

Autotuning dense linear algebra libraries on GPUs and overview of the MAGMA library

Rajib Nath, Stan Tomov, Jack Dongarra

Innovative Computing Laboratory
University of Tennessee, Knoxville

Speaker: Emmanuel Agullo

PMAA'10, University of Basel, Switzerland
June 29 – July 02, 2010



Outline

- **Challenges related to GPU and Multicore**
- **The *Hybridization Methodology* of MAGMA**
- **Auto-tuning: GTX280 and Tesla C1060**
- **High-level one-sided and two-sided factorizations and solvers**
- **Auto-tuning: move to Fermi (Tesla C2050)**
- **Conclusion and future work**

Challenges

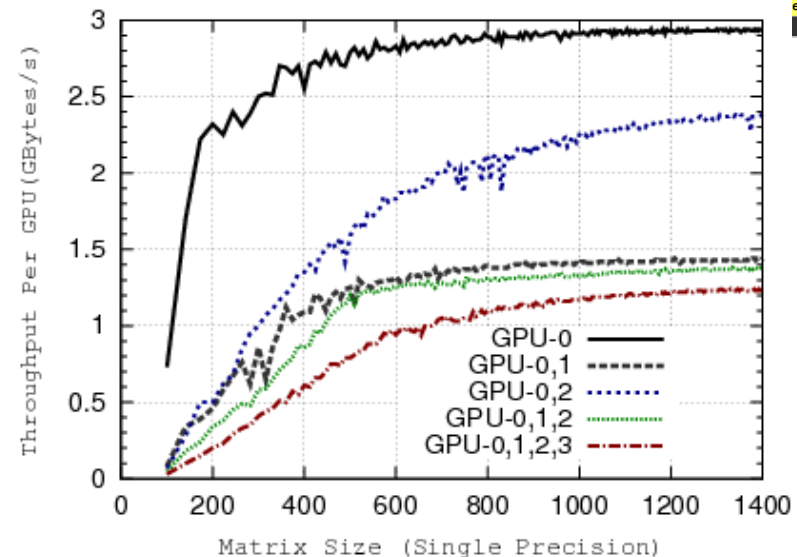
- Increase in parallelism

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**

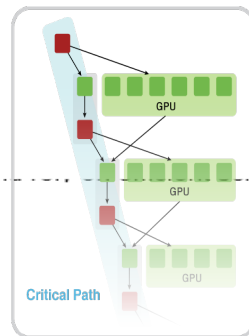
A New Generation of Algorithms

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

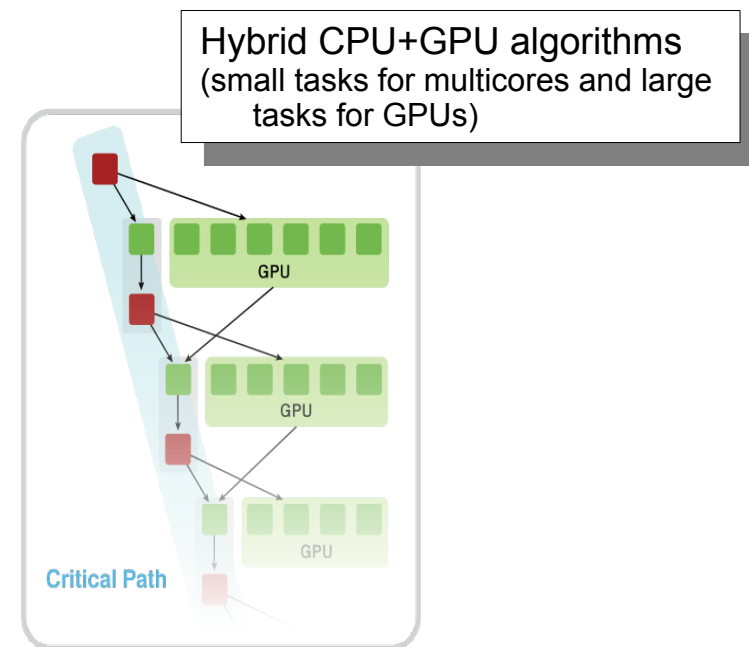
- ◆ Representing linear algebra algorithms as collections of **TASKS** and **DATA DEPENDANCIES** among them
- ◆ Properly **SCHEDULING** the tasks' execution over the multicore and the GPU hardware components

