

MAGMA – a New Generation of Linear Algebra Libraries for GPU and Multicore Architectures

Stan Tomov

Innovative Computing Laboratory
University of Tennessee, Knoxville

OLCF Seminar Series, ORNL
June 16, 2010

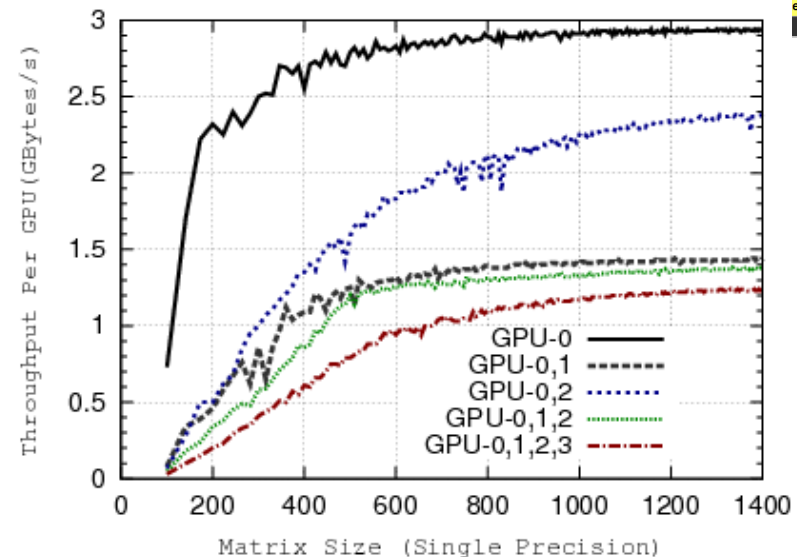


Outline

- **Challenges related to GPU and Multicore**
- **The *Hybridization Methodology* of MAGMA**
- **One-sided factorizations and solvers**
- **Two-sided factorizations and solvers**
- **MAGMA and the Fermi GPU**
- **Iterative linear and eigen-solvers**
- **Future work**

Challenges





- Increase in parallelism
Tesla C2050 (Fermi): 448 CUDA cores @1.15 GHz
SP peak is 1075 GFlop/s, DP peak is 515 Gflop/s
- Increase in communication cost [vs computation]
Processor speed improves ~59% / year
memory bandwidth by only 23%
- Heterogeneity



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, starting with current multicore+MultiGPU systems
Homepage: <http://icl.cs.utk.edu/magma/>
- **MAGMA & LAPACK**
 - **MAGMA** - based on LAPACK and extended for hybrid systems (multi-GPUs + multicore systems);
 - **MAGMA** - designed to be similar to LAPACK in functionality, data storage and interface, in order to allow scientists to effortlessly port any of their LAPACK-relying software components to take advantage of the new architectures
 - **MAGMA** - to leverage years of experience in developing open source LA software packages and systems like LAPACK, ScaLAPACK, BLAS, ATLAS as well as the newest LA developments (e.g. communication avoiding algorithms) and experiences on homogeneous multicores (e.g. PLASMA)
- **Support**
 - NSF, Microsoft, NVIDIA [now **CUDA Center of Excellence at UTK** on the development of **Linear Algebra Libraries for CUDA-based Hybrid Architectures**]
- **MAGMA developers**
 - University of Tennessee, **Knoxville**; University of California, **Berkeley**; University of Colorado, **Denver**

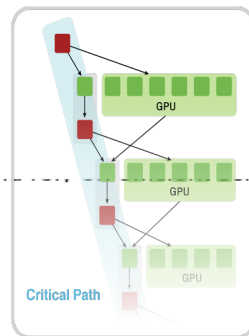
A New Generation of Algorithms

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (00's) New Algorithms (many-core friendly)		Rely on - a DAG/scheduler - block data layout - some extra kernels

“delayed update” to organize successive Level 2 BLAS as a single Level 3 BLAS

Localized (over tiles) elementary transformations

MAGMA Hybrid Algorithms (heterogeneity friendly)



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

Hybridization methodology

- MAGMA uses **HYBRIDIZATION** methodology based on

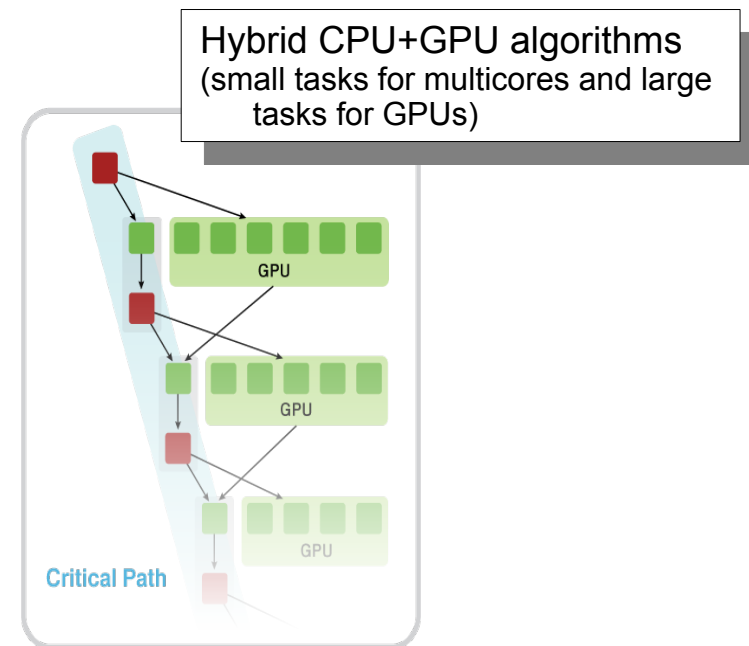
- Representing linear algebra algorithms as collections of **TASKS** and **DATA DEPENDANCIES** among them
- Properly **SCHEDULING** the tasks' execution over the multicore and the GPU hardware components

- Successfully applied to fundamental linear algebra algorithms

- One and two-sided factorizations and solvers
- Iterative linear and eigen-solvers

- Faster, cheaper, better ?

- High-level
- Leveraging prior developments
- Exceeding in performance (and sometimes accuracy) homogeneous solutions



