# Matrix Algebra on GPU and Multicore Architectures

Stan Tomov and Jack Dongarra
Research Director
Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

Workshop on "*Electronic Structure Calculation Methods on Accelerators*"
Oak Ridge National Laboratory, TN
February 5-8, 2012

# The Need for HP Linear Algebra

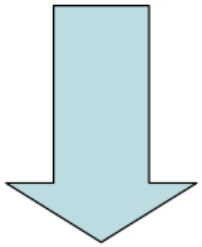
## Electronic structure calculations

- Density functional theory

  Many-body Schrödinger equation (exact but exponential scaling)

$$\{-\sum_i \frac{1}{2}\nabla_i^2 + \sum_{i,j} \frac{1}{|r_i - r_j|} + \sum_{i,I} \frac{Z}{|r_i - R_I|}\}\Psi(r_1,..r_N) = E\Psi(r_1,..r_N)$$

  · Nuclei fixed, generating external potential (system dependent, non-trivial)
  · N is number of electrons



**Kohn Sham Equation: The many body problem of interacting electrons is reduced to non-interacting electrons (single particle problem) with the same electron density and a different effective potential (cubic scaling).**

$$\{-\frac{1}{2}\nabla^2 + \int \frac{\rho(r')}{|r - r'|}dr' + \sum_I \frac{Z}{|r - R_I|} + V_{XC}\}\psi_i(r) = E_i\psi_i(r)$$

$$\rho(r) = \sum_i |\psi_i(r)|^2 = |\Psi(r_1,..r_N)|^2$$

· $V_{XC}$ represents effects of the Coulomb interactions between electrons

· $\rho$ is the density (of the original many-body system)

$V_{XC}$ is not known except special cases ⇒ use approximation, e.g. Local Density Approximation (LDA)

where $V_{XC}$ depends only on $\rho$

- A model leading to self-consistent iteration computation with need for HP LA (e.g, **diagonalization** and **orthogonalization)**

# The Need for HP Linear Algebra

## From Schodinger equation to eigenvalue solver

- Schodinger equation:

$$H\psi = E\psi$$

- Choose a basis set of wave functions
- Two cases:
  - Orthonormal basis:

$$H\,x = E\,x$$

    in general it needs a big basis set
  - Non-orthonormal basis:

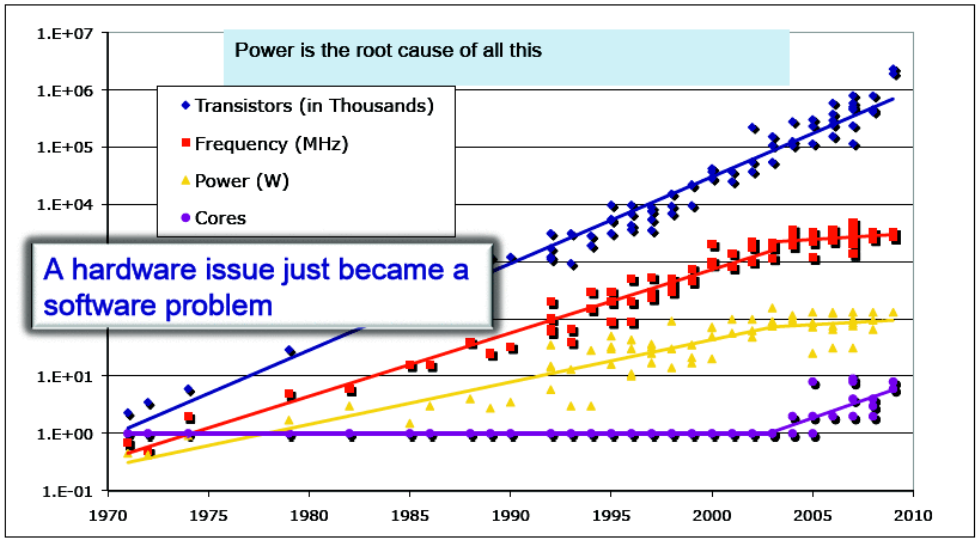
$$H\,x = E\,S\,x$$

    where S is the overlap matrix.

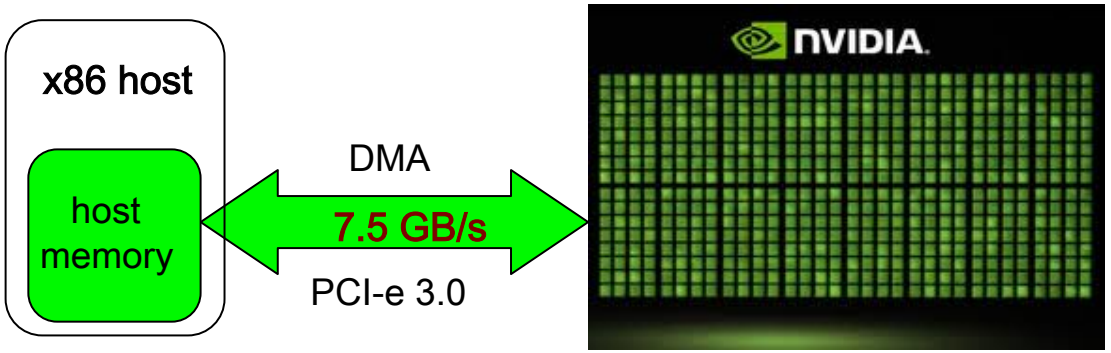
# Hardware Trends

- Power consumption and the move towards multicore
- Hybrid architectures
- GPU
- Hybrid GPU-based systems
  - CPU and GPU to get integrated (NVIDIA to make ARM CPU cores alongside GPUs)

## Performance Has Also Slowed, Along with Power

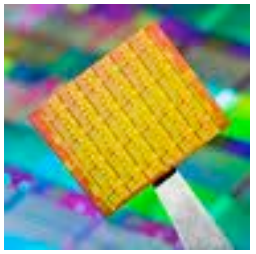



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

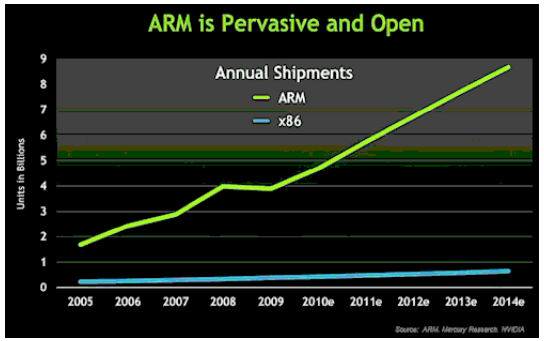


# Future Computer Systems

- Most likely be a hybrid design
  - Think standard multicore chips and accelerator (GPUs)
- Today accelerators are attached
- Next generation more integrated
- Intel's MIC architecture "Knights Ferry" and "Knights Corner" to come.
  - 48 x86 cores
- AMD's Fusion in 2012 - 2013
  - Multicore with embedded graphics ATI
- Nvidia's Project Denver plans to develop an integrated chip using ARM architecture in 2013.



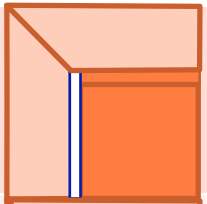
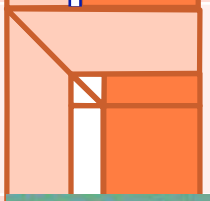
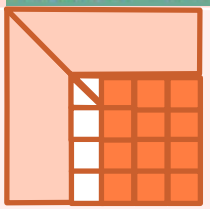
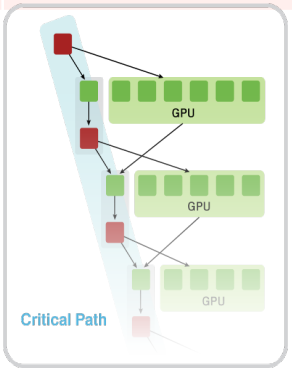
AMD
The future is fusion



# Major change to Software

- **Must rethink the design of our software**
  - 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

# A New Generation of Software

| Software/Algorithms follow hardware evolution in time | | |
|---|---|---|
| LINPACK (70's)<br>(Vector operations) |  | Rely on<br>- Level-1 BLAS operations |
| LAPACK (80's)<br>(Blocking, cache friendly) |  | Rely on<br>- Level-3 BLAS operations |
| ScaLAPACK (90's)<br>(Distributed Memory) | | Rely on<br>- PBLAS Mess Passing |
| PLASMA (00's)<br>New Algorithms<br>(many-core friendly) |  | Rely on<br>- a DAG/scheduler<br>- block data layout<br>- some extra kernels |
| MAGMA<br>Hybrid Algorithms<br>(heterogeneity friendly) |  | Rely on<br>- hybrid scheduler (of DAGs)<br>- hybrid kernels<br>(for nested parallelism)<br>- existing software infrastructure |