- 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

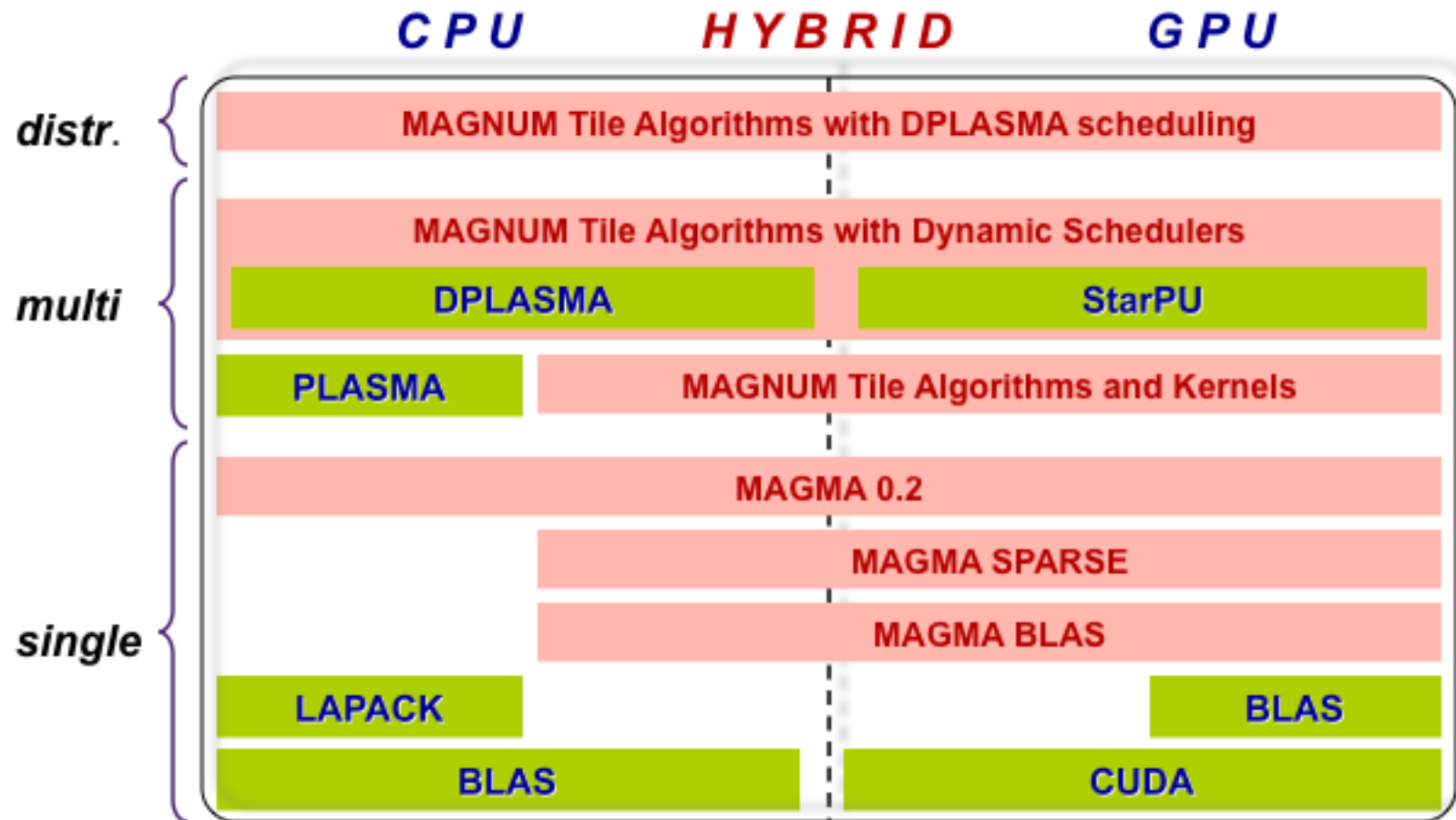
MAGMA 0.2

- **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*

Auto-tuning MAGMA BLAS (GTX280 and Tesla C1060)

Two components of the auto-tuner:

- Code Generator:

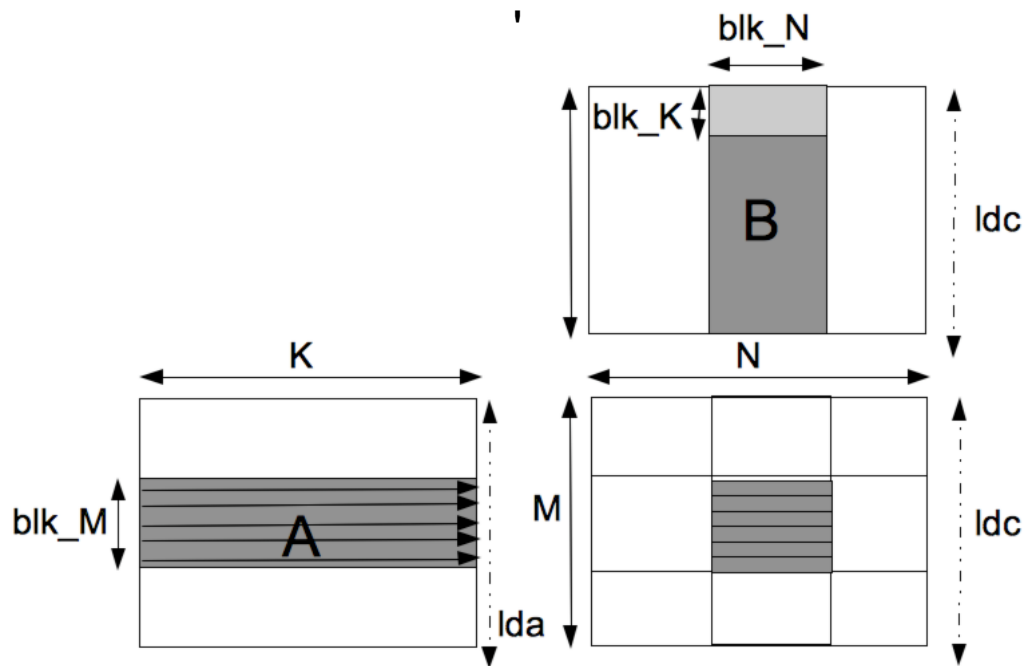
- ◆ produces code variants according to a set of pre-defined, parametrized templates and algorithms
- ◆ also applies certain state of the art optimization techniques

- Heuristic Search Engine:

- ◆ runs the variants produced by the code generator and finds out the best one using a feedback loop
- ◆ uses exhaustive search as the kernels are really important

xGEMM kernel (1/4) (GTX280 and Tesla C1060)