MAGMA Status

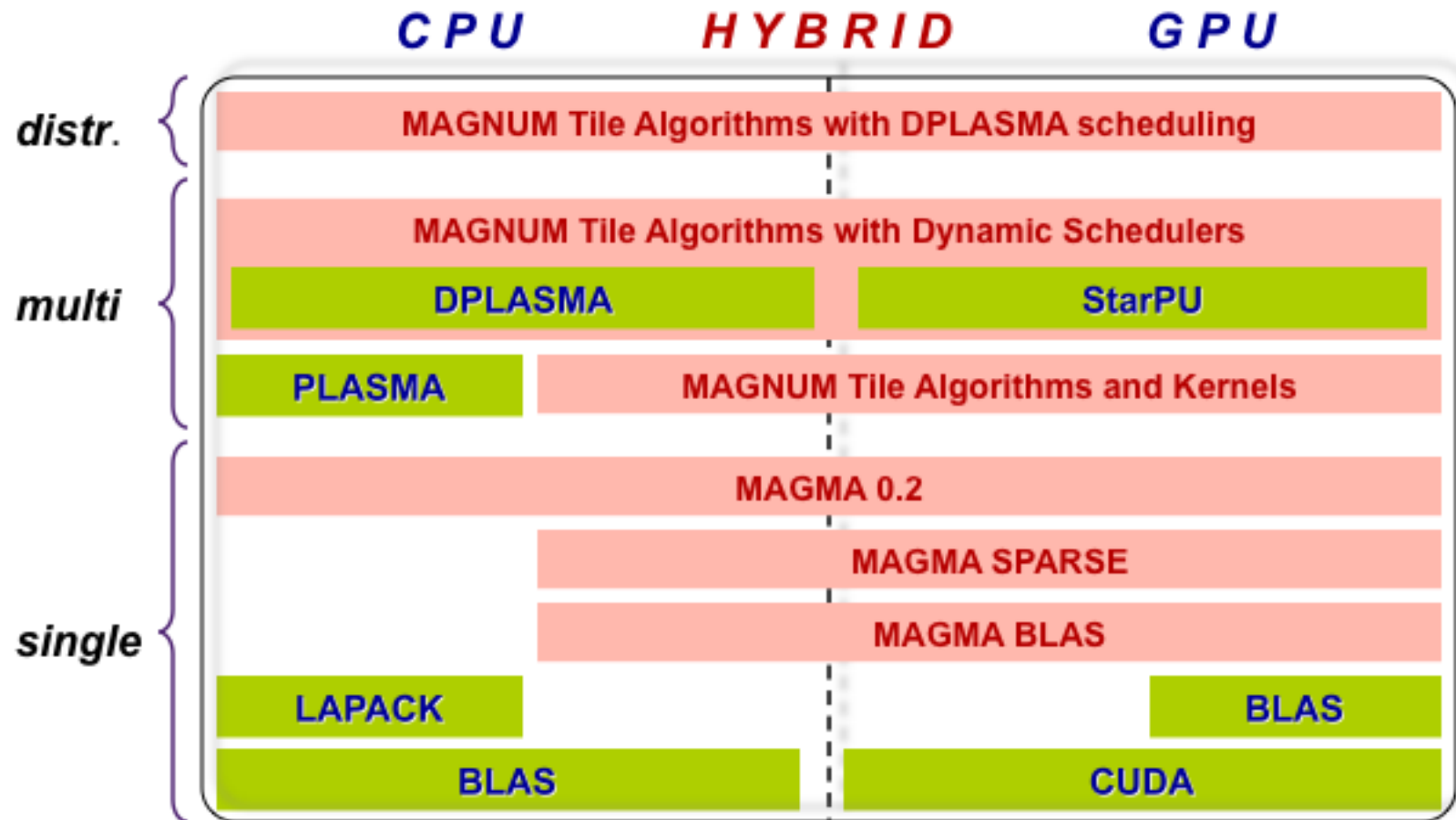
MAGMA 0.2

- **LU, QR, Cholesky (S, C, D, Z)**
- **Linear solvers**
 - ◆ In working precision, based on LU, QR, and Cholesky
 - ◆ Mixed-precision iterative refinement
- **CPU and GPU interfaces**
- **Two-sided factorizations**
 - ◆ Reduction to upper Hessenberg form for the general eigenvalue problem
- **MAGMA BLAS**
 - ◆ Routines critical for MAGMA (GEMM, SYRK, TRSM, GEMV, SYMV, etc.)

Unreleased

- **Bidiagonal two-sided reduction for SVD**
- **Tridiagonal two-sided for the symmetric eigenvalue problem**
- **Divide & Conquer for the symmetric eigenvalue problem**
- **GEMM for FERMI**
- **Cholesky and QR for multiGPUs on MAGNUM tiles**
 - ◆ Hybrid kernels (building blocks) for tile algorithms (e.g., dynamically scheduled)
- **GMRES and PCG**

MAGMA Software Stack



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

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”

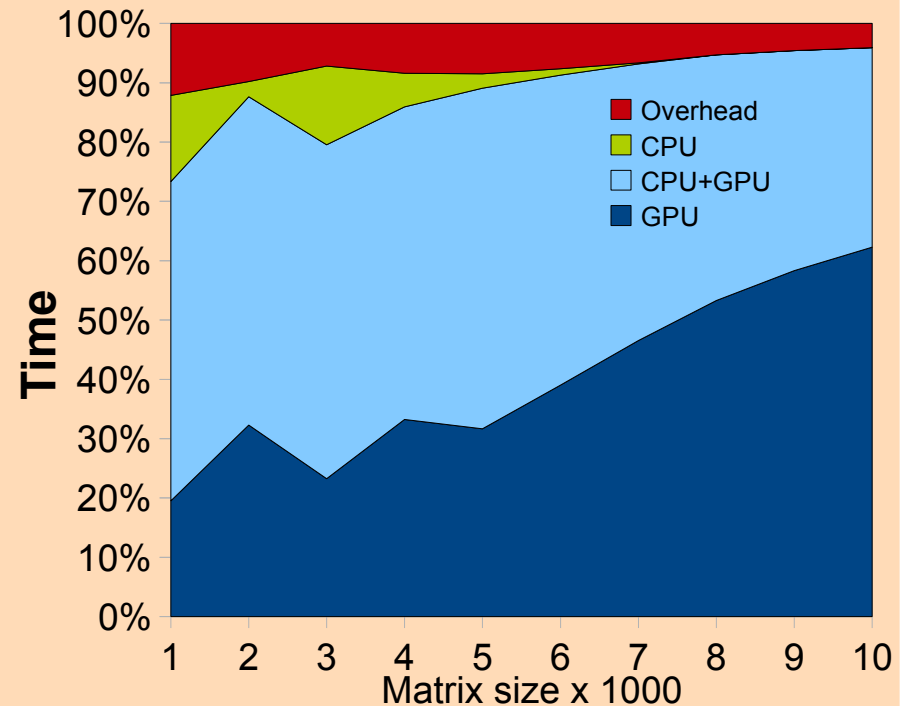
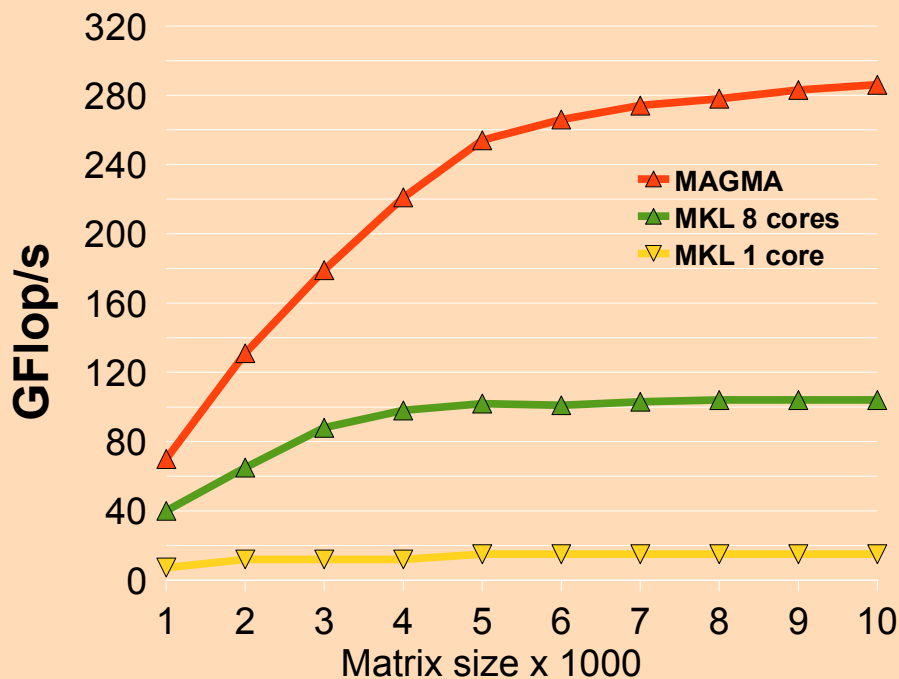
- Note