# Challenges of using GPUs

- **High levels of parallelism**
  - **Many GPU cores**
  - [ e.g. Tesla C2050 (Fermi) has 448 CUDA cores ]
- **Hybrid/heterogeneous architectures**
  - **Match algorithmic requirements to architectural strengths**
  - [ e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU ]
- **Compute *vs* communication gap**
  - **Exponentially growing gap; persistent challenge**
  - [ Processor speed improves 59%, memory bandwidth 23%, latency 5.5% ]
  - [ on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection ]

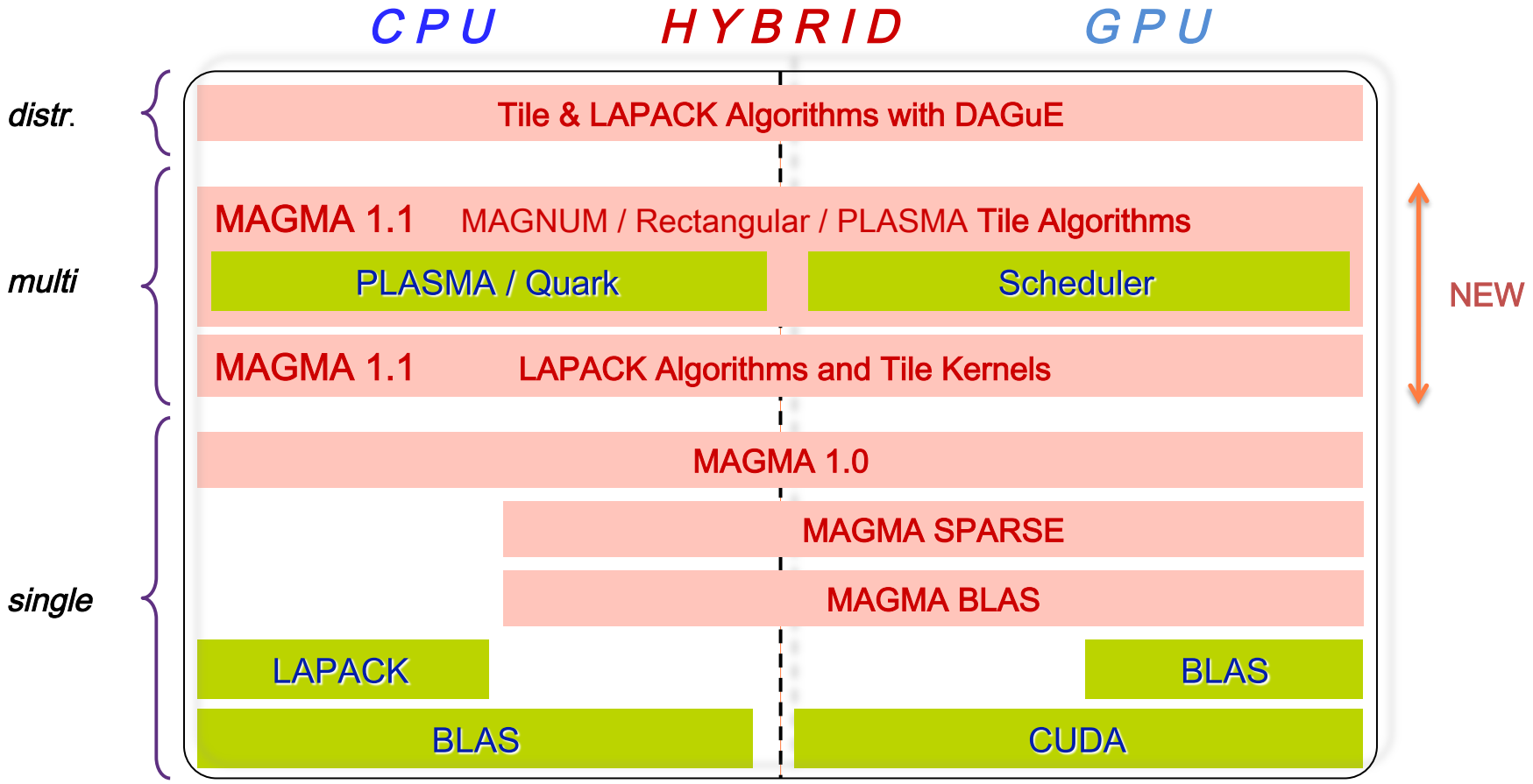
# Matrix Algebra on GPU and Multicore Architectures (MAGMA)

- MAGMA: **a new generation linear algebra (LA) libraries** to achieve the fastest possible time to an accurate solution **on hybrid/heterogeneous architectures**
  Homepage: http://icl.cs.utk.edu/magma/

- MAGMA & LAPACK
  - **MAGMA** uses LAPACK and extends its functionality to hybrid systems (w/ GPUs);
  - **MAGMA** is designed to be similar to LAPACK in functionality, data storage and interface
  - **MAGMA** leverages years of experience in developing open source LA software packages like LAPACK, ScaLAPACK, BLAS, ATLAS, and PLASMA

- MAGMA developers/collaborators
  - U of Tennessee, **Knoxville**; U of California, **Berkeley**; U of Colorado, **Denver**
  - INRIA Bordeaux - Sud Ouest & INRIA Paris – Saclay, France; KAUST, Saudi Arabia
  - **Community effort** [similarly to the development of LAPACK / ScaLAPACK]

# MAGMA Software Stack




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

# MAGMA 1.1

- **50+ hybrid LAPACK algorithms** have been developed (total of 200+ 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
- Support is for CUDA-enabled NVIDIA GPUs
- MAGMA BLAS
  - A subset of GPU BLAS, optimized for Tesla and Fermi GPUs

| MAGMA 1.1 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 1.1

## COMPUTATIONAL ROUTINES IN MAGMA 1.1

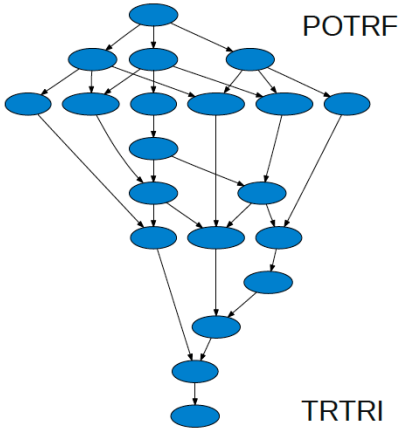
| | MATRIX | OPERATION | ROUTINE | INTERFACES CPU | INTERFACES GPU |
|---|---|---|---|---|---|
| LINEAR EQUATIONS | GE | LU | {sdcz}getrf | ✓ | ✓ |
| | | Solve | {sdcz}getrs | | ✓ |
| | | Invert | {sdcz}getri | | ✓ |
| | SPD/HPD | Cholesky | {sdcz}potrf | ✓ | ✓ |
| | | Solve | {sdcz}potrs | | ✓ |
| | | Invert | {sdcz}potri | | ✓ |
| | TR | Invert | {sdcz}trtri | ✓ | |
| ORTHOGONAL FACTORIZATIONS | GE | QR | {sdcz}geqrf | ✓ | |
| | | Generate Q | {sd}orgqr | ✓ | ✓ |
| | | | {cz}ungqr | ✓ | ✓ |
| | | Multiply matrix by Q | {sd}ormqr | ✓ | ✓ |
| | | | {cz}unmqr | ✓ | ✓ |
| | GE | LQ factorization | {sdcz}gelqf | ✓ | ✓ |
| | | QL factorization | {sdcz}geqlf | ✓ | |
| | | Multiply matrix by Q | {sd}ormql | ✓ | ✓ |
| | | | {cz}unmql | ✓ | ✓ |
| STANDARD EVP | GE | Hessenberg reduction | {sdcz}gehrd | ✓ | |
| | | Generate Q | {sd}orghr | ✓ | |
| | | | {cz}unghr | ✓ | |
| | SY/HE | Tridiagonalization | {sd}sytrd | ✓ | |
| | | | {cz}hetrd | ✓ | |
| | | Generate Q | {sd}orgtr | | ✓ |
| | | | {cz}ungtr | | ✓ |
| | | Multiply by Q | {sd}ormtr | ✓ | ✓ |
| | | | {cz}unmtr | ✓ | ✓ |
| SVD | GE | Bidiagonalization | {sdcz}gebrd | ✓ | |
| GENER-ALIZED EVP | SPD/HPD | Reduction to standard form | {sd}sygst | ✓ | ✓ |
| | | | {cz}hegst | ✓ | ✓ |

