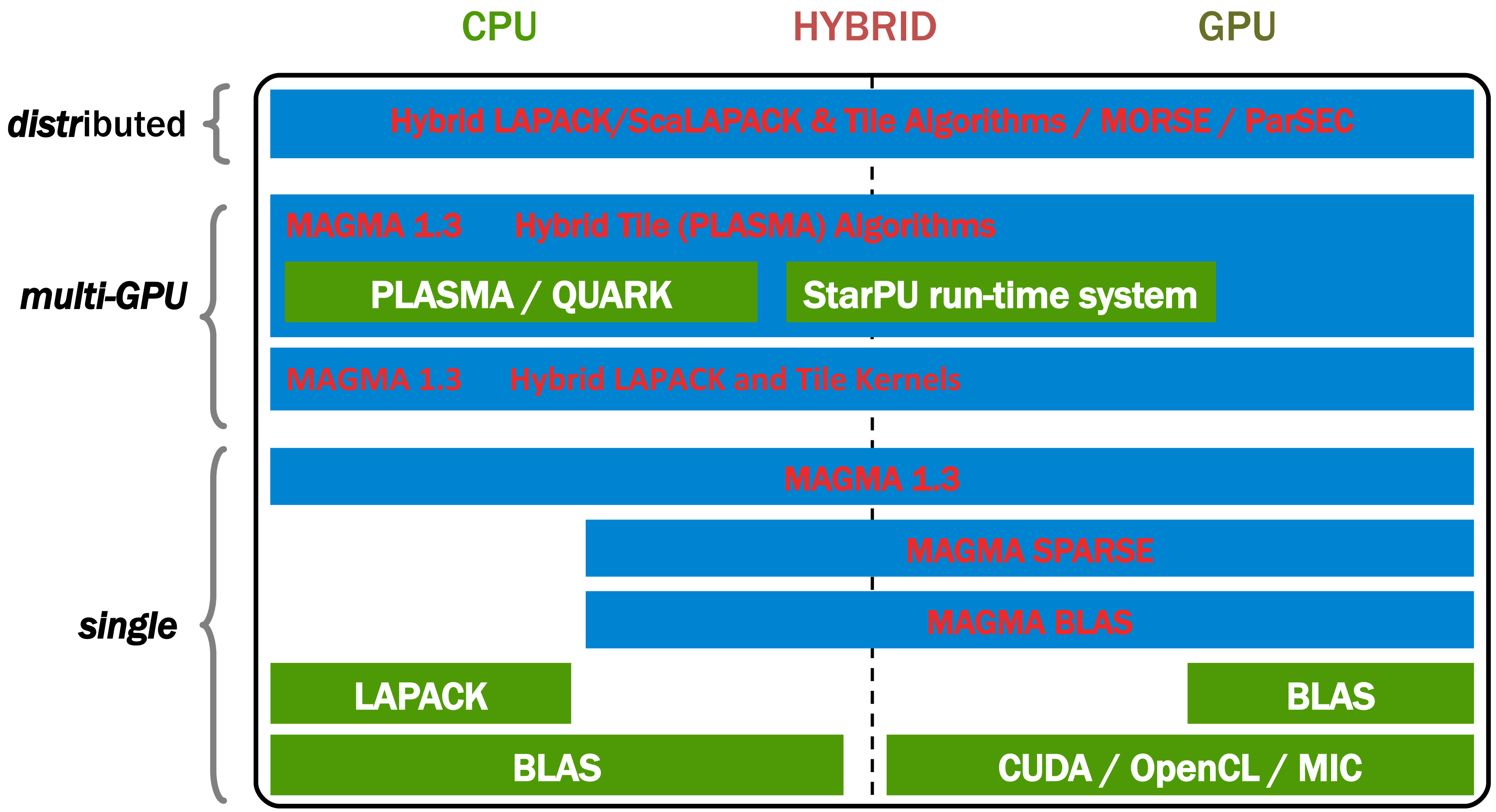
Piotr Luszczek (presenter)

web.eecs.utk.edu/~luszczek/conf/

MAGMA: LAPACK for GPUs

- MAGMA
  - Matrix Algebra for GPU and Multicore Architecture
  - To provide LAPACK/ScaLAPACK on hybrid architectures
  - http://icl.cs.utk.edu/magma/
- MAGMA BLAS
  - A subset of BLAS for GPUs
  - Highly optimized for NVIDIA GPGPUs
  - Fast GEMM for Fermi
- MAGMA developers & collaborators
  - UTK, UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  - Community effort, similar to LAPACK/ScaLAPACK

MAGMA Software Stack



- *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 BLAS
- A subset of GPU BLAS, optimized for Tesla and Fermi GPUs

MAGMA 1.3 ROUTINES & FUNCTIONALITIES	SINGLE GPU	MULTI-GPU STATIC	MULTI-GPU DYNAMIC
One-sided Factorizations (LU, QR, Cholesky)	✓	✓	✓
Linear System Solvers	✓		✓
Linear Least Squares (LLS) Solvers	✓		✓
Matrix Inversion	✓		✓
Singular Value Problem (SVP)	✓		
Non-symmetric Eigenvalue Problem	✓	✓	
Symmetric Eigenvalue Problem	✓	✓	
Generalized Symmetric Eigenvalue Problem	✓	✓	

SINGLE GPU
Hybrid LAPACK algorithms with static scheduling and LAPACK data layout

MULTI-GPU STATIC
Hybrid LAPACK algorithms with 1D block cyclic static scheduling and LAPACK data layout