- ◆ 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

Performance of the one-sided statically scheduled hybrid factorizations

QR factorization in single precision arithmetic, CPU interface
Performance of MAGMA vs MKL

MAGMA QR time breakdown



GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

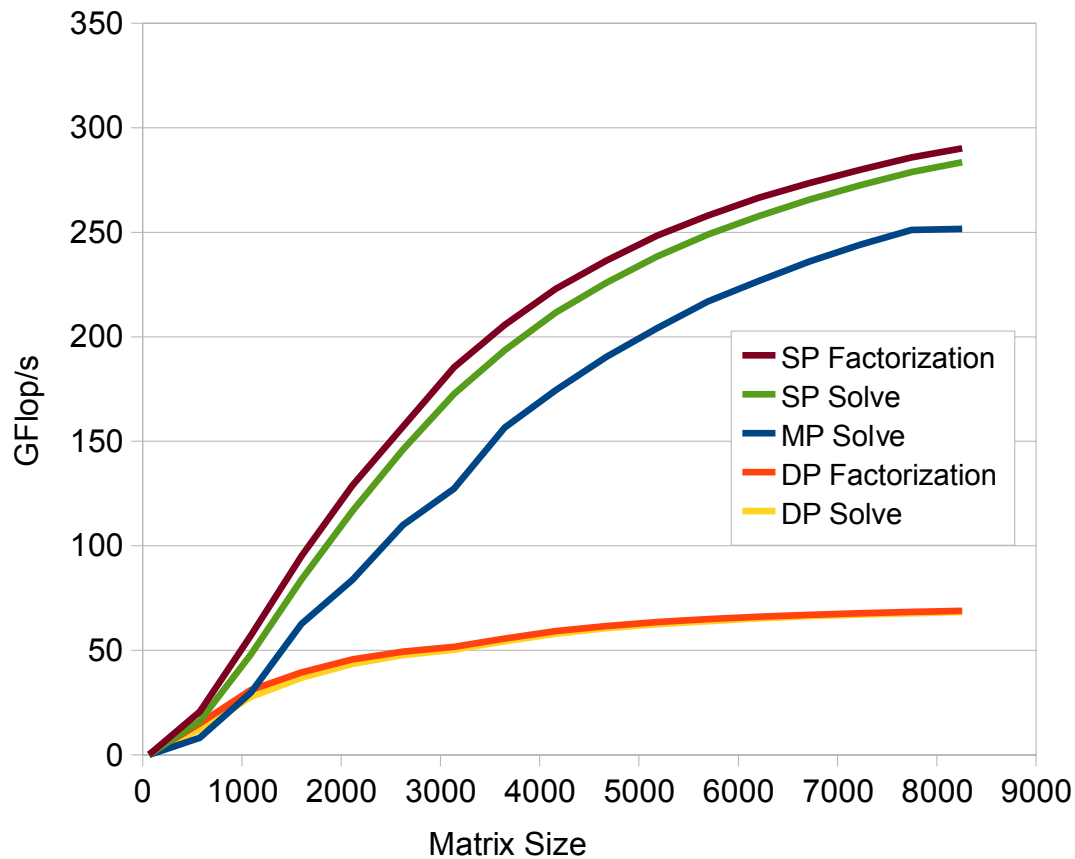
GPU BLAS : CUBLAS 2.2, sgemm peak: 375 GFlop/s
CPU BLAS : MKL 10.0 , sgemm peak: 128 GFlop/s

[for more performance data, see <http://icl.cs.utk.edu/magma>]

Linear Solvers

Solving $Ax = b$ using LU factorization

Intel(R) Xeon(R)E541@2.34GHz / 8 Cores + GTX 280 @1.30GHz / 240 Cores



Direct solvers

- Factor and do triangular solves in the same, 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.

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

MultiGPU and Multicore

• MAGNUM tiles

- ◆ Tasks are **hybrid**, **GPU BLAS-based**, or **multicore** kernels

Scheduling:

- ◆ Using **PLASMA** with customized extensions to reduce communication and w/ hybrid MAGMA kernels
 - Demonstrated scalability for one-sided factorizations
 - Highly optimized, used as benchmark to compare with dynamic schedulers [e.g., performance on 4 C1060 GPUs for Cholesky is up to 1200 Gflop/s, for QR is up to 830 Gflop/s in SP]
- ◆ Using **StarPU** to schedule hybrid, GPU and multicore kernels (from PLASMA and MAGMA) <http://runtime.bordeaux.inria.fr/StarPU/> [in collaboration with INRIA's StarPU team]
- ◆ Using the **DPLASMA** scheduler

• Rectangular tiles

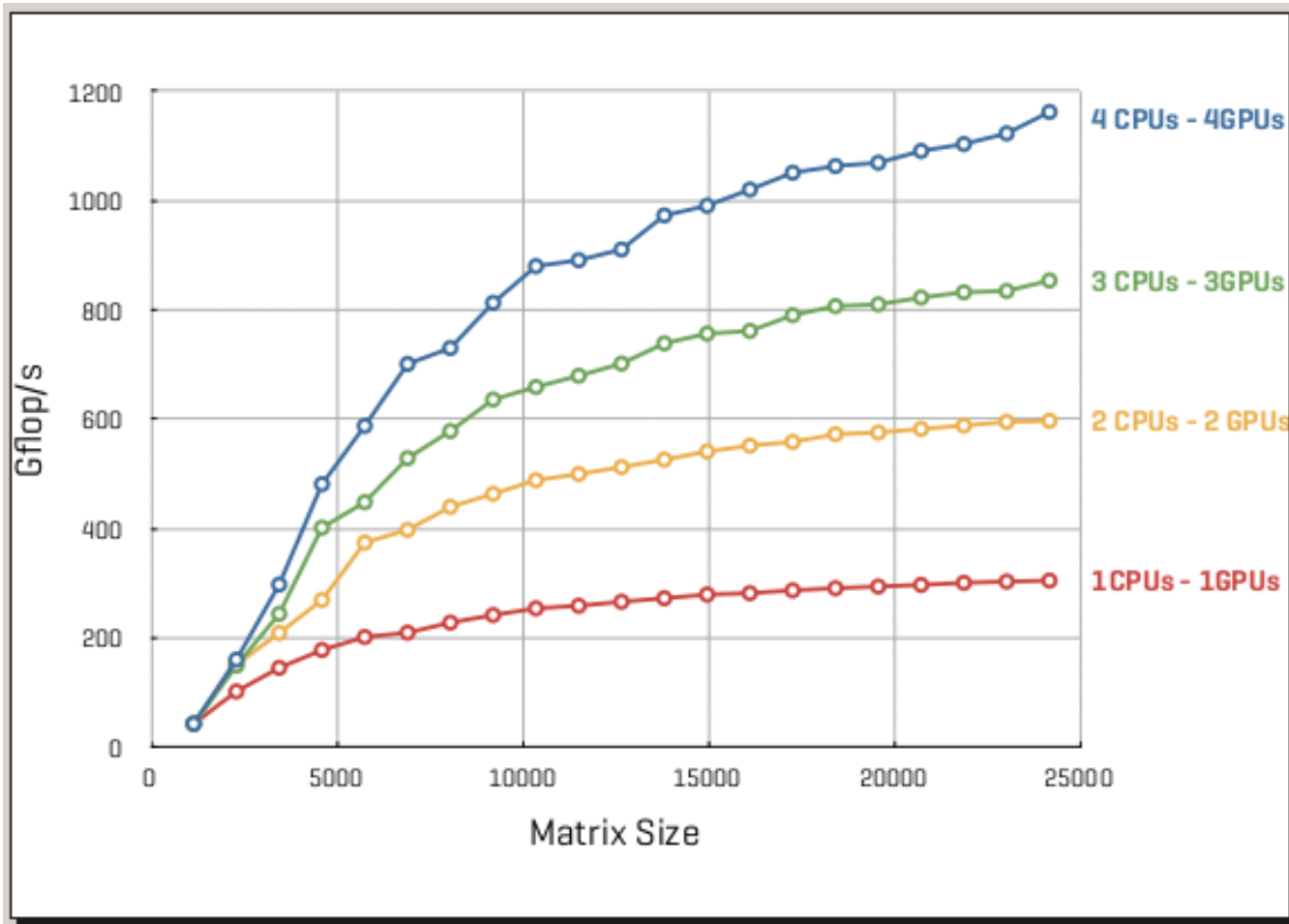
- ◆ Tiles of variable sizes to be used to account for the heterogeneity of the system
- ◆ To experiment with “communication-avoiding” algorithms