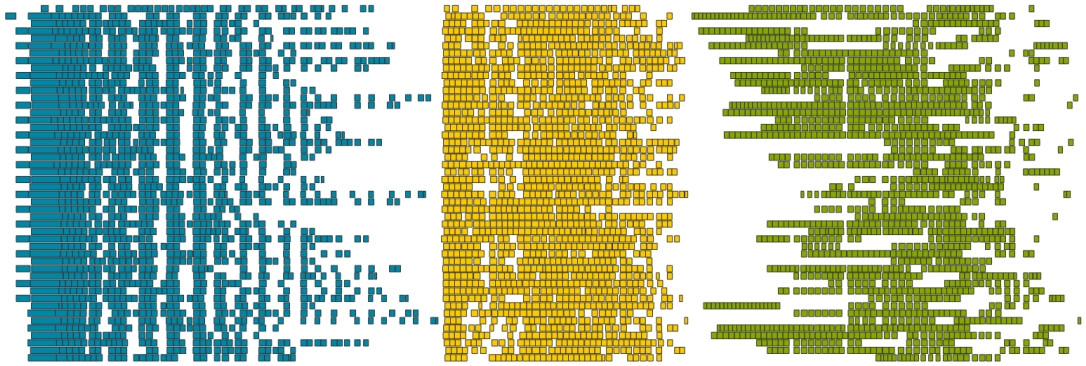
# MAGMA 1.1

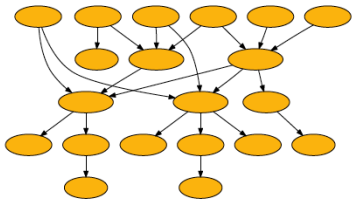
## DRIVER ROUTINES IN MAGMA 1.1

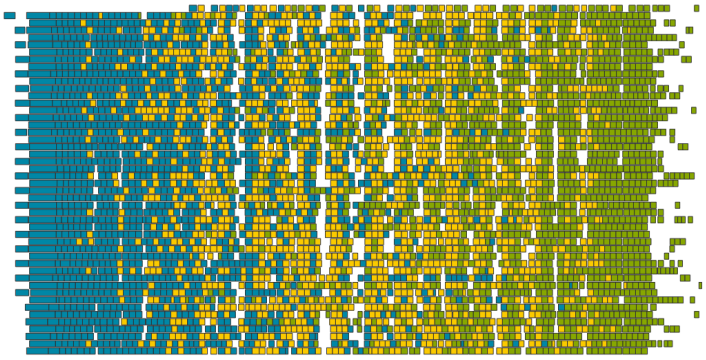
| | MATRIX | OPERATION | ROUTINE | INTERFACES CPU | INTERFACES GPU |
|---|---|---|---|---|---|
| LINEAR EQUATIONS | GE | Solve using LU | {sdcz}gesv | ✓ | ✓ |
| | | Solve using MP | {zc,ds}gesv | | ✓ |
| | SPD/HPD | Solve using Cholesky | {sdcz}posv | ✓ | ✓ |
| | | Solve using MP | {zc,ds}posv | | ✓ |
| LLS | GE | Solve LLS using QR | {sdcz}geqrs | | ✓ |
| | | Solve using MP | {zc,ds}geqrsv | | ✓ |
| STANDARD EVP | GE | Compute e-values, optionally e-vectors | {sdcz}geev | ✓ | |
| | SY/HE | Computes all e-values, | {sd}syevd | ✓ | ✓ |
| | | optionally e-vectors | {cz}heevd | ✓ | ✓ |
| | | Range ( D&C ) | {cz}heevdx | | ✓ |
| | | Range ( B & I It. ) | {cz}heevx | ✓ | ✓ |
| | | Range ( MRRR ) | {cz}heevr | ✓ | ✓ |
| STAND. SVP | GE | Compute SVD, | {sdcz}gesvd | ✓ | |
| | | optionally s-vectors | | ✓ | |
| GENERALIZED EVP | SPD/HPD | Compute all e-values, | {sd}sygvd | ✓ | |
| | | optionally e-vectors | {cz}hegvd | ✓ | |
| | | Range ( D&C ) | {cz}hegvdx | ✓ | |
| | | Range ( B & I It.) | {cz}hegvx | ✓ | |
| | | Range ( MRRR ) | {cz}hegvr | ✓ | |

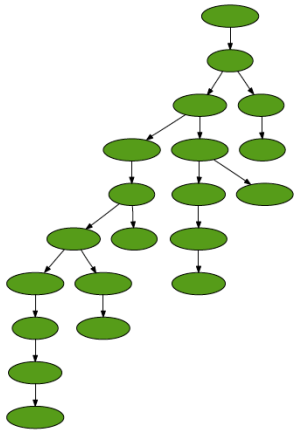
# Scheduling Algorithms as DAGs



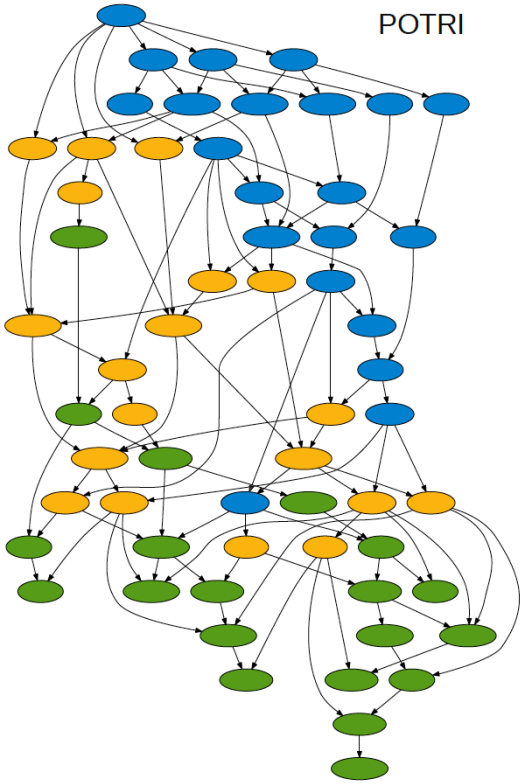





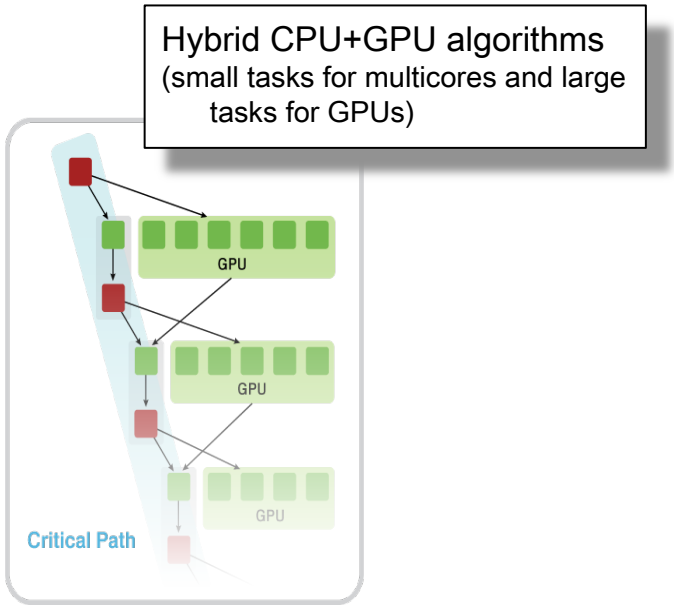



48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,



# Methodology overview

- **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**
  - High-level
  - Leveraging prior developments
  - Exceeding in performance homogeneous solutions



# Statically Scheduled **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"
- **Note**
  - Panels are memory bound but are only $O(N^2)$ flops and can be overlapped with the $O(N^3)$ flops of the updates
  - In effect, the GPU is used only for the high-performance Level 3 BLAS updates,
    i.e., no low performance Level 2 BLAS is scheduled on the GPU