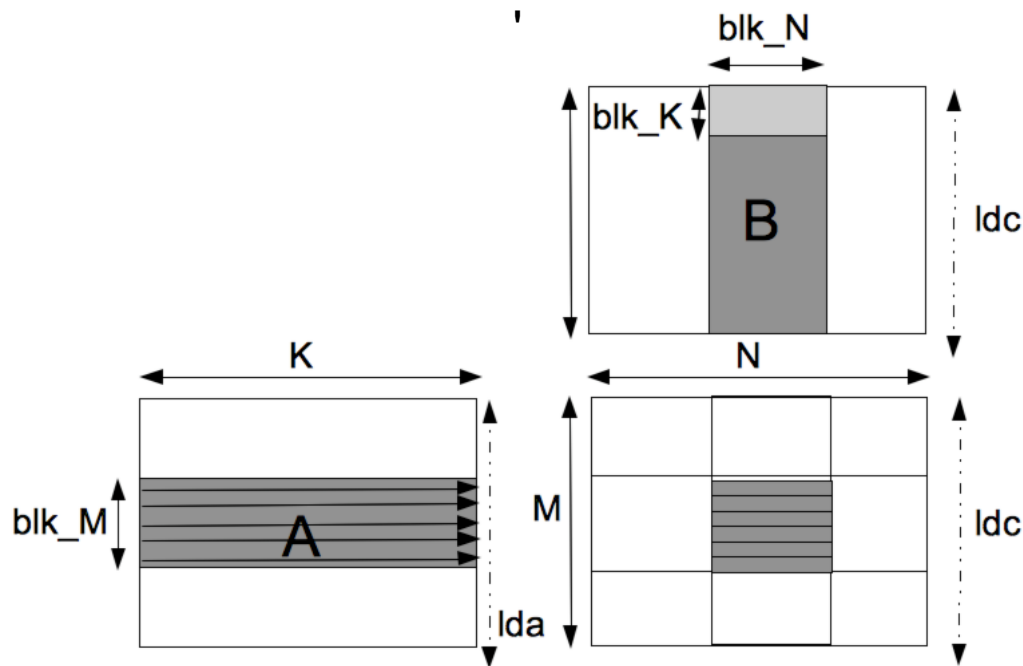
Principle (from Volkov and Demmel)



- ◆ A thread block computes a block of matrix C
- ◆ Each thread computes a row of the block submatrix of C
- ◆ Part of matrix B is loaded into shared memory and computations are done in terms of axpy

xGEMM kernel (2/4) (GTX280 and Tesla C1060)

Parametrization



Parameters:

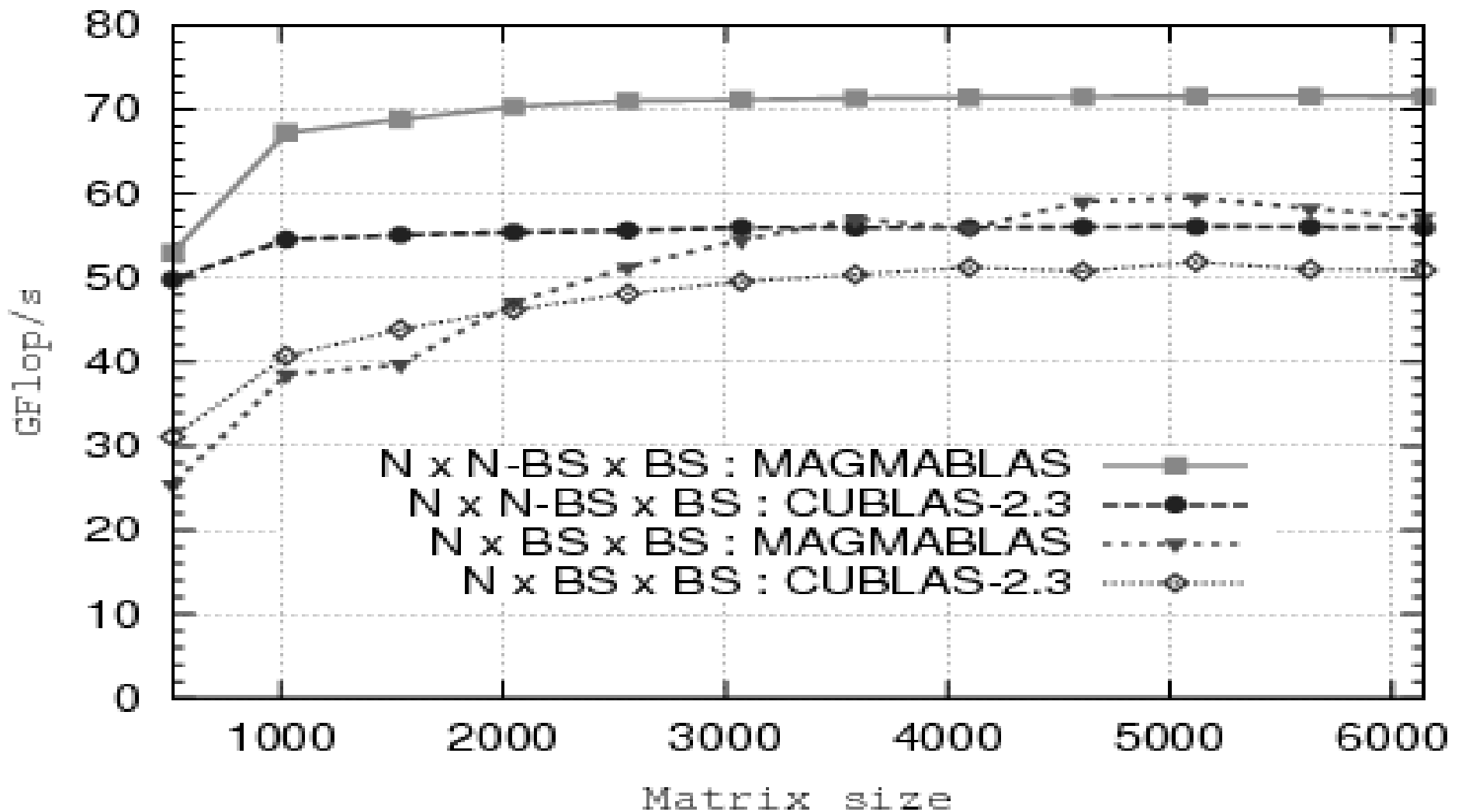
- ◆ Blocking size: BlkM, BlkN
- ◆ Blocking Size in shared memory: BlkK
- ◆ Thread Block Size: NTX , NTY
- ◆ A and/or B in shared memory
- ◆ Amount of allocated shared memory
- ◆ Precision: single or double