• PLASMA tile algorithms

- ◆ A single GPU kernel processing multiple tile tasks in parallel (Jakub) [vs only one, but magnum tile, at a time]

Scheduling with PLASMA

Cholesky factorization in SP



HOST: 4x AMD Opteron core @1.8GHz
GPUs: 4x C1060 (240 cores each @1.44GHz)

Scheduling with StarPU

Statistics for codelet spotrf

```

CUDA 0 (Quadro FX 5800) -> 3 / 36 (8.33 %)
CUDA 1 (Quadro FX 5800) -> 1 / 36 (2.78 %)
CUDA 2 (Quadro FX 5800) -> 3 / 36 (8.33 %)
CPU 0 -> 6 / 36 (16.67 %)
CPU 1 -> 9 / 36 (25.00 %)
CPU 2 -> 4 / 36 (11.11 %)
CPU 3 -> 6 / 36 (16.67 %)
CPU 4 -> 4 / 36 (11.11 %)

```

Statistics for codelet ssyrk

```

CUDA 0 (Quadro FX 5800) -> 41 / 630 (6.51 %)
CUDA 1 (Quadro FX 5800) -> 40 / 630 (6.35 %)
CUDA 2 (Quadro FX 5800) -> 49 / 630 (7.78 %)
CPU 0 -> 105 / 630 (16.67 %)
CPU 1 -> 85 / 630 (13.49 %)
CPU 2 -> 105 / 630 (16.67 %)
CPU 3 -> 102 / 630 (16.19 %)
CPU 4 -> 103 / 630 (16.35 %)

```

Statistics for codelet strsm

```

CUDA 0 (Quadro FX 5800) -> 125 / 630 (19.84 %)
CUDA 1 (Quadro FX 5800) -> 127 / 630 (20.16 %)
CUDA 2 (Quadro FX 5800) -> 122 / 630 (19.37 %)
CPU 0 -> 50 / 630 (7.94 %)
CPU 1 -> 52 / 630 (8.25 %)
CPU 2 -> 52 / 630 (8.25 %)
CPU 3 -> 54 / 630 (8.57 %)
CPU 4 -> 48 / 630 (7.62 %)

```

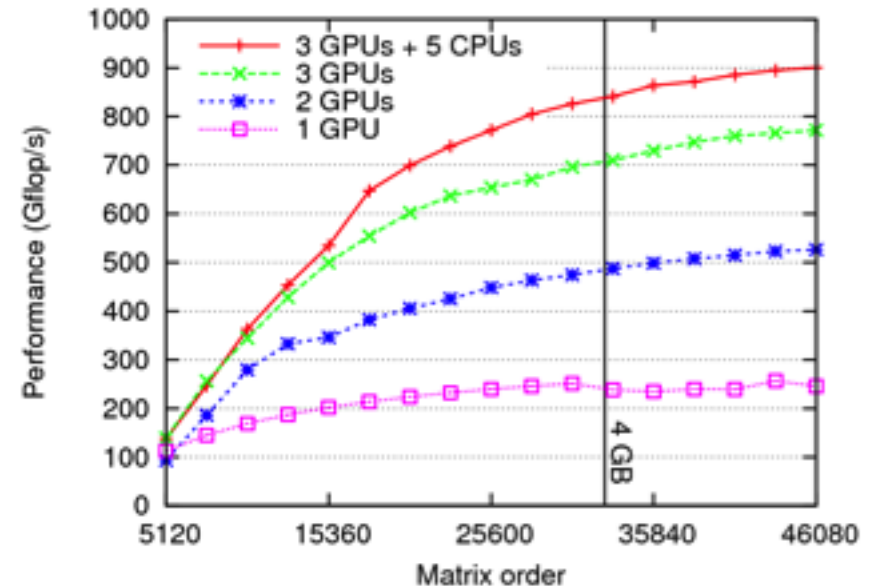
Statistics for codelet sgemm

```

CUDA 0 (Quadro FX 5800) -> 2258 / 7140 (31.62 %)
CUDA 1 (Quadro FX 5800) -> 2182 / 7140 (30.56 %)
CUDA 2 (Quadro FX 5800) -> 2261 / 7140 (31.67 %)
CPU 0 -> 87 / 7140 (1.22 %)
CPU 1 -> 94 / 7140 (1.32 %)
CPU 2 -> 85 / 7140 (1.19 %)
CPU 3 -> 85 / 7140 (1.19 %)
CPU 4 -> 88 / 7140 (1.23 %)