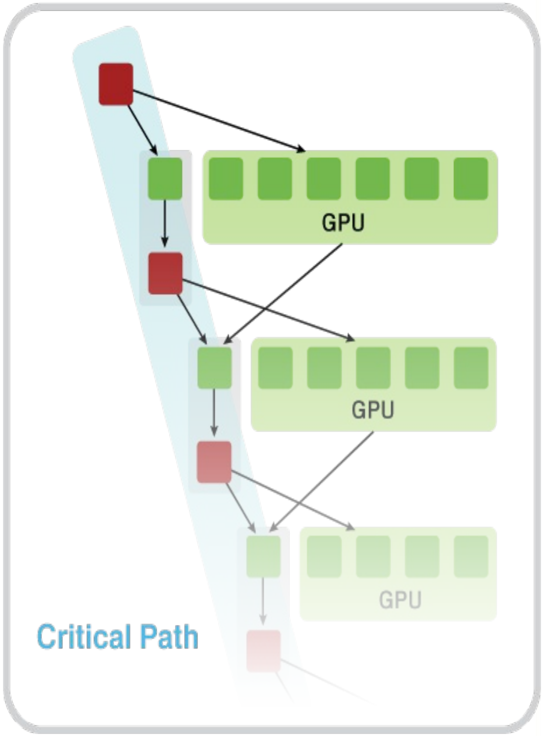
MULTI-GPU DYNAMIC
Tile algorithms with StarPU scheduling and tile matrix layout

MAGMA Methodology Overview

A methodology to use all available resources:

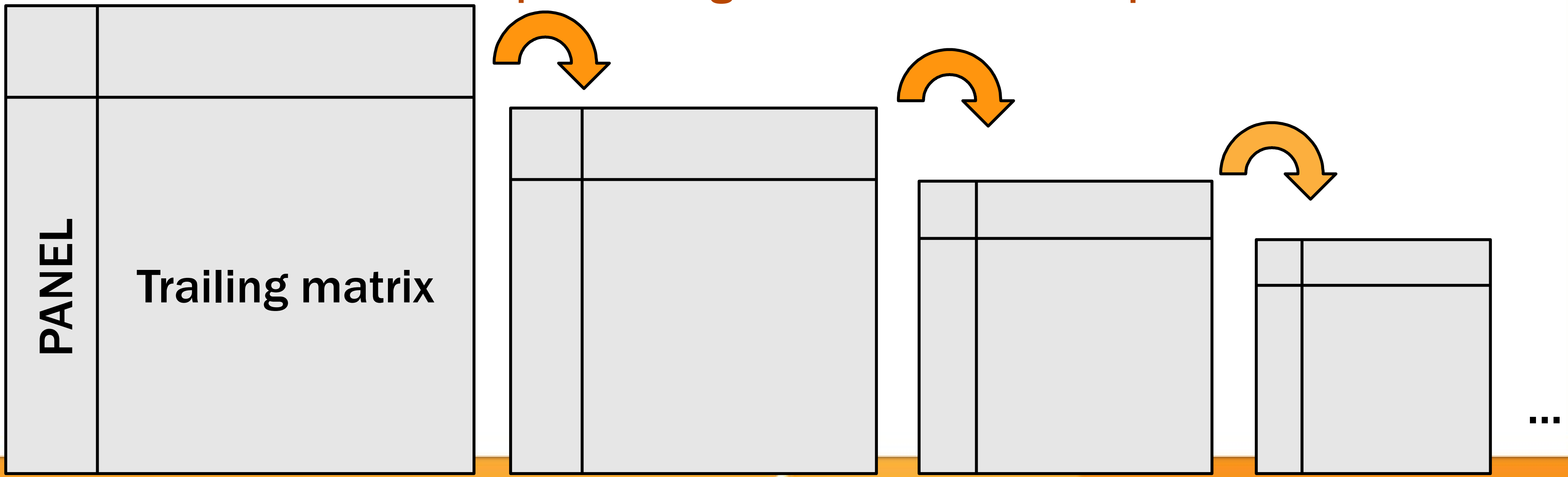
- MAGMA uses hybridization methodology based on
  - Representing linear algebra algorithms as collections of tasks and data dependencies among them
  - Properly scheduling tasks' execution over multicore and GPU hardware components
- Successfully applied to fundamental linear algebra algorithms
  - One- and two-sided factorizations and solvers
  - Iterative linear and eigensolvers
- Productivity
  - Use high-level description; low-level hidden with proper abstractions
  - Leverage prior efforts
  - Exceed the performance of homogeneous solutions

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



Hybrid Algorithms

- Use case: one-sided factorization
  - LU, QR, Cholesky
- Hybridization procedure
  - Panels are factored on CPU using LAPACK (or equivalent)
    - It is slow on the GPU
    - Off-load from GPU to CPU
  - Trailing matrix updates are done on the GPU
  - Look-ahead helps in hiding communication and panel factorization