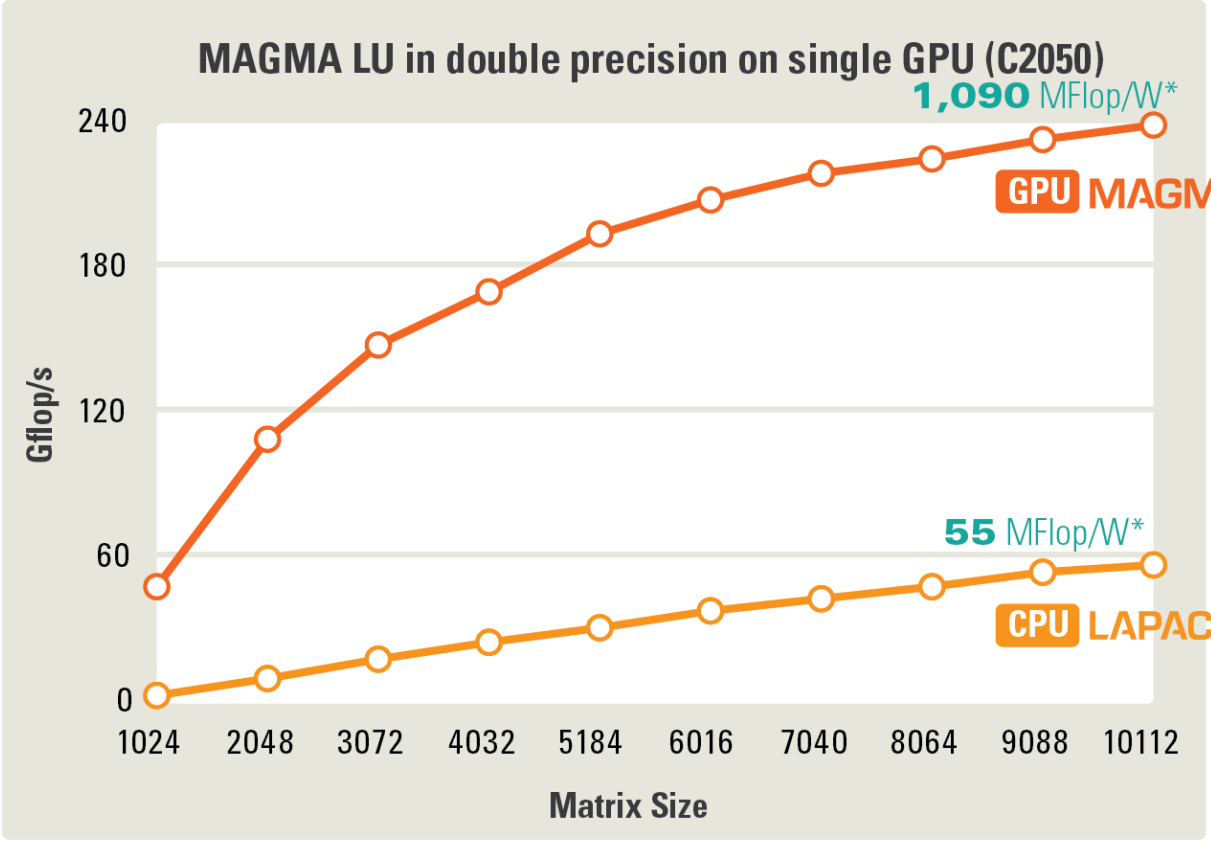
# A hybrid algorithm example

- Left-looking hybrid Cholesky factorization in MAGMA 1.0

```
1  for (j = 0;  j < *n; j += nb) {
2    jb = min(nb, *n-j);
3    cublasSsyrk('l','n', jb, j, -1, da(j,0),*lda, 1, da(j,j),*lda);
4    cudaMemcpy2DAsync(work, jb*sizeof(float), da(j,j), *lda*sizeof(float),
5                      sizeof(float)*jb, jb, cudaMemcpyDeviceToHost, stream[1]);
6    if (j + jb < *n)
7       cublasSgemm('n','t', *n-j-jb, jb, j, -1, da(j+jb,0), *lda, da(j,0),
8                    *lda, 1, da(j+jb,j), *lda);
9    cudaStreamSynchronize(stream[1]);
10   spotrf_("Lower", &jb, work, &jb, info);
11   if (*info != 0)
12      *info = *info + j, break;
13   cudaMemcpy2DAsync(da(j,j), *lda*sizeof(float), work, jb*sizeof(float),
14                     sizeof(float)*jb, jb, cudaMemcpyHostToDevice, stream[0]);
15   if (j + jb < *n)
16      cublasStrsm('r','l','t','n', *n-j-jb, jb, 1, da(j,j), *lda,
17                   da(j+jb,j), *lda);
18 }
```

- The difference with LAPACK - the 3 additional lines in red
- Line 10 (done on CPU) is overlapped with work on the GPU (line 7)

# MAGMA Performance (single GPU)

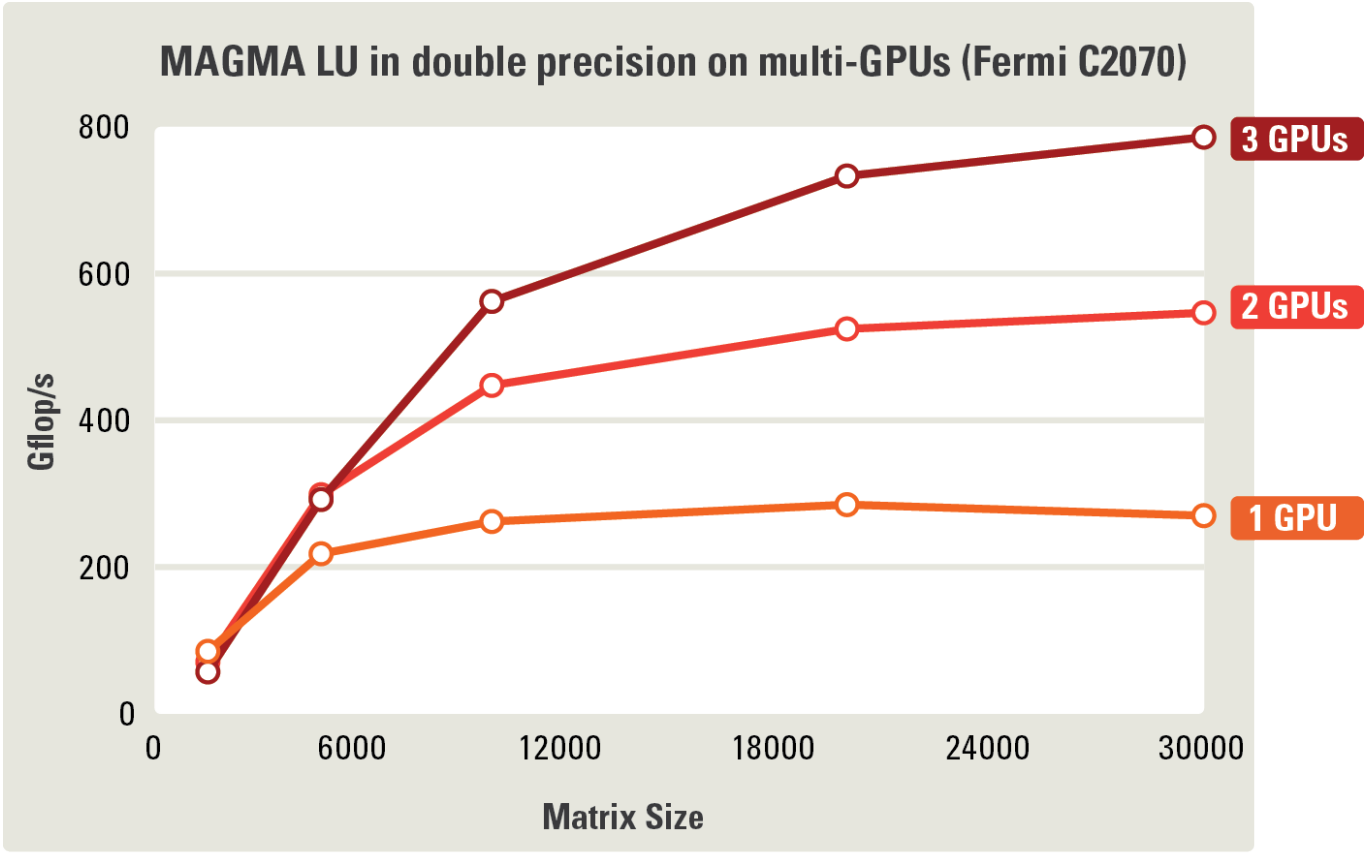



**GPU** Fermi C2050 (448 CUDA Cores @ 1.15 GHz)
+ Intel Q9300 (4 cores @ 2.50 GHz)
DP peak **515** + **40** GFlop/s
Power * ~**220** W

**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

# MAGMA Performance (scaling)




**Keeneland system, using one node**

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

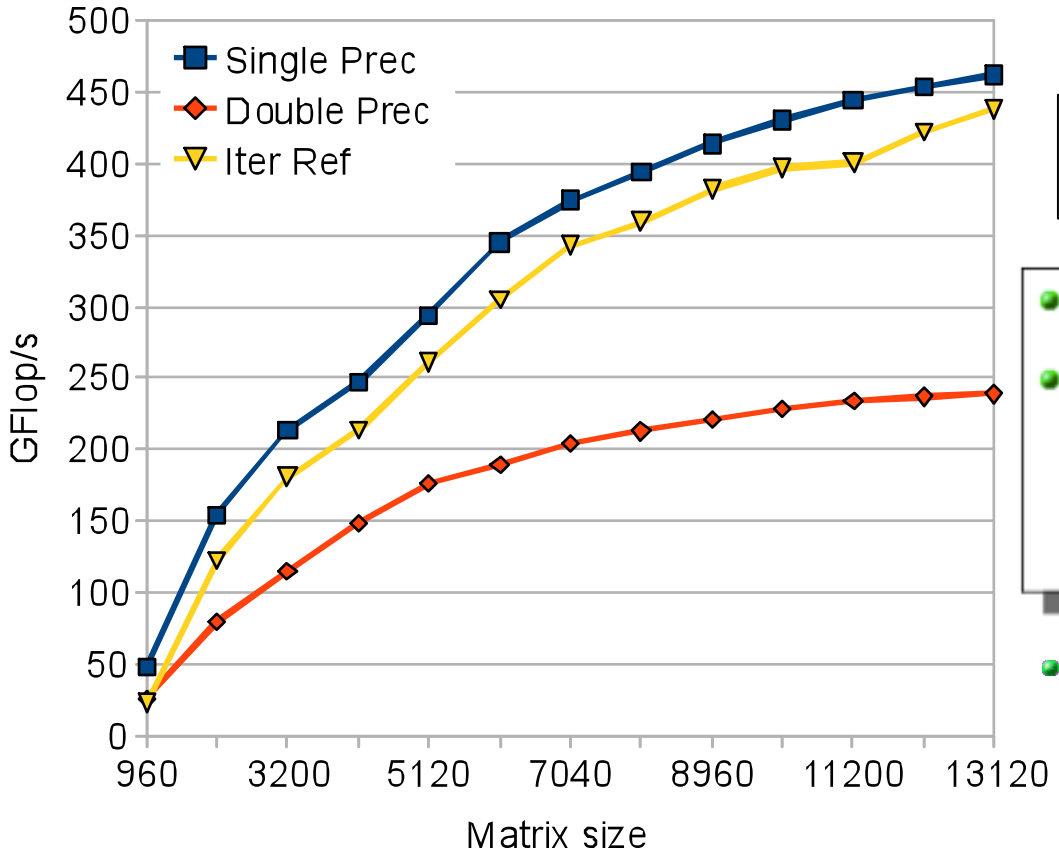
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

# Mixed Precision Methods

- Mixed precision, use the lowest precision required to achieve a given accuracy outcome
  - Improves runtime, reduce power consumption, lower data movement
  - Reformulate to find correction to solution, rather than solution [ $\Delta x$ rather than $x$ ].