```

Performance of Cholesky factorization in SP



SGEMM

```

gpu : 333.04 GFlop/s
cpu : 20.06 GFlop/s

```

STRSM

```

gpu : 59.46 GFlop/s
cpu : 18.96 GFlop/s

```

SSYRK

```

gpu : 298.74 GFlop/s
cpu : 19.50 GFlop/s

```

SPOTRF

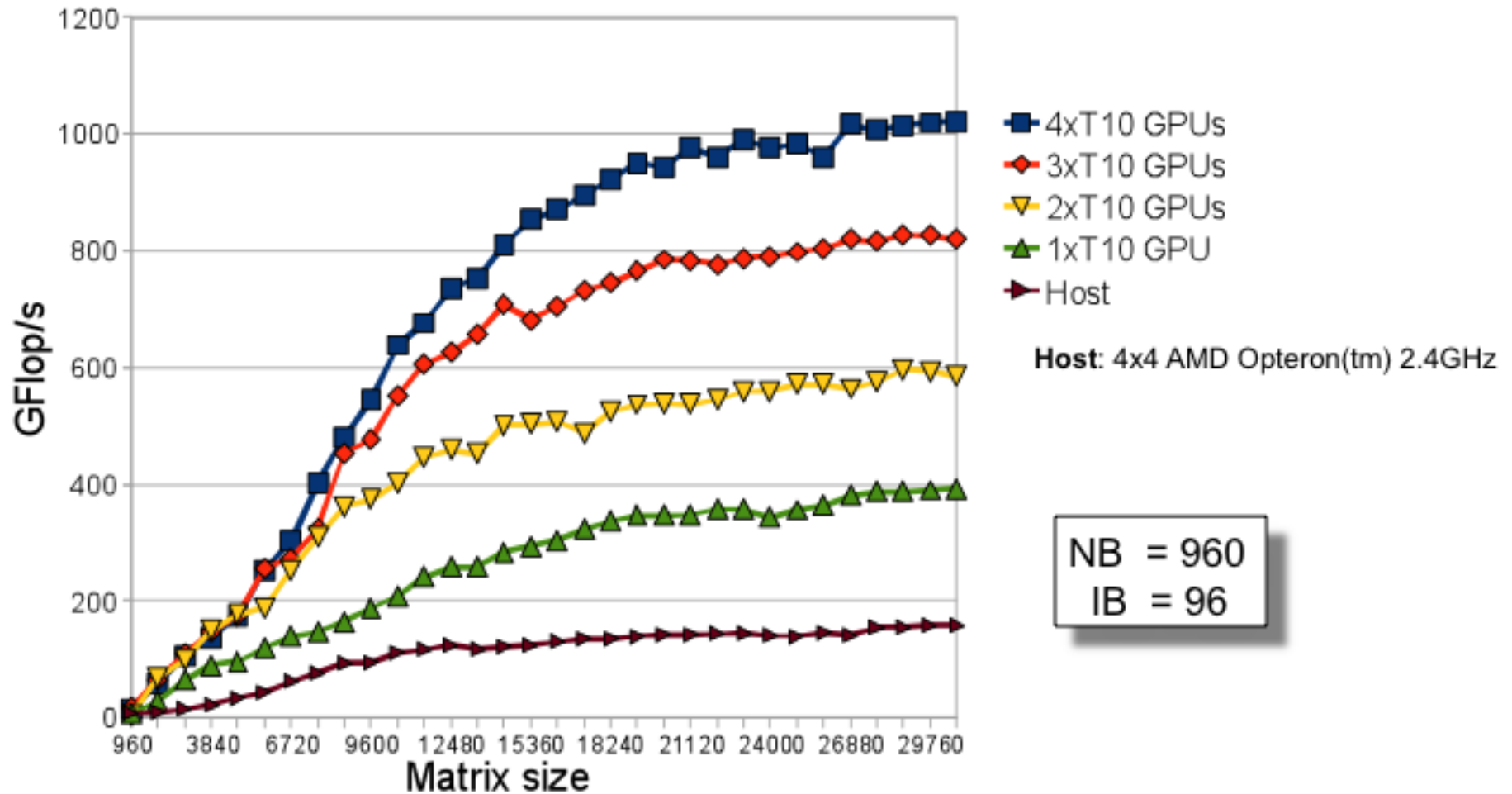
```

gpu : 57.51 GFlop/s
cpu : 17.45 GFlop/s

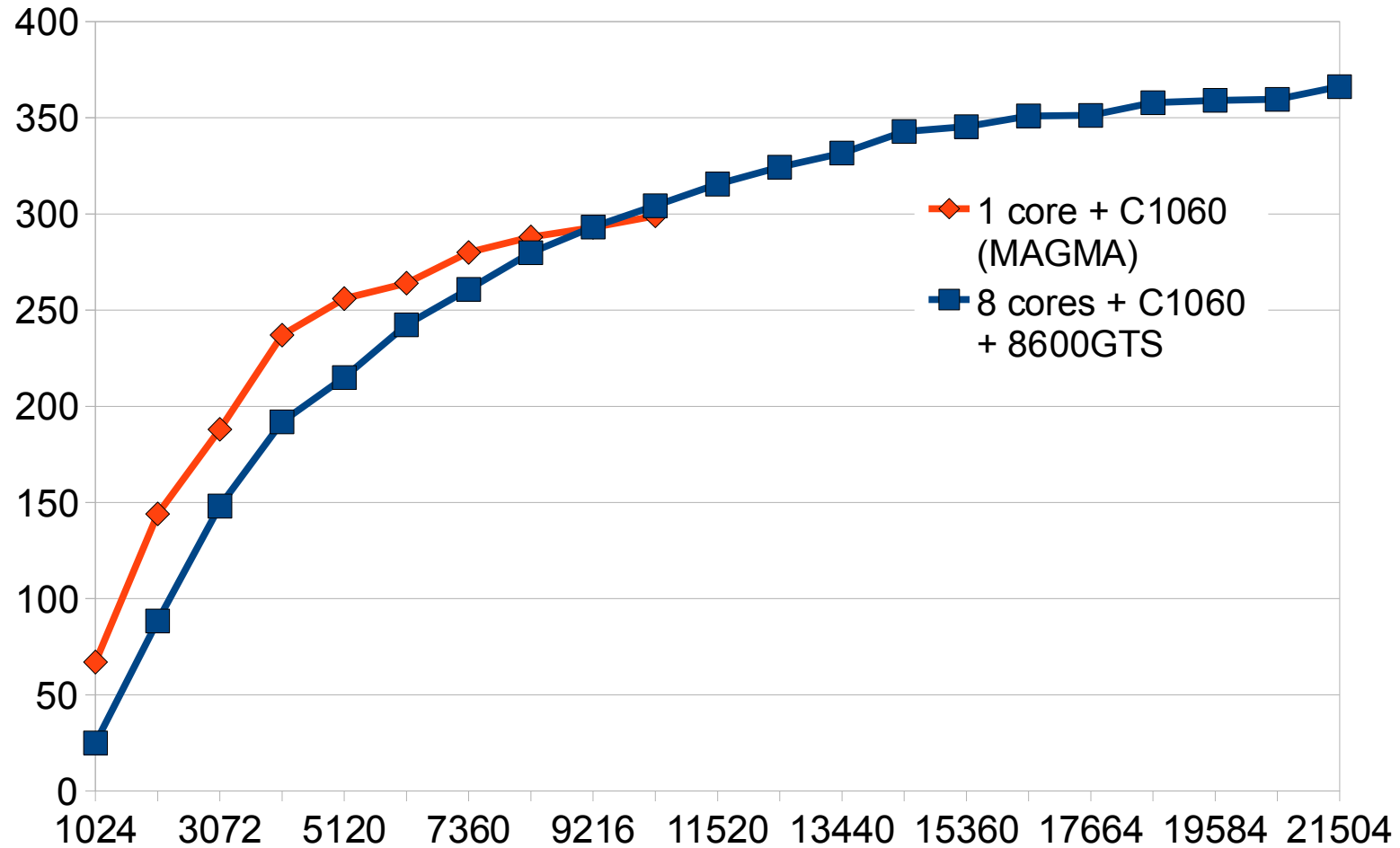
```

Scheduling with StarPU

Performance of Cholesky factorization in SP

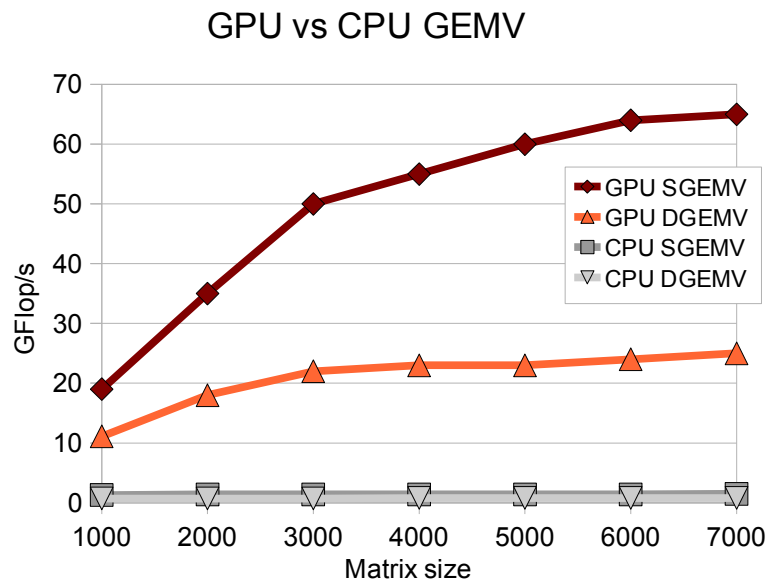


Scheduling with DPLASMA



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



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]