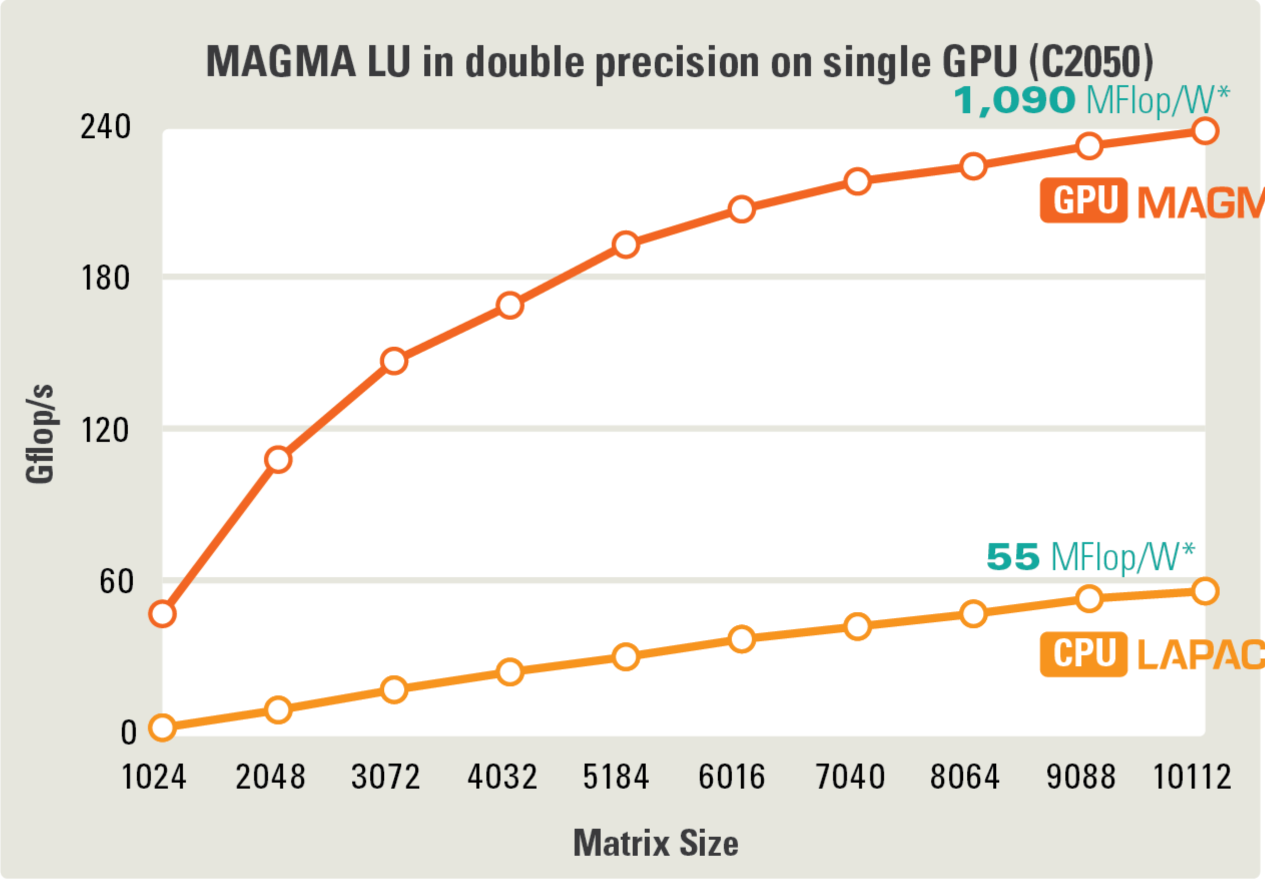
A Hybrid Algorithm Example

- Left-looking hybrid Cholesky factorization in MAGMA

```
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,
                              dA(0, j ), ldda, dA(0, j+jb), ldda, z_one,  dA(j, j+jb), ldda, queue );
7          magma_event_sync( event );
8          lapackf77_zpotrf( MagmaUpperStr, &jb, work, &jb, info );
9          if ( *info != 0 )
10              *info += j;
11         magma_zsetmatrix_async( jb, jb, work, 0, jb, dA(j,j), ldda, queue, &event );
12         if ( j+jb < n ) {
13              magma_event_sync( event );
14              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)

LU Factorization (single GPU)



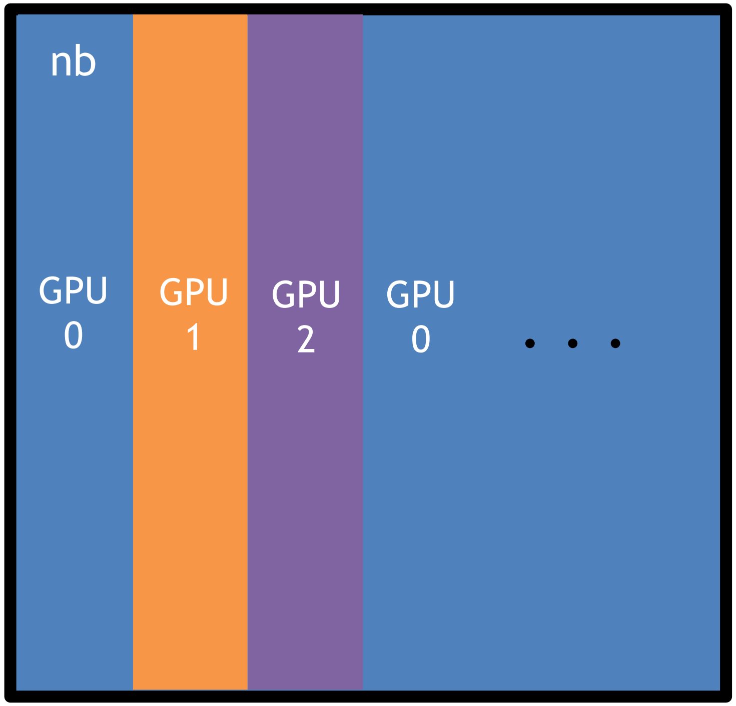
GPU Fermi C2050 (448 CUDA Cores @ 1.15 GHz)
+ Intel Q9300 (4 cores @ 2.50 GHz)
DP peak **515** + **40** GFlop/s
Power * ~**220** W

CPU AMD Istanbul
[8 sockets x 6 cores (48 cores) @2.8GHz]
DP peak **538** GFlop/s
Power * ~**1,022** W

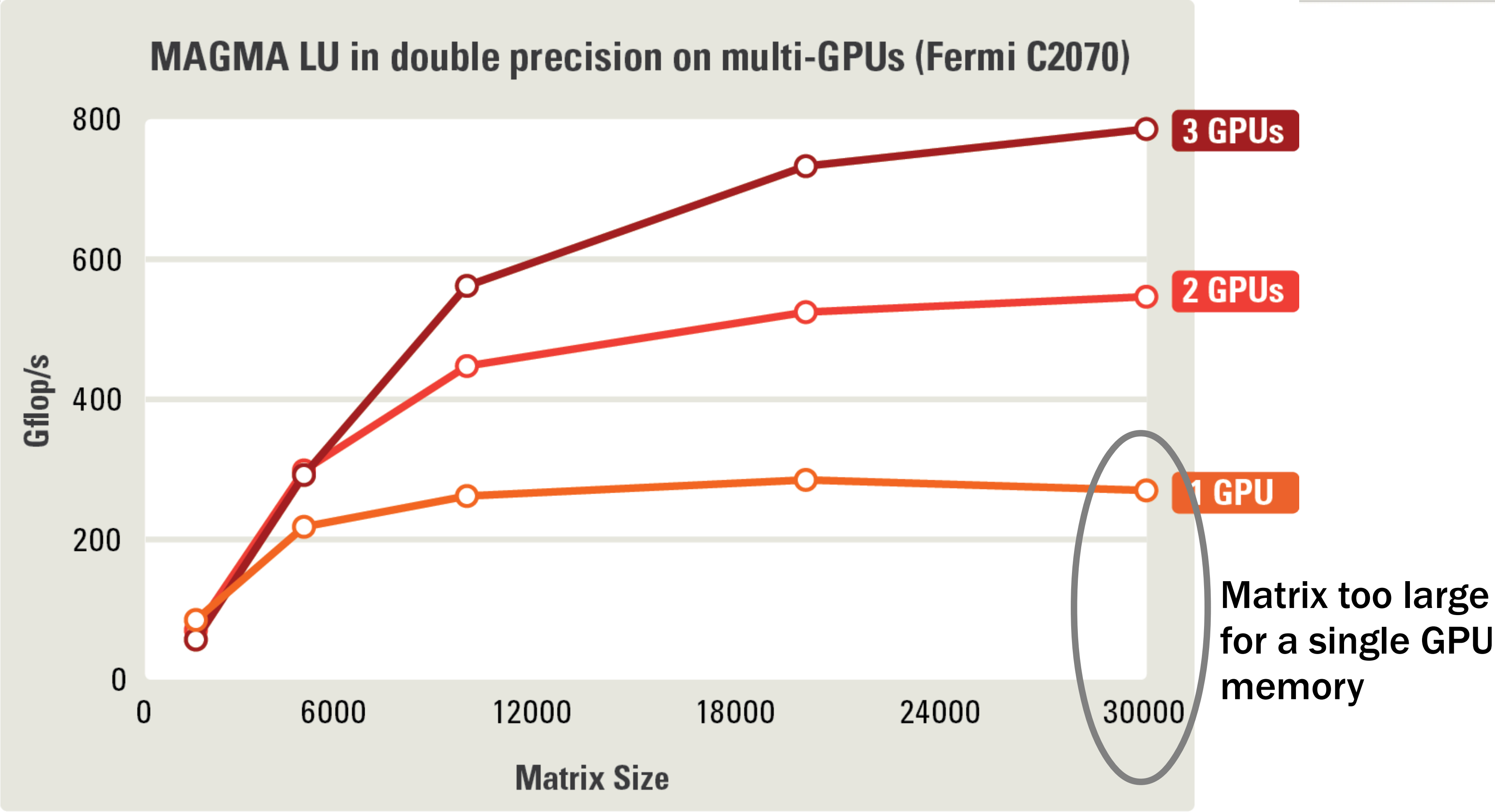