Other optimizations:

- ◆ Prefetching into registers
- ◆ ...

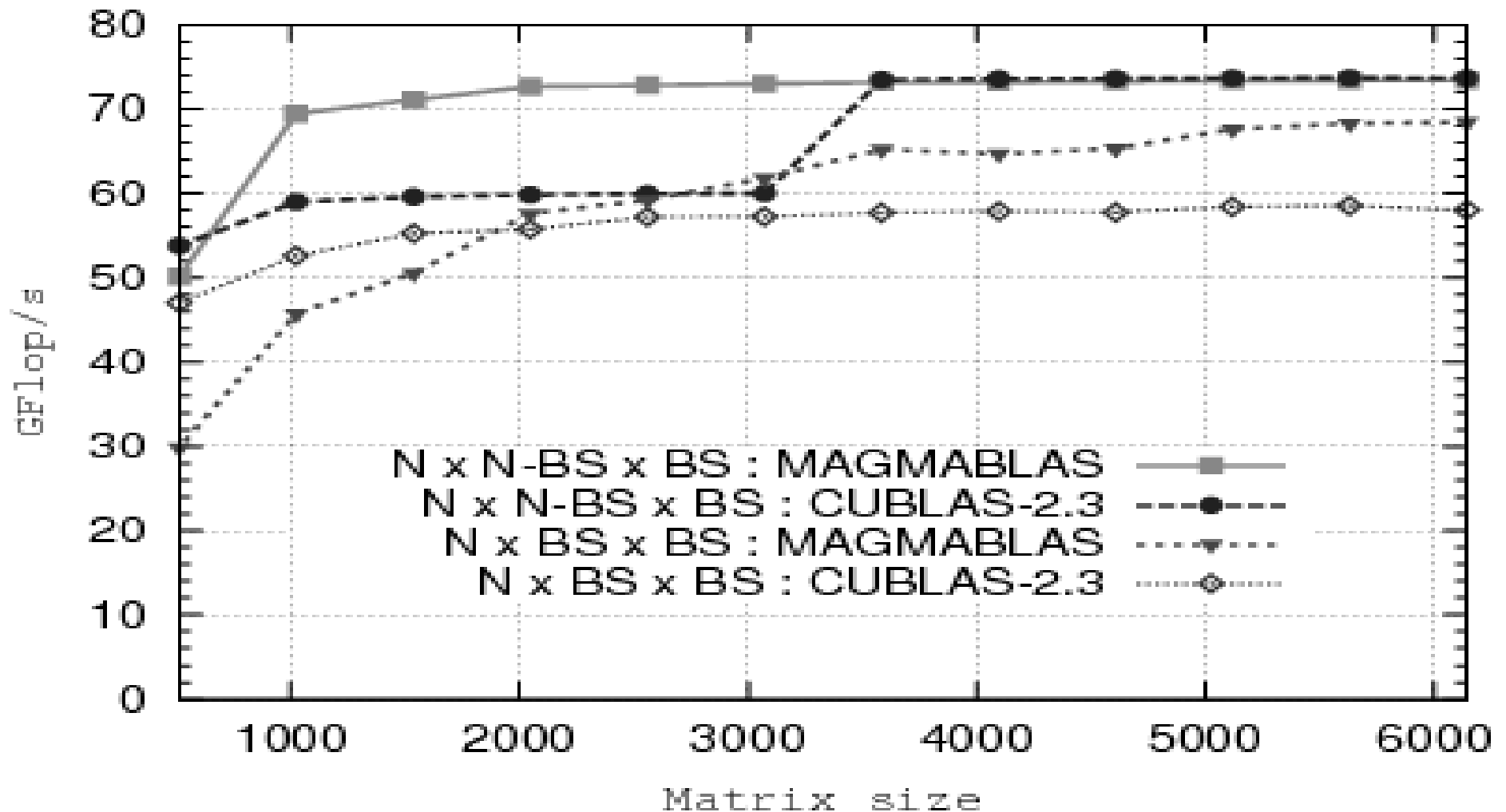
dGEMM kernel (3/4) (GTX280 and Tesla C1060)