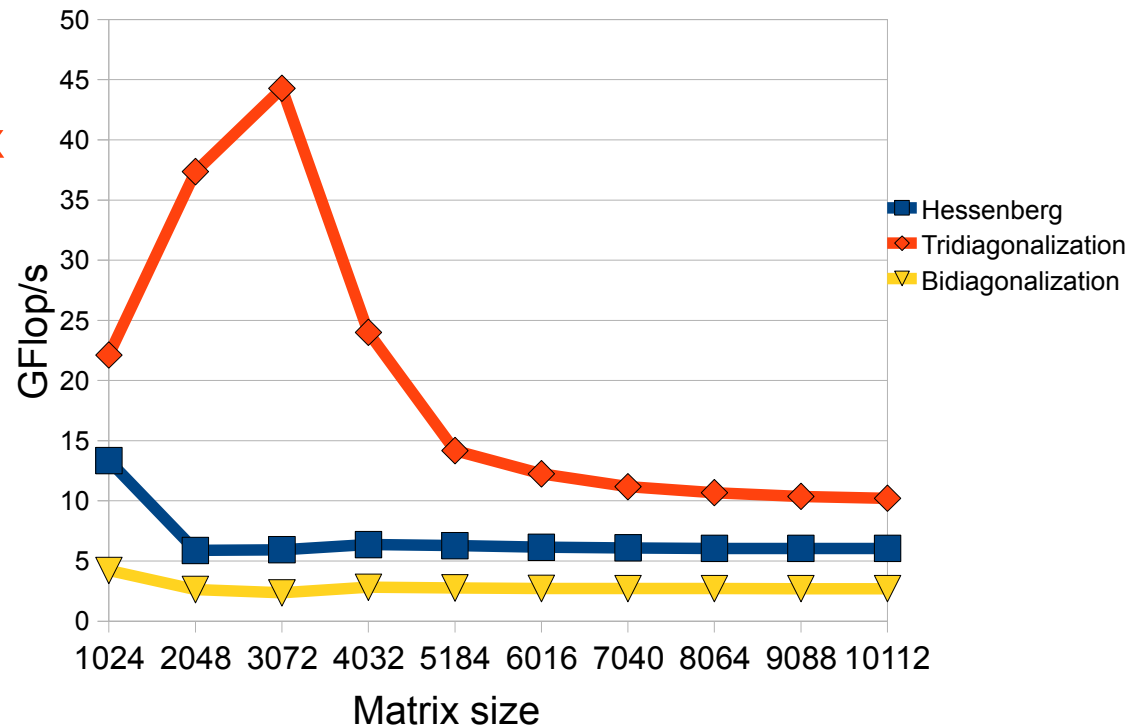
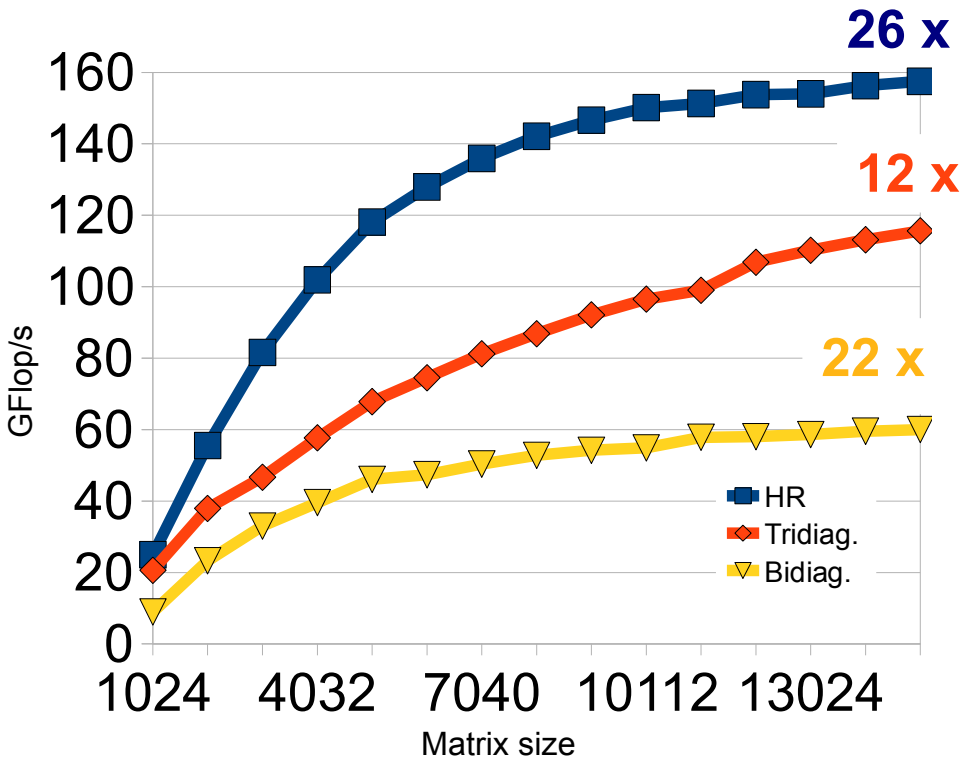
GPU: GTX280 (240 cores @ 1.30GHz, 141 GB/s)
CPU: 2 x 4 cores Intel Xeon @ 2.33GHz, 10.4 GB/s

Two-sided factorizations

(performance in single precision arithmetic)