* Computation consumed power rate (total system rate minus idle rate), measured with KILL A WATT PS, Model P430

From Single to Multi-GPU 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 Factorization: Multiple GPUs



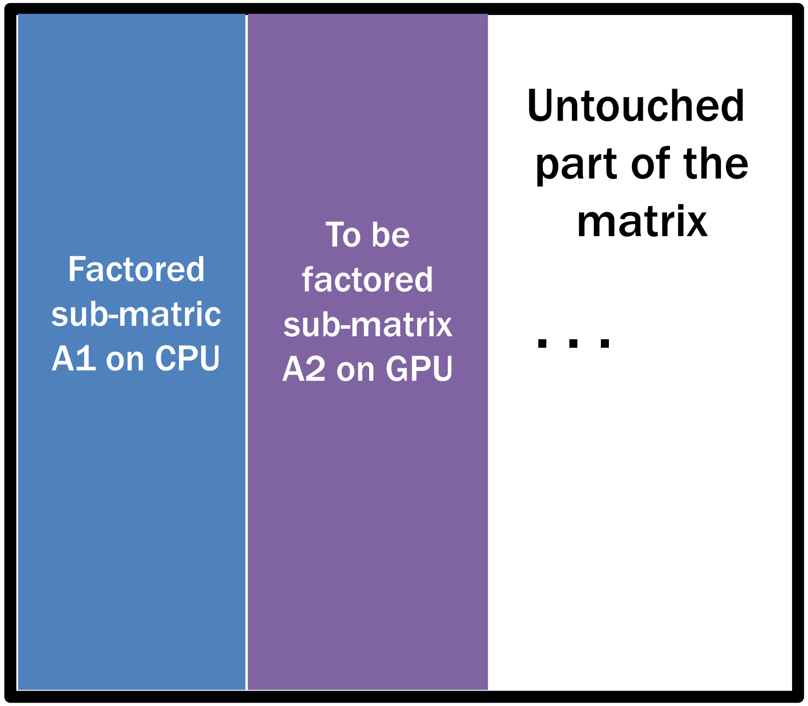
Keeneland system, using one node

3 NVIDIA GPUs (M2070 @ 1.1 GHz, 5.4 GB)

2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

Out of GPU Memory Algorithms

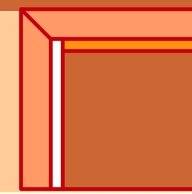
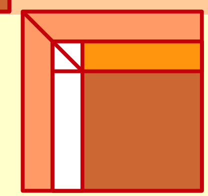
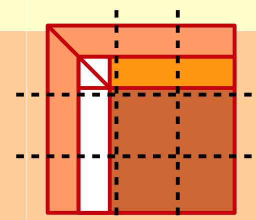
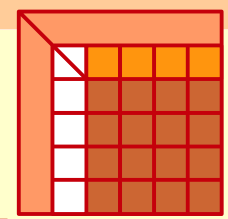
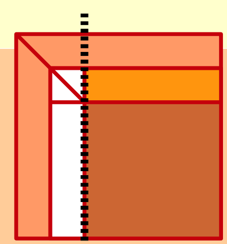
- Perform left-looking factorizations on sub-matrices that fit in the GPU memory (using existing algorithms)
- The rest of the matrix stays on the CPU
- Left-looking versions minimize writing on the CPU



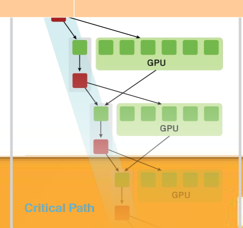
1) Copy A2 to the GPU
2) Update A2 using A1 (a panel of A1 at a time)
3) Factor the updated A2 using existing hybrid code
4) Copy factored A2 to the CPU

Trivially extended to multi-GPUs:
A2 is "larger" with 1-D block cyclic distribution, again reusing existing algorithms