Performance on rectangular matrices (BS=64)



dGEMM kernel (4/4) (GTX280 and Tesla C1060)

Performance on rectangular matrices (BS=128)



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”

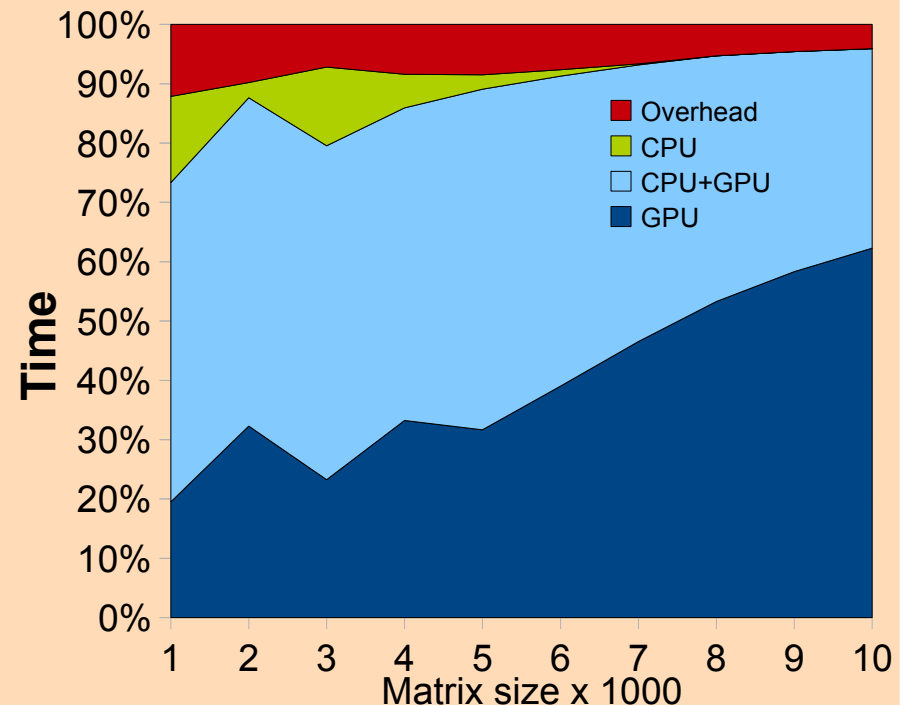
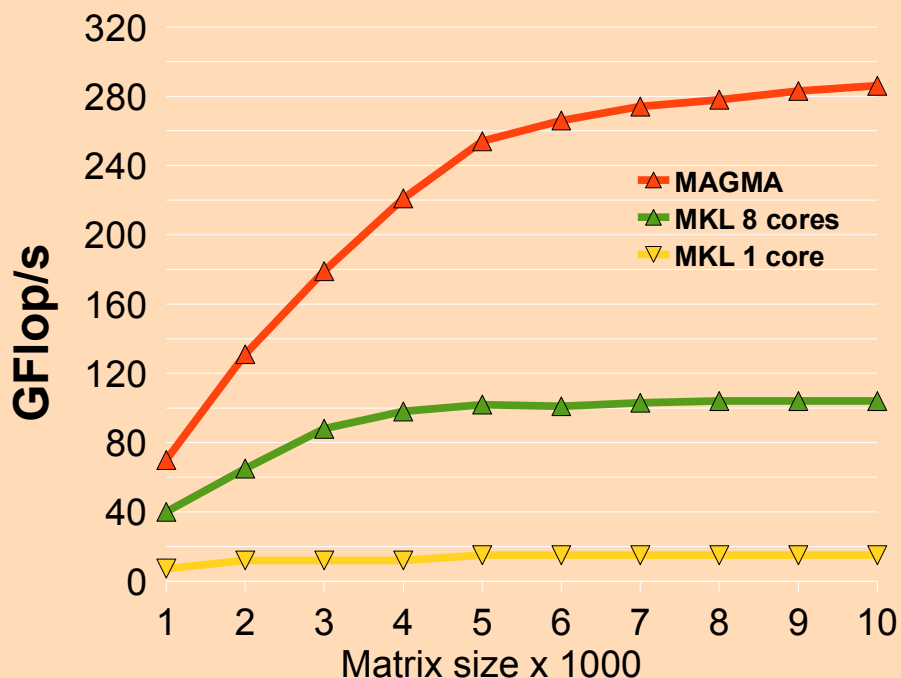
- Note

- ◆ 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)