GPU Performance

Multicore Performance



GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

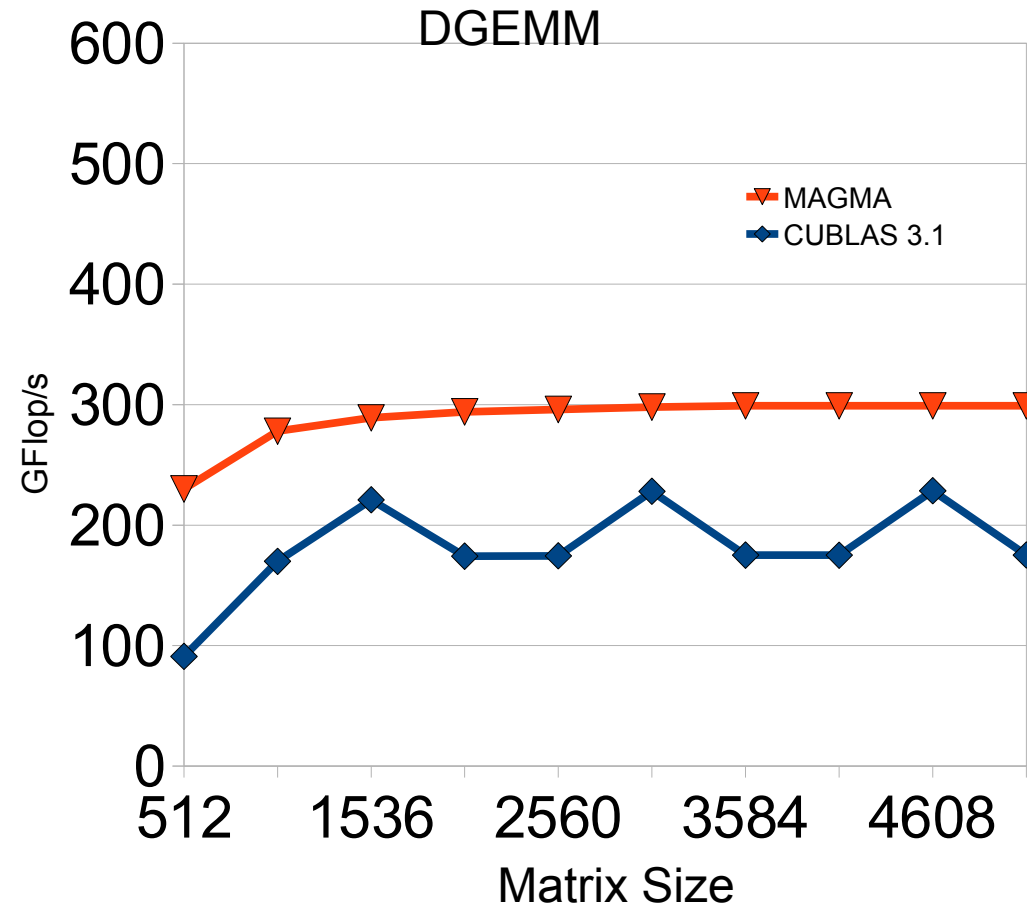
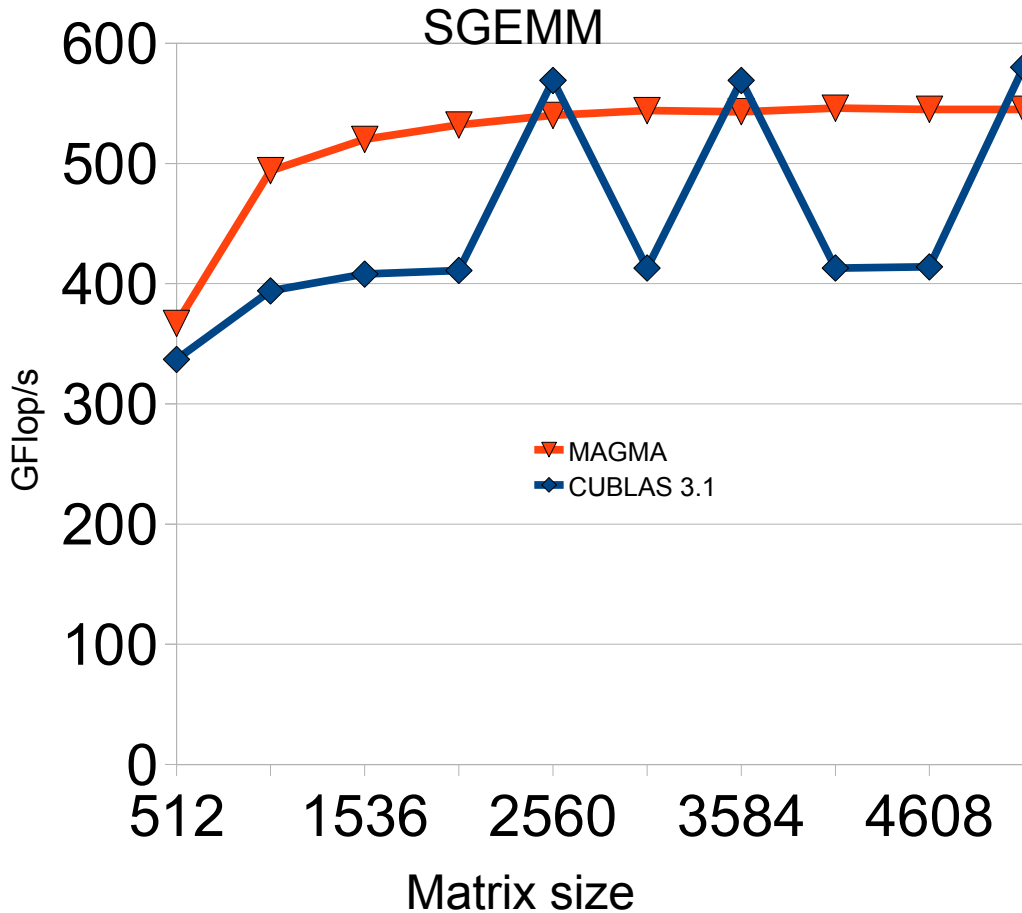
GPU BLAS : CUBLAS 2.3, dgemm peak: 75 GFlop/s
CPU BLAS : MKL 10.0 , dgemm peak: 65 GFlop/s

FERMI

- What has changed regarding MAGMA algorithms?
 - MAGMA is coded on high-level, extracting its performance from the performance of low-level kernels, i.e.,
everything works for FERMI and nothing has changed on high-level
 - We have relied on being able to develop the low-level kernels needed of very high-performance
as GPUs become more complex, this has become more difficult
 - **Auto-tuning has become more important**