A New Generation of DLA Software

Package	Era	Features	Concept	Abstractions
LINPACK	70's	Vector operations		Level-1 BLAS
LAPACK	80's	Blocking, cache friendly		Level-3 BLAS
ScaLAPACK	90's	Distributed memory		PBLAS, MPI
PLASMA	mid 00's	Multicore, Manycore		DAG scheduler, tile data layout, extra kernels
MAGMA	late 00's	Accelerated multicore		Hybrid scheduler, hybrid kernels

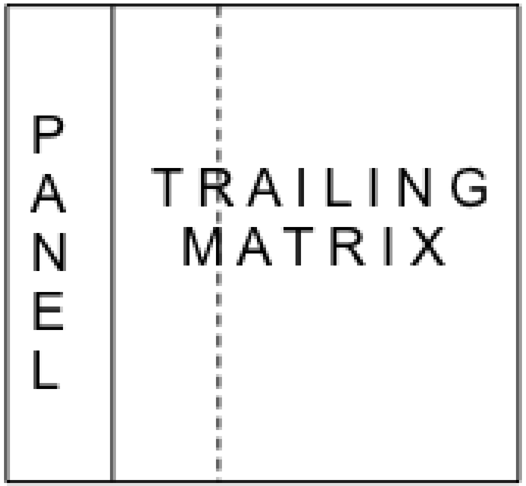
Hybrid Algorithms
(heterogeneity friendly)



Rely on
- hybrid scheduler
- hybrid kernels

Hybrid Algorithms: One-Sided Transformations

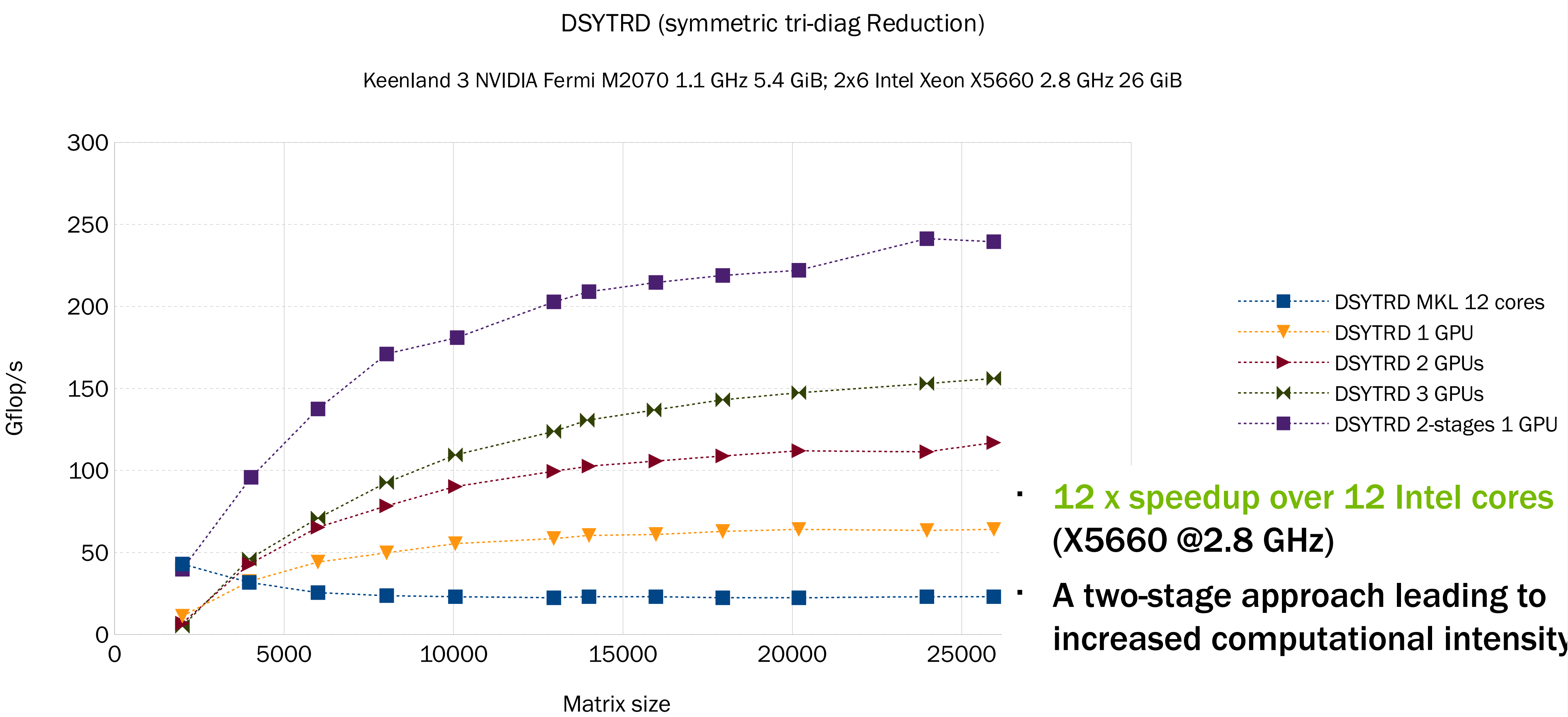
- One-Sided Factorizations
  - LU
  - QR, and
  - Cholesky
- Hybridization
  - Panels (Level 2 BLAS) are factored on CPU using LAPACK
  - Trailing matrix updates (Level 3 BLAS) are done on the GPU using “look-ahead”



Hybrid Algorithms: Two-Sided Transformations

- Two-Sided Factorizations
  - Bidiagonal — singular values
  - Tridiagonal — symmetric/generalized eigenvalues
  - Upper Hessenberg — non-symmetric eigenvalues
- Hybridization
  - Trailing matrix updates (Level 3 BLAS) are done on the GPU