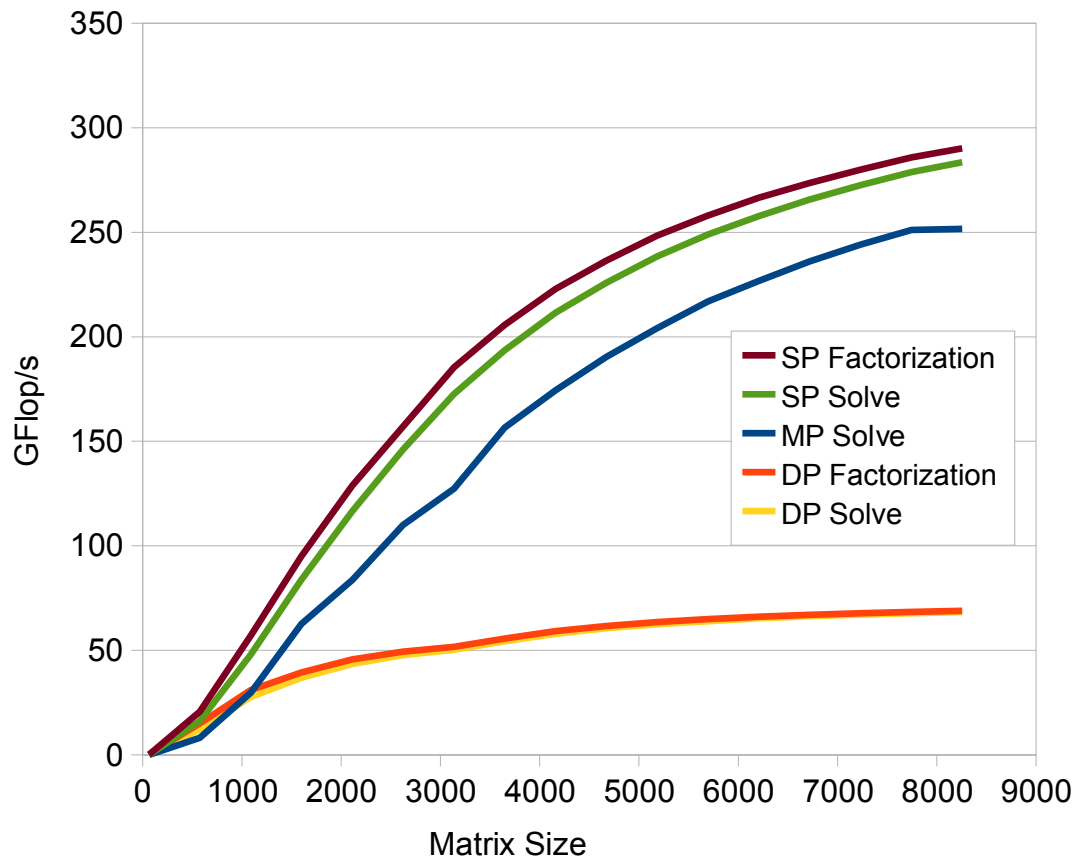
GPU BLAS : CUBLAS 2.2, sgemm peak: 375 GFlop/s
CPU BLAS : MKL 10.0 , sgemm peak: 128 GFlop/s

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

Linear Solvers

Solving $Ax = b$ using LU factorization

Intel(R) Xeon(R)E5410@2.34GHz / 8 Cores + GTX 280 @1.30GHz / 240 Core



- **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 Bordeaux Sud Ouest]
- ◆ Using the **DPLASMA** scheduler

• Rectangular tiles

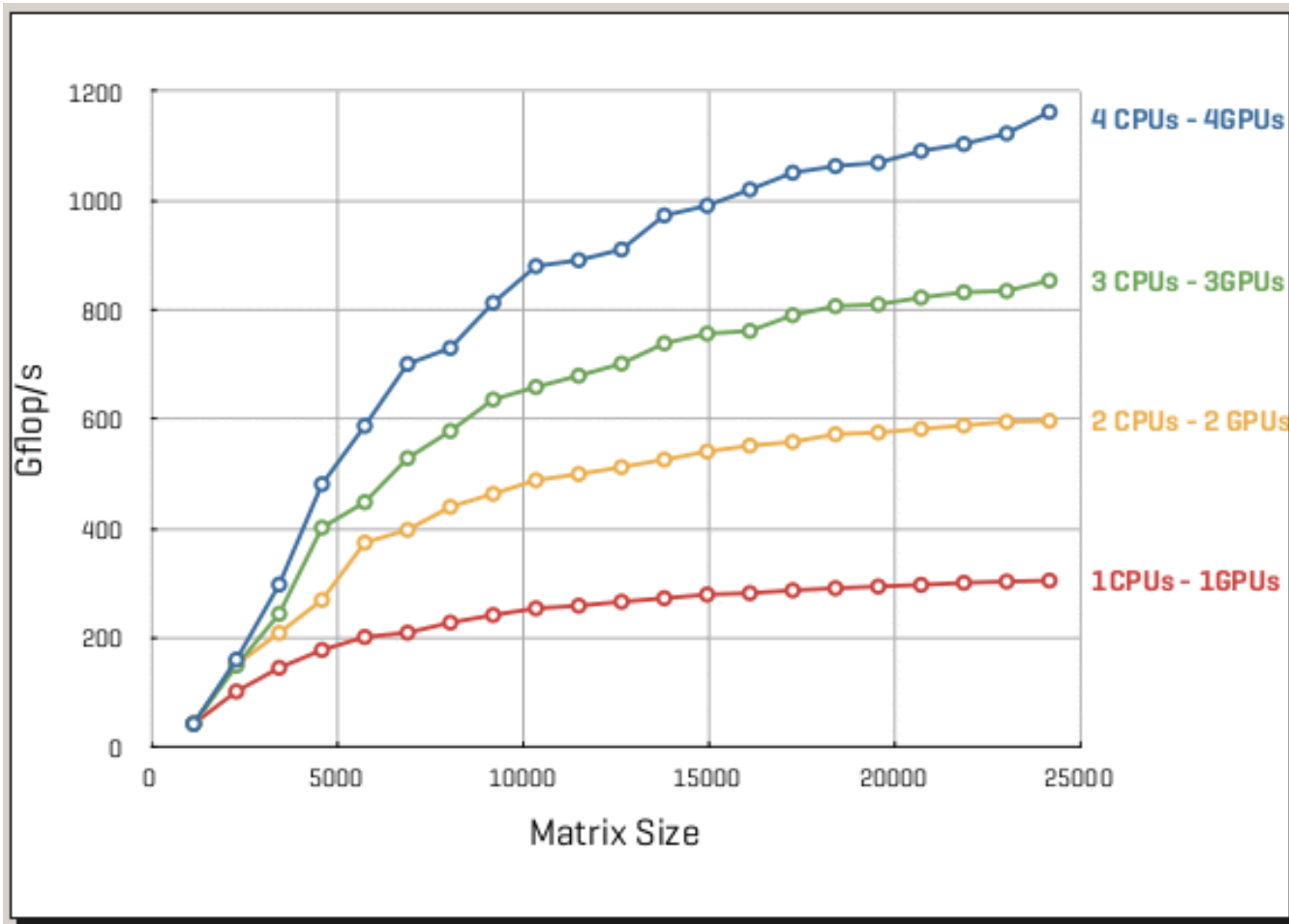
- ◆ Tiles of variable sizes to be used to account for the heterogeneity of the system
- ◆ To experiment with “communication-avoiding” algorithms