# Results – linear solvers

MAGMA LU-based solvers on Fermi (C2050)




**FERMI** Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 GFlop/s

- **Direct solvers**
  - Factor and solve in working precision
- **Mixed Precision Iterative Refinement**
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g.

$$x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$$

- Similar results for Cholesky & QR

# Two-sided matrix factorizations

- Two-sided factorizations
  $Q' A Q = H$ , $H$ – upper Hessenberg / tridiagonal,
  $Q' A P = B$ , $B$ – bidiagonal
  ***Q and P*** – orthogonal similarity transformations

- **Importance**

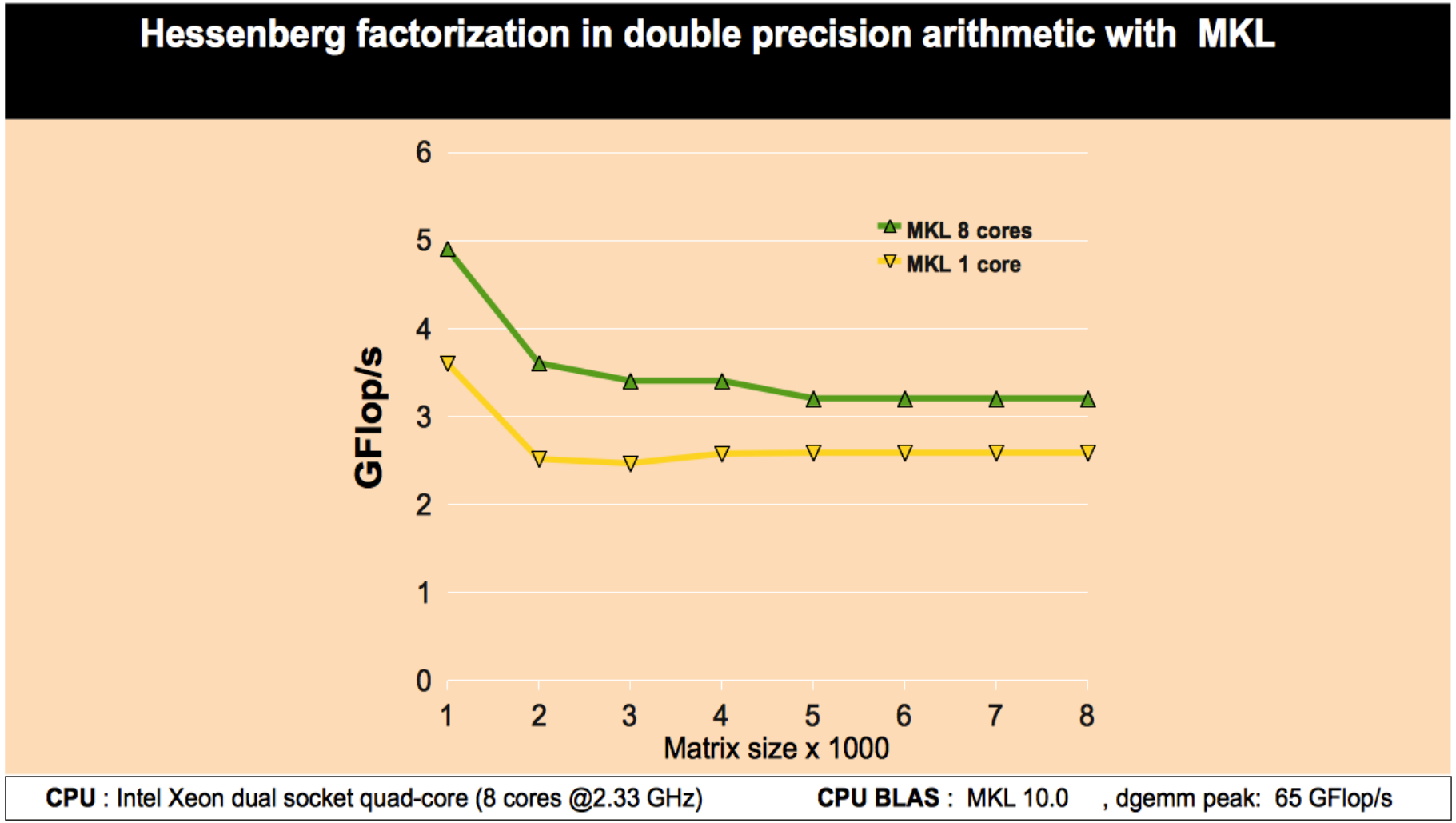
  **One-sided factorizations**
  - bases for linear solvers

  **Two-sided factorizations**
  - bases for eigen-solvers

- Block algorithm
  ***Q*** – a product of n-1 elementary reflectors
  $Q = H_1 H_2 \ldots H_{n-1}$, $H_i = I - \tau_i v_i v_i'$
  $H_1 \ldots H_{nb} = I - V T V'$ (*WY transform*; the bases for delayed update or block algorithm)

- **Can we accelerate it ?**
  [similarly to the one-sided using hybrid GPU-based computing]
  [ to see **much higher acceleration** due to a removed bottleneck ]

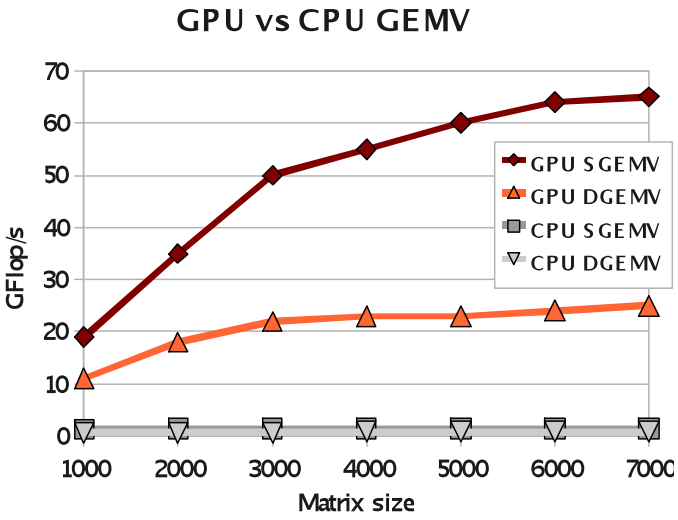
# Homogeneous multicore acceleration?




- There have been difficulties in accelerating it on homogeneous multicores

# Two-sided matrix factorizations

- Used in singular-value and eigen-value problems
- LAPACK-based two-sided factorizations are rich in Level 2 BLAS and therefore can not be properly accelerated on multicore CPUs
- We developed hybrid algorithms exploring GPUs' high bandwidth