    - Similar to the one-sided factorizations
  - Panels (Level 2 BLAS) are hybrid
    - Operations with memory footprint restricted to the panel are done on CPU
    - The time consuming matrix-vector products involving the entire trailing matrix are done on the GPU

Additional 4x Speedup from Faster GPU BLAS

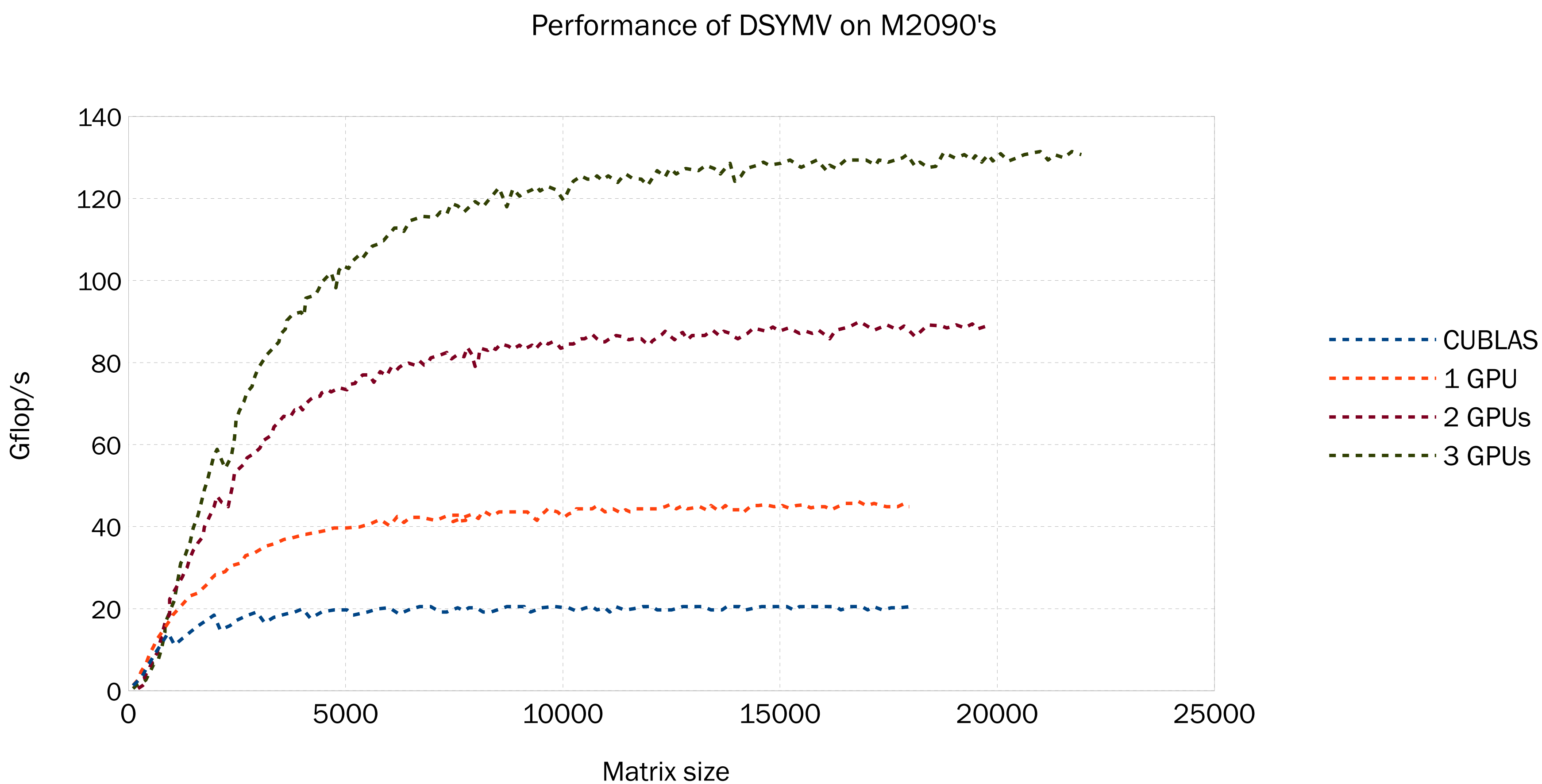


- **12 x speedup over 12 Intel cores (X5660 @2.8 GHz)**
- **A two-stage approach leading to increased computational intensity**

A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks*, ICL Technical report, 03/2012.

Multi-GPU Two-Sided Factorizations

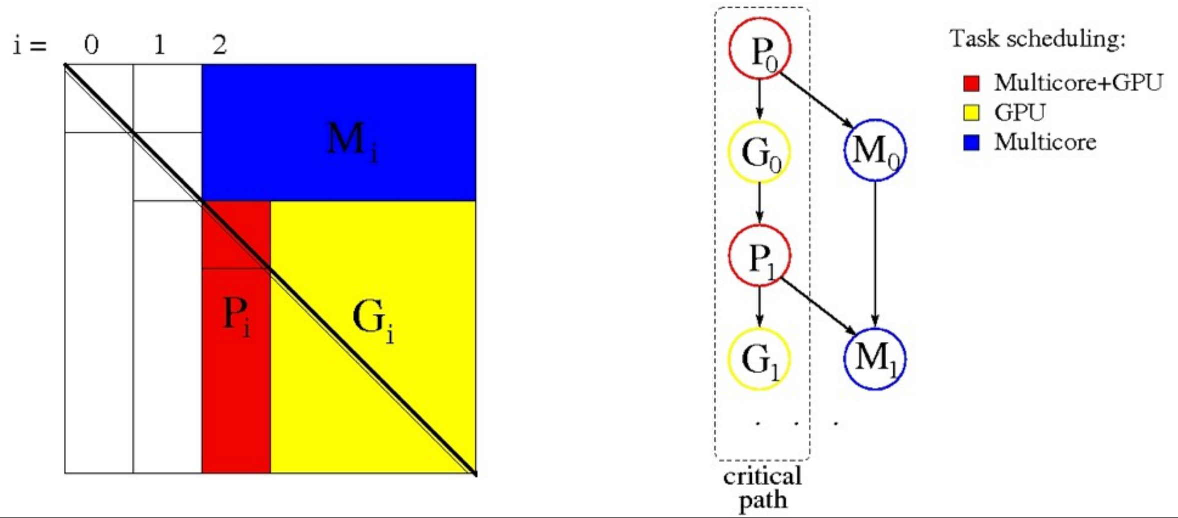
- Need HPC multi-GPU Level 2 BLAS (e.g., 50% of flops in the

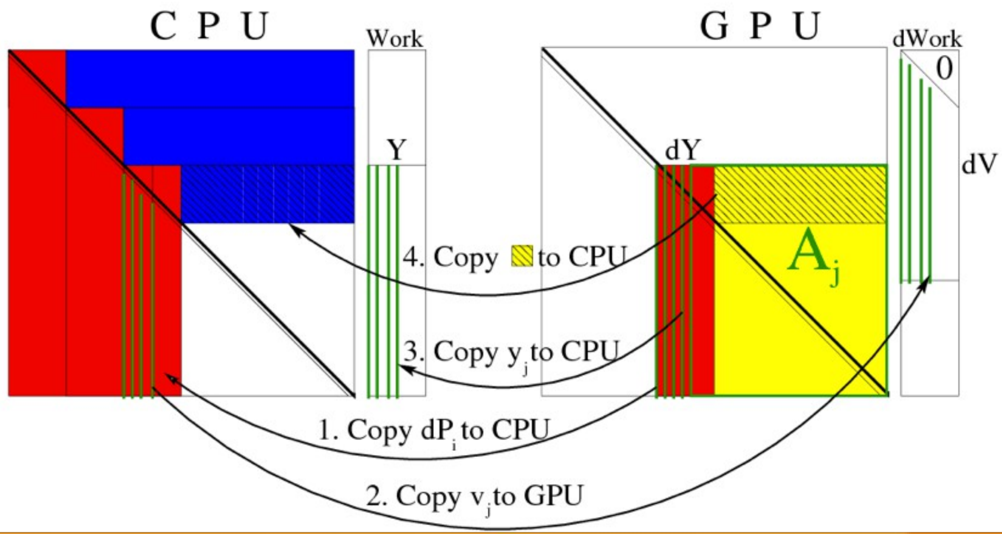


T. Dong, J. Dongarra, S. Tomov, I. Yamazaki, T. Schulthess, and R. Solca, *Symmetric dense matrix-vector multiplication on multiple GPUs and its application to symmetric dense and sparse eigenvalue problems*, ICL Technical report, 03/2012.

Hybrid Two-Sided Factorizations

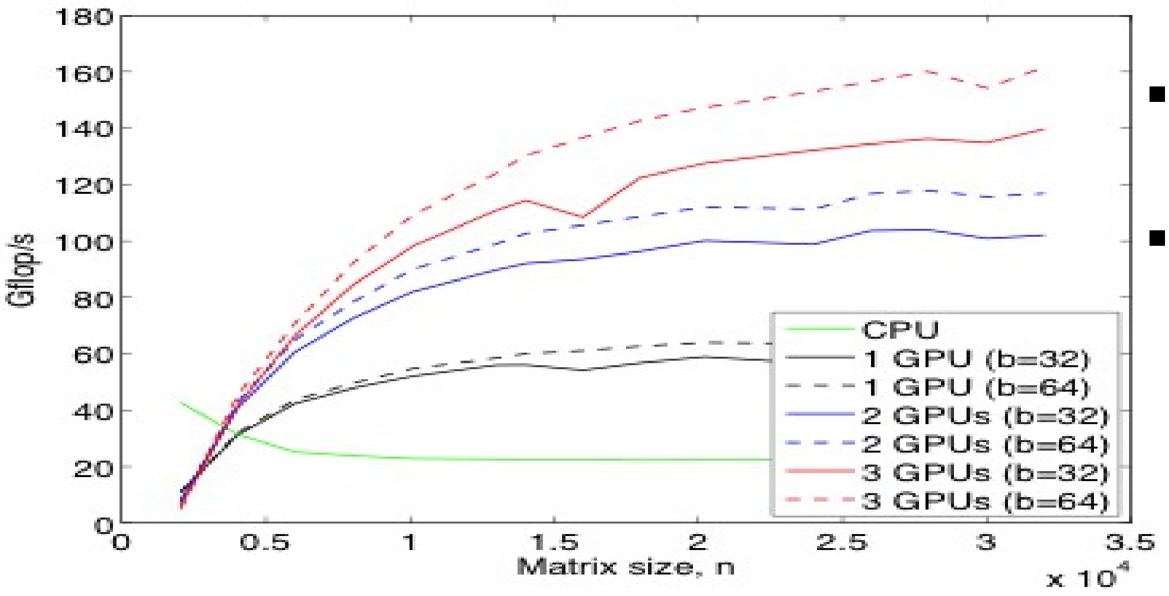