• PLASMA tile algorithms

- ◆ A single GPU kernel processing multiple tile tasks in parallel [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 %)

```

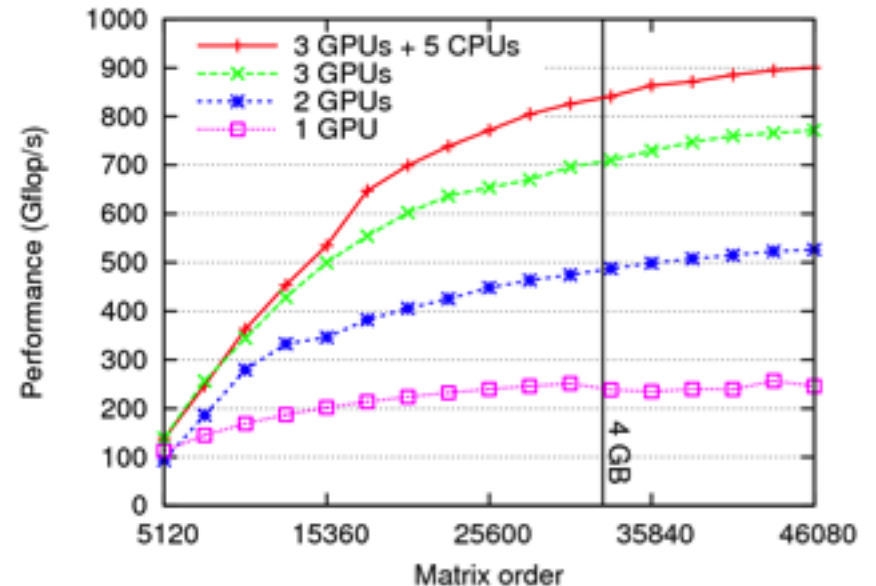
Statistics for codelet sgemm

```

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

```

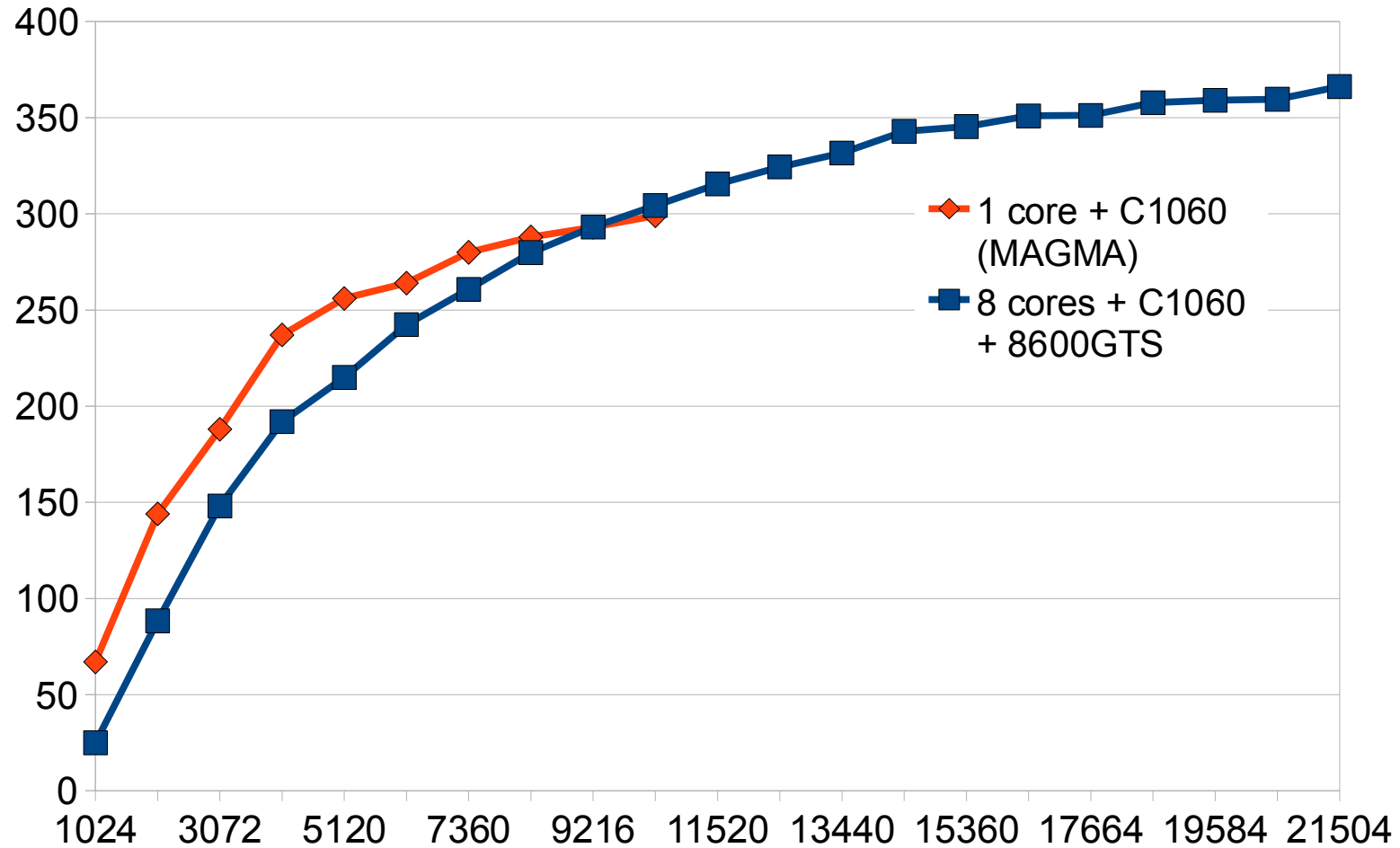
SPOTRF