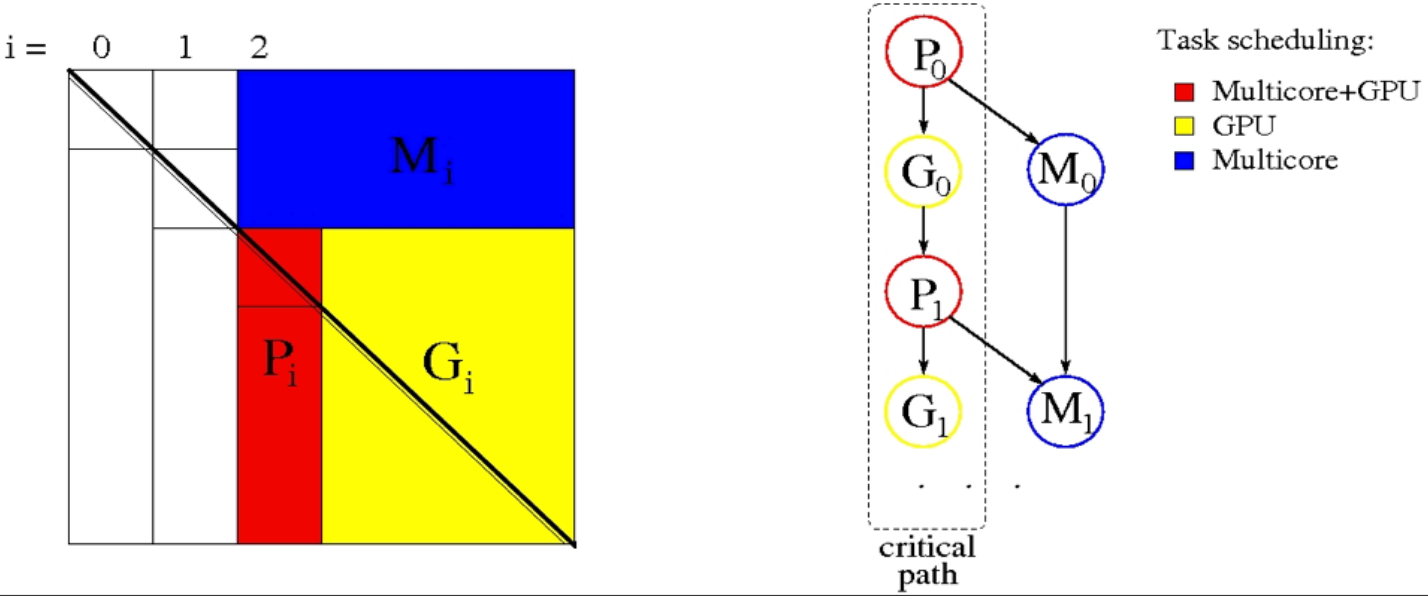

**GPU**: GTX280 (240 cores @ 1.30GHz, 141 GB/s)
**CPU**: 2 x 4 cores Intel Xeon @ 2.33GHz, 10.4 GB/s)

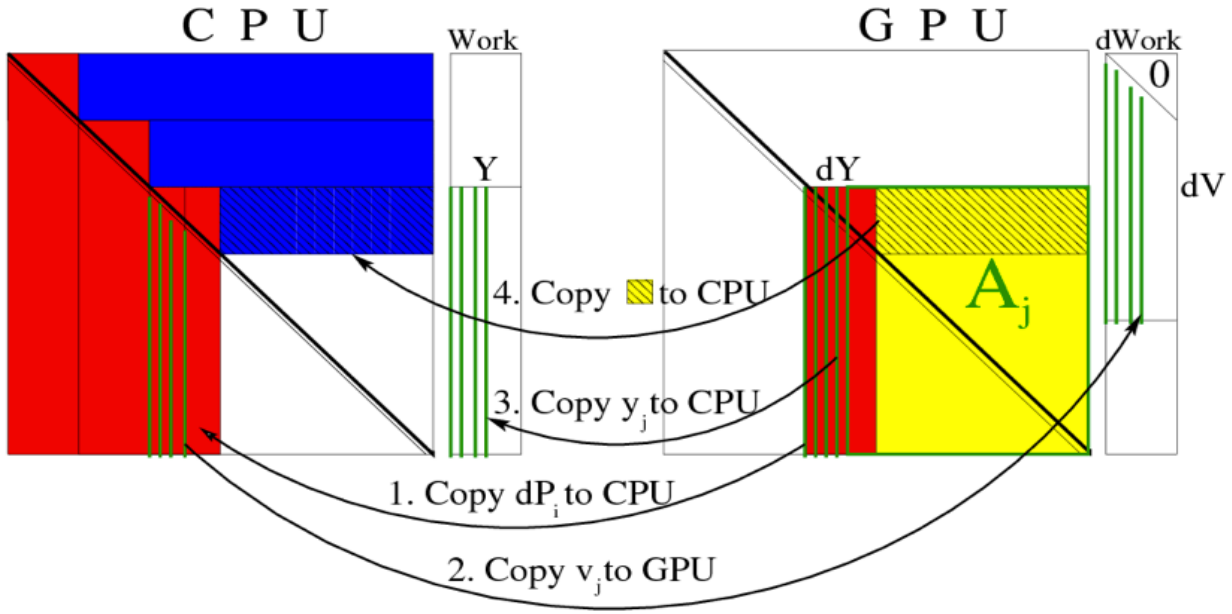
High-performance CUDA kernels were developed for various matrix-vector products
[ e.g., **ssymv** reaching up to **102 Gflop/s** for the symmetric eigenvalue problem ]

# Statically Scheduled **Two-Sided Factorizations**
## [ Hessenber, tridiagonal, and bidiagonal reductions ]

- **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
- **Note**
  - CPU-to-GPU communications and subsequent computations always stay in surface-to-volume ratio

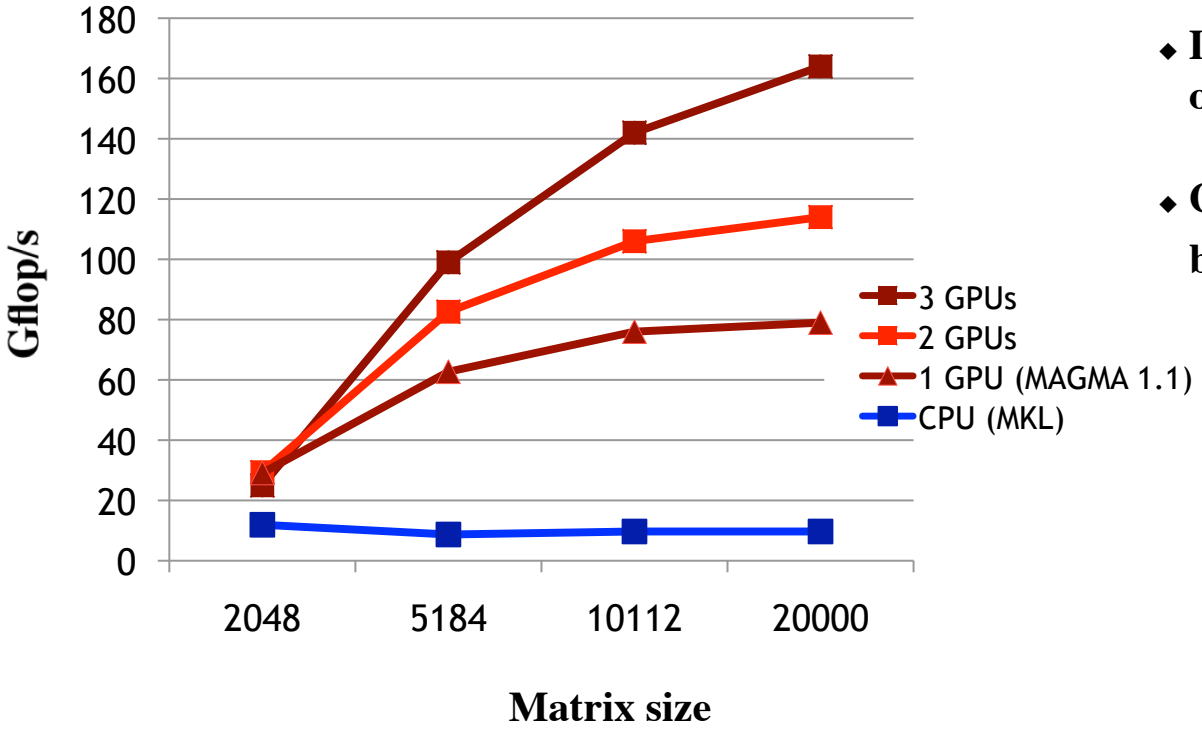
# Task **Splitting** & Task **Scheduling**



# Two-sided factorizations in MAGMA

**MAGMA Hessenberg reduction in double precision on Fermi**




- **Developed routines for multiGPUs obtaining scalable performance**
- **GPUs on a Keeneneland node bring a speedup of 16 x**