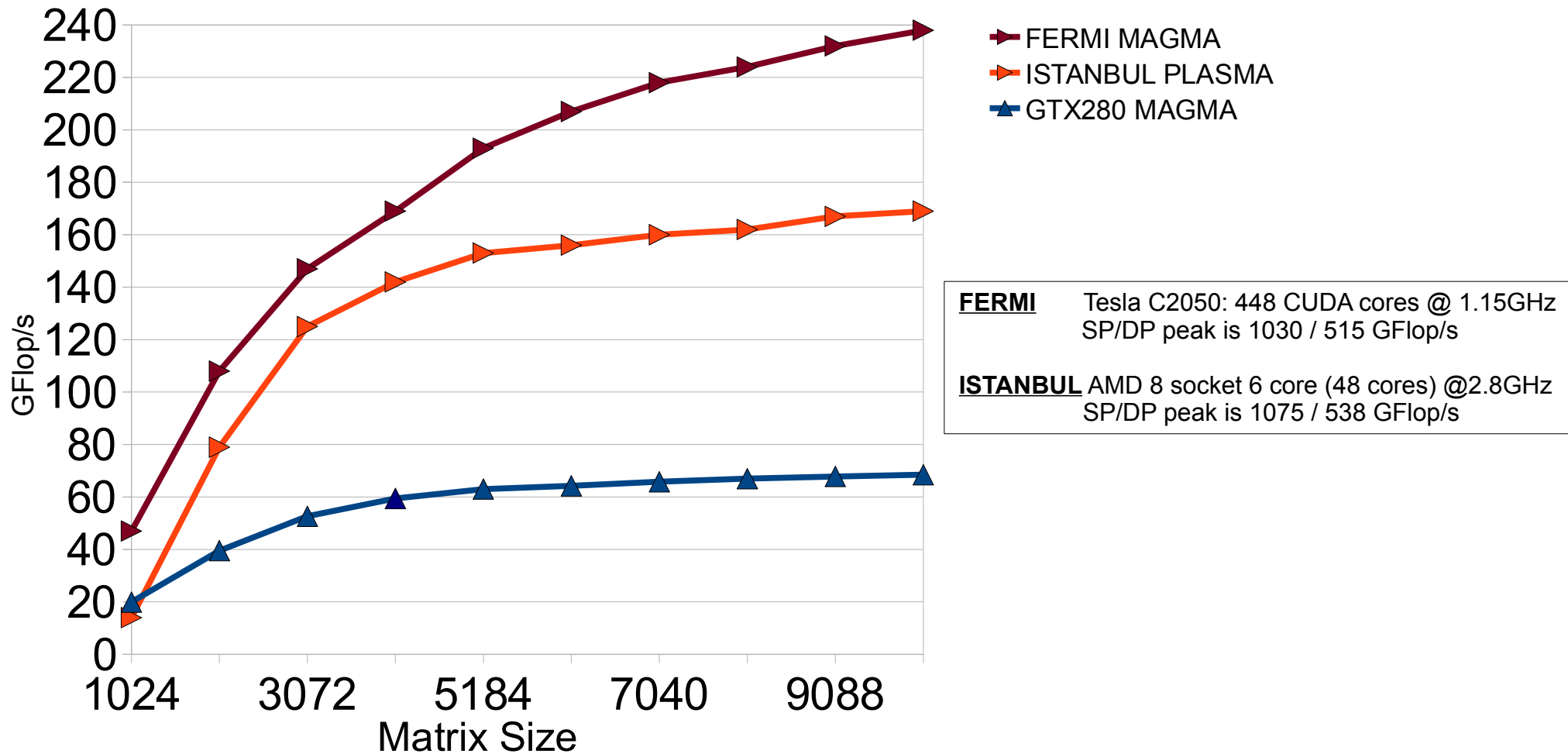
GEMM for FERMI

(Tesla C2050: 448 CUDA cores @ 1.15GHz, theoretical SP peak is 1.03 Tflop/s, DP peak 515 GFlop/s)

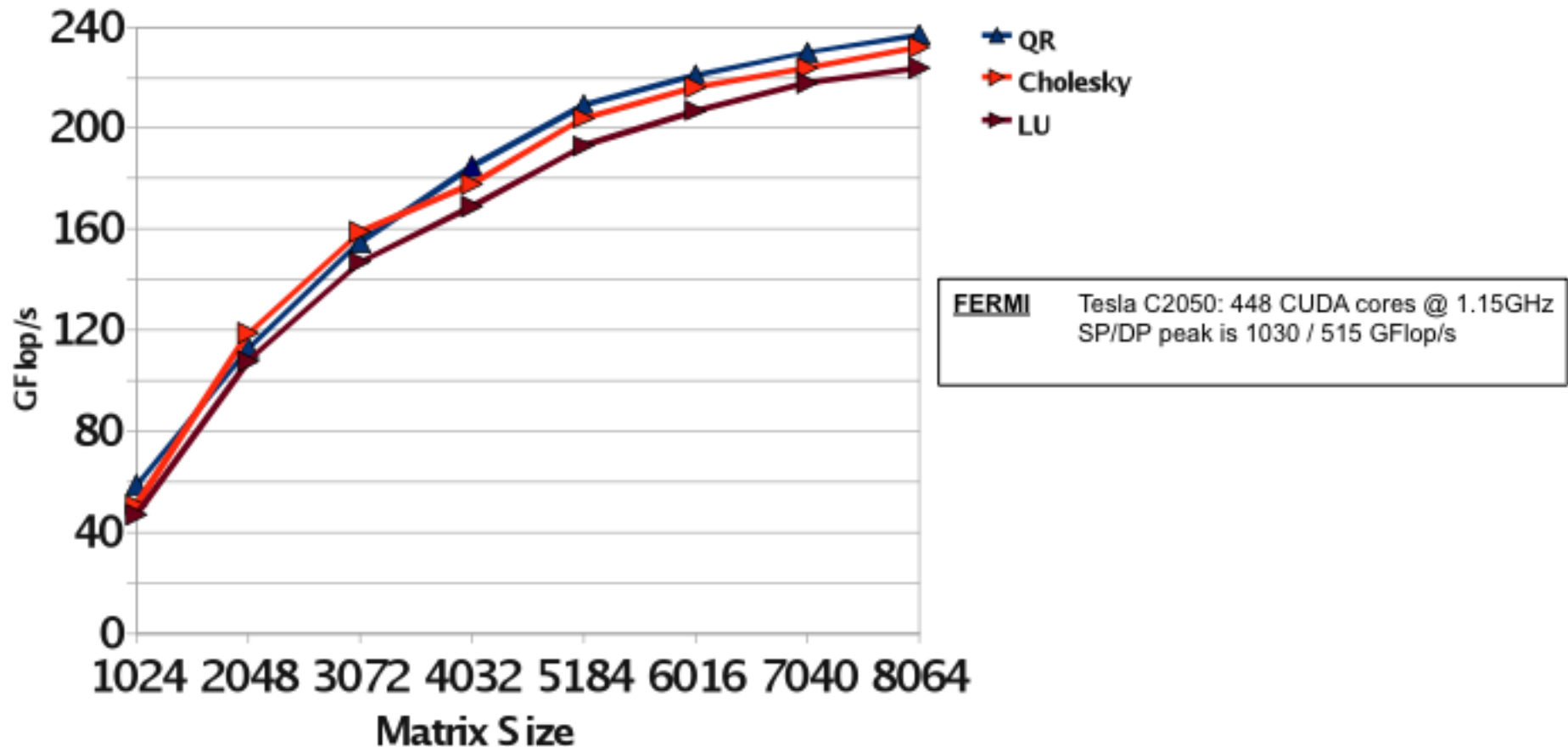


- Kernels have to be **redesigned** and **auto-tuned** for Fermi, e.g., inner-most blocking sizes have to be increased; add register blocking, etc.
- May even need to be written in **assembly**

LU factorization in double precision



LU, Cholesky, and QR on Fermi in DP



Sparse Linear Algebra

Algorithm 1 GMRES for GPUs

```

1: for  $i = 0, 1, \dots$  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  $k = 1, \dots, m$  do
6:      $v_k = r / h_{k,k-1}$  (magma_sscal)
7:      $r = A v_k$  (magma_sspmv)
8:     for  $j=1, \dots, k$  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  $V_k = [v_1, \dots, v_k]$ ,  $H_k = \{h_{i,j}\}$ 
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  $i = 0, 1, \dots$  do
2:    $R = P(AX_i - \Lambda X_i)$  (magma_sspmv)
3:   check convergence and exit if done
4:    $[X_i, \Lambda] = \text{Rayleigh-Ritz on}$ 
        $\text{span}\{X_i, X_{i-1}, R\}$  (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]
- Observed better accuracy (compared to CPU)
- Fast sparse matrix-vector product on Fermi does not have to use texture memory
- Explore ideas to reduce communication [e.g., mixed precision, reduced storage for integers for the indexing, etc.]
- Need high bandwidth

MAGMA Future Plans

- Experimentation with various schedulers (StarPU, PLASMA, DPLASMA) and improvements for multicore with multiGPUs
- “Communication avoiding” algorithms for systems of multicore CPUs and GPUs
- Kernels for FERMI (GEMM, SYRK, TRSM, SpMV)
- Auto-tuning framework
- Port [or facilitate the port] to Windows, Python and Matlab
- Krylov space iterative linear solvers and block eigensolvers

Collaborators / Support

- **MAGMA** Matrix Algebra on GPU and Multicore Architectures

[ICL team: J. Dongarra, S. Tomov, R. Nath, H. Ltaief]

<http://icl.cs.utk.edu/magma/>



- **PLASMA** Parallel Linear Algebra for Scalable Multicore Architectures
<http://icl.cs.utk.edu/plasma>



- Collaborating partners

University of Tennessee, Knoxville
University of California, Berkeley
University of Colorado, Denver

University of Coimbra, Portugal
INRIA, France (StarPU team)