Task **Splitting** & Task **Scheduling**





From Fast BLAS to Fast Tridiagonalization

Performance of MAGMA DSYTRD on multi M2090 GPUs



- 50 % of the flops are in SYMV
- Memory bound, i.e. does not scale well on multicore CPUs
- Use the GPU's high memory bandwidth and optimized SYMV
- 8 x speedup over 12 Intel cores (X5660 @2.8 GHz)

Keeneland system, using one node
3 NVIDIA GPUs (M2070@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

T. Dong, J. Dongarra, S. Tomov, I. Yamazaki, T. Schulthess, and R. Solca, *Symmetric dense matrix-vector multiplication on multiple GPUs and its application to symmetric dense and sparse eigenvalue problems*, ICL Technical report, 03/2012.

From Static to Dynamic Scheduling …

- Static may stall in situations where work is available
- Hand tuned optimizations
- Hardware heterogeneity
- Kernel heterogeneity
- Separation of concerns
- Dynamic Runtime System

Matrices Over Runtime Systems at Exascale

- MORSE
- Mission statement:
  - "Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"
- Runtime challenges due to the ever growing hardware complexity
- Algorithmic challenges to exploit the hardware capabilities to the fullest
- Integrated into MAGMA software stack

MAGMA-MORSE: x86 + Multiple GPUs

- Lessons Learned from PLASMA
- New high performance numerical kernels
- StarPU Runtime System
  - Augonnet et. Al, INRIA, Bordeaux
- Use of both: x86 and GPUs leads to Hybrid Computations
- Similar to LAPACK in functionality

High Productivity: Sequential Code

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 Productivity: Parallel Code

From Sequential Nested-Loop Code to Parallel Execution

```
for (k = 0; k < min(MT, NT); k++) {

    starPU_Insert_Task( &cl_zgeqrt, A, k, k, ...);

    for (n = k+1; n < NT; n++)
        starPU_Insert_Task( &cl_zunmqr( A, k, n, ...);

    for (m = k+1; m < MT; m++) {

        starPU_Insert_Task( &cl_ztsqrt( A m, k, ...);

        for (n = k+1; n < NT; n++)

            starPU_Insert_Task( &cl_ztsmqr( A, m, n, k, ...);

    }

}
```

Contact Information and Generous Sponsors

Stan Tomov

tomov@eecs.utk.edu

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