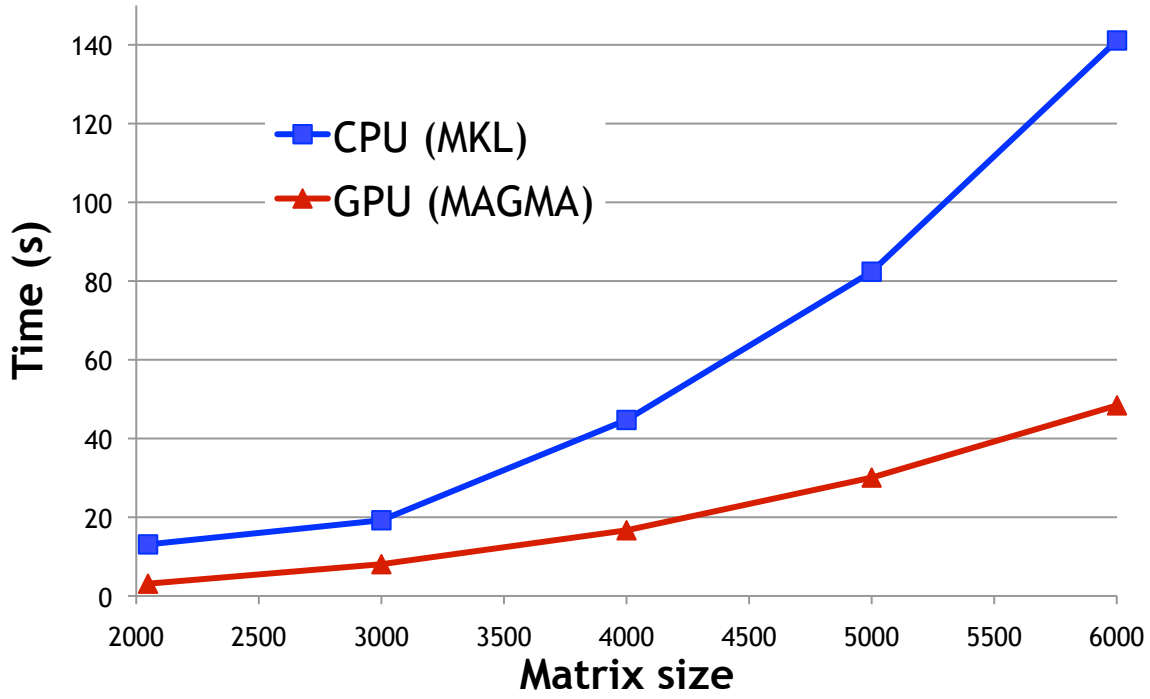
w/ Mark Gates, UTK

**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)

# Complete Eigensolvers

Generalized Hermitian-definite eigenproblem solver ( **A x = λ B x** )

[double complex arithmetic; based on Divide & Conquer; eigenvalues + eigenvectors]




w/ **Thomas Schulthess** & **Raffaele Solca**
ETH Zurich, Switzerland

| | | | |
|---|---|---|---|
| **GPU** | Fermi C2050 [448 CUDA Cores @ 1.15 GHz ] + Intel Q9300 [ 4 cores @ 2.50 GHz] | **CPU** | AMD ISTANBUL [ 8 sockets x 6 cores (48 cores) @2.8GHz ] |
| DP peak | **515 + 40** GFlop/s | DP peak | **538** GFlop/s |
| System cost | ~ $**3,000** | System cost | ~ $**30,000** |
| Power * | ~ **220** W | Power * | ~ **1,022** W |

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

# Complete Eigensolvers

## Hermitian general eigenvalue solver

- Solve $A x = \lambda B x$
- Compute Cholesky factorization of B.
  $B = LL^H$
  - xPOTRF
- Transform the problem to a standard eigenvalue problem
  $A = L^{-1}AL^{-H}$
  - xHEGST
- Solve Hermitian standard Eigenvalue problem
  $A' y = \lambda y$
  - xHEEVx
- Transform back the eigenvectors
  $x = L^{-H} y$
  - xTRSM

# Complete Eigensolvers

## Hermitian standard eigenvalue solver

- Solve $A\,y = \lambda\,y$
- Tridiagonalize A

  $$T = Q^H A' Q$$

  - xHETRD
- Compute eigenvalues and eigenvectors of the tridiagonal matryx

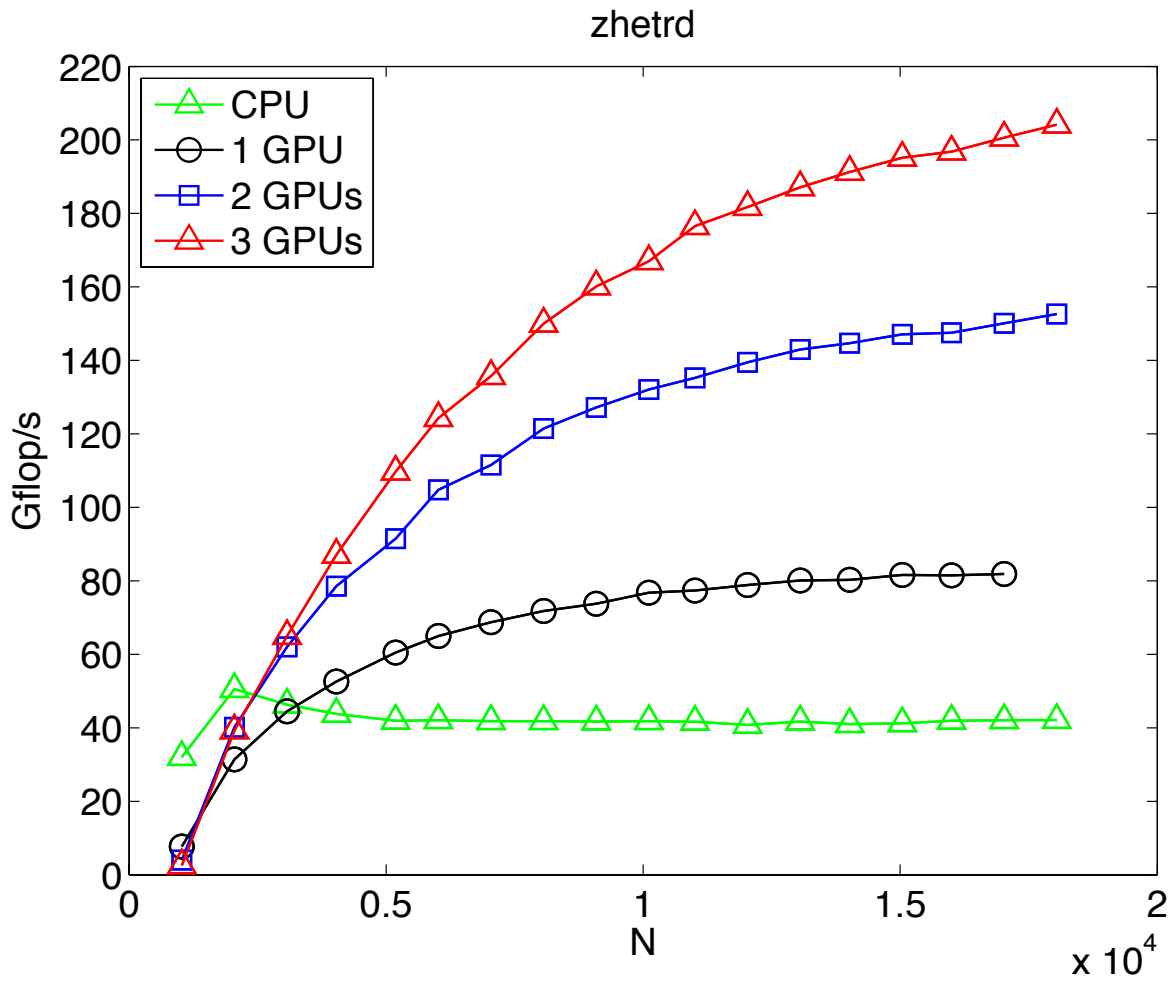
  $$T\,y = \lambda\,y'$$

  - xSTExx
- Transform back the eigenvectors

  $$y = Q\,y'$$

  - xUNMTR

# Tridiagonalization on multiGPUs



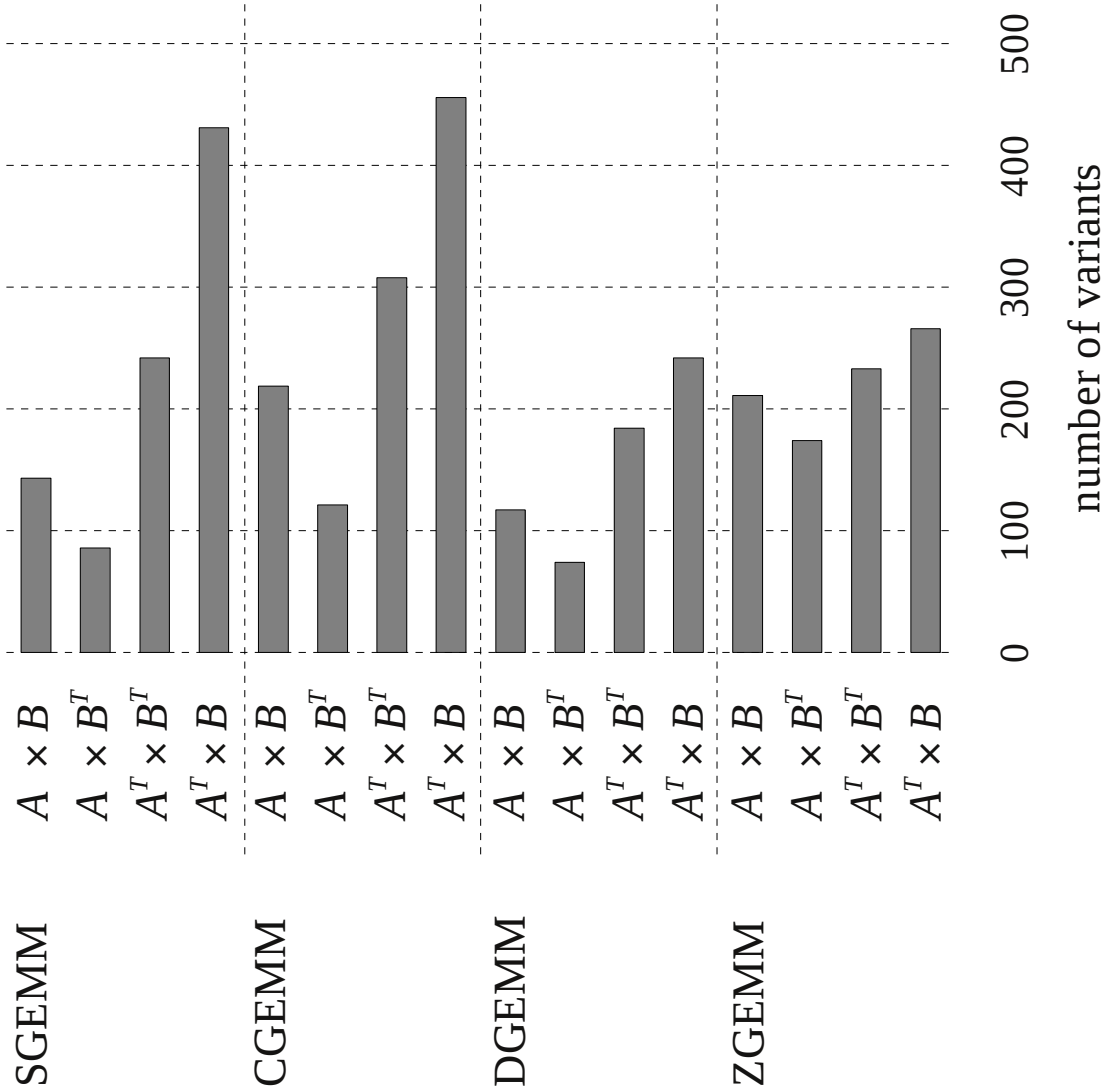

w/ Ichitaro Yamazaki, UTK
Tingxing Dong, UTK

**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)

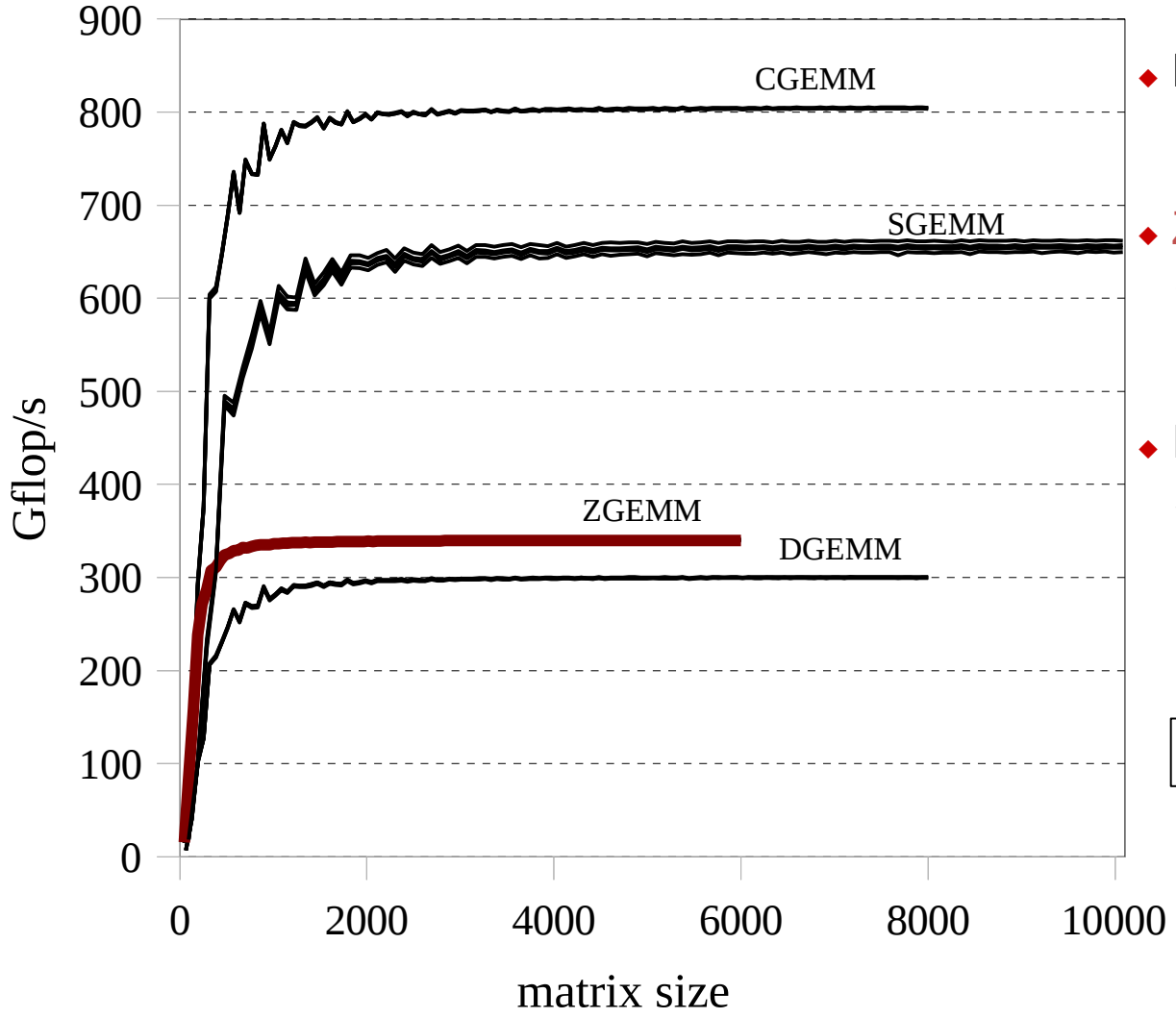
# Autotuning in MAGMA 1.1




- Number of GEMM variants generated and tested
  - automatically from “stencils” (parameterized code)

w/ **Jakub Kurzak**, UTK

# Autotuning in MAGMA 1.1




- Performance on Fermi (C2050) in Gflop/s
- ZGEMM improved significantly compared to CUBLAS
  - from 308 to 341 Gflop/s
- Improvement up to 2x on some specific matrices (e.g., of “rectangular” shape)

w/ **Jakub Kurzak**, UTK

# Productivity: sequential to hybrid code

- **Productivity -** develop parallel multicore + multiGPU algorithms from sequential algorithms using DAG-based runtime systems

```
// Sequential Tile Cholesky
FOR k = 0..TILES-1
    DPOTRF(A[k][k])
    FOR m = k+1..TILES-1
        DTRSM(A[k][k], A[m][k])
        FOR n = k+1..TILES-1
            DSYRK(A[n][k], A[n][n])
            FOR m = n+1..TILES-1
                DGEMM(A[m][k], A[n][k], A[m][n])
```

```
// Hybrid Tile Cholesky
FOR k = 0..TILES-1
   Insert_Task(DPOTRF, …)
   FOR m = k+1..TILES-1
      Insert_Task(DTRSM, …)
      FOR n = k+1..TILES-1
         Insert_Task(DSYRK, …)
         FOR m = n+1..TILES-1
            Insert_Task(DGEMM, …)
```

- Tile kernels and one-sided factorizations and solvers (using StarPU) are released in MAGMA 1.1

# Sparse iterative solvers

**Algorithm 1** GMRES for GPUs

1: **for** $i = 0, 1, \ldots$ **do**
2: $r = b - Ax_i$ (magma_sspmv)
3: $\beta = h_{1,0} = ||r||_2$ (cublasSnrm2)
4: check convergence and exit if done
5: **for** $k = 1, \ldots, m$ **do**
6: $v_k = r \ / \ h_{k,k-1}$ (magma_sscal)
7: $r = A \ v_k$ (magma_sspmv)
8: **for** j=1,...,k **do**
9: $h_{j,k} = r^T v_j$ (cublasSdot)
10: $r = r - h_{j,k} \ v_j$ (cublasSaxpy)
11: **end for**
12: $h_{k+1,k} = ||r||_2$ (cublasSnrm2)
13: **end for**
14: Define $V_k = [v_1, \ldots, v_k]$, $H_k = \{h_{i,j}\}$
15: Find $y_k$ that minimizes $||\beta e_1 - H_k \ y_k||_2$
16: $x_{i+1} = x_i + V_k \ y_k$ (magma_sgemv)
17: **end for**

**Algorithm 2** LOBPCG for GPUs

1: **for** $i = 0, 1, \ldots$ **do**
2: $R = P(AX_i - \Lambda X_i)$ (magma_sspmv)
3: check convergence and exit if done
4: $[X_i, \Lambda]$ = Rayleigh-Ritz on span$\{X_i, \ X_{i-1}, \ R\}$ (hybrid)
5: **end for**

- The hybridization approach naturally works
  [e.g., Richardson iteration in mixed-precision iterative refinement solvers, Krylov space iterative solvers and eigen-solvers ]
- Fast sparse matrix-vector product on Fermi
- Explore ideas to reduce communication
  [ e.g., mixed precision, reduced storage for integers for the indexing, etc. ]
- Need high bandwidth

# 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