```

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

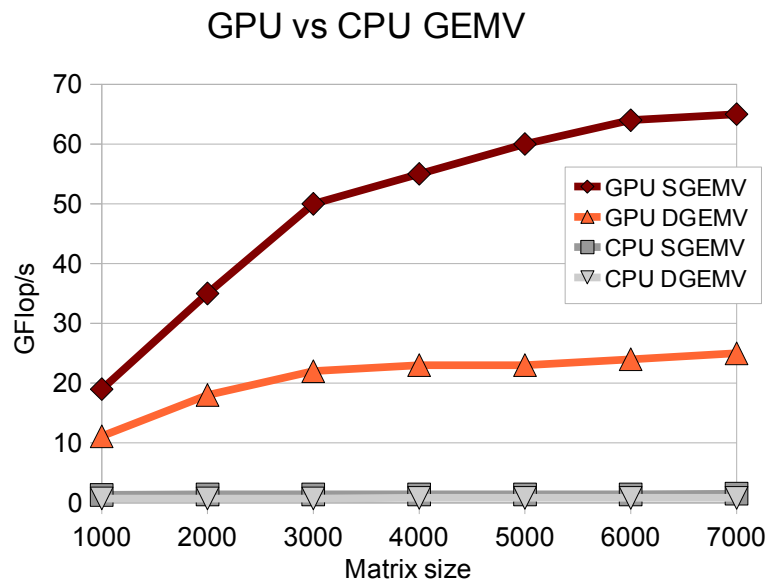
```

Scheduling with 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



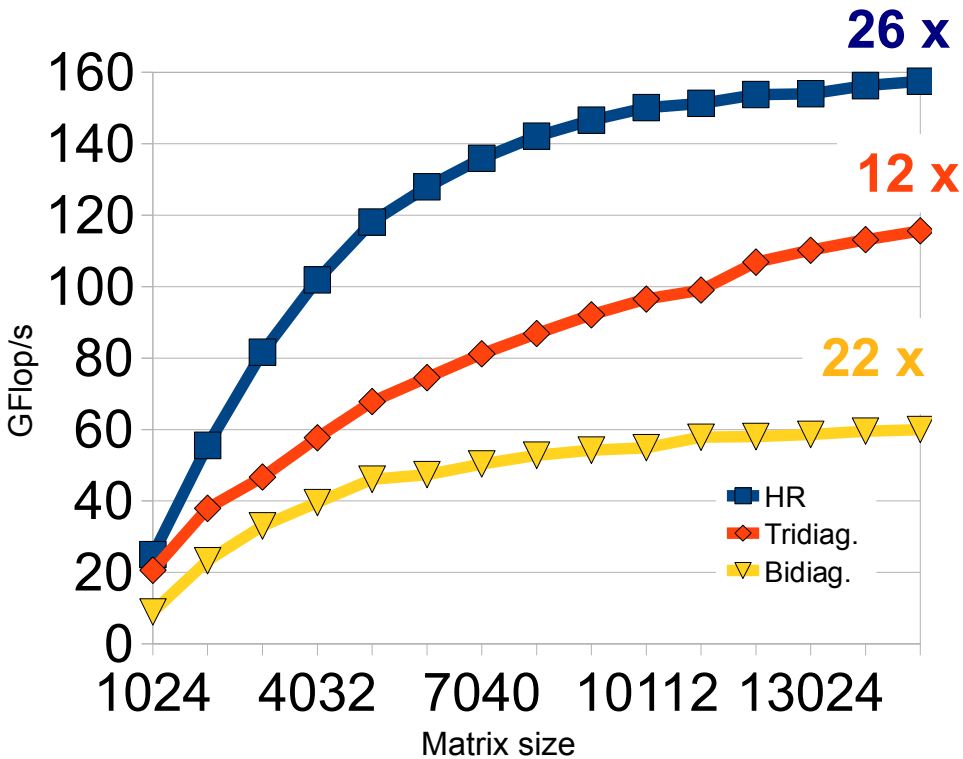
High-performance CUDA kernels were developed for various matrix-vector products [e.g., **ssymv** reaching up to **102 Gflop/s** for the symmetric eigenvalue problem]

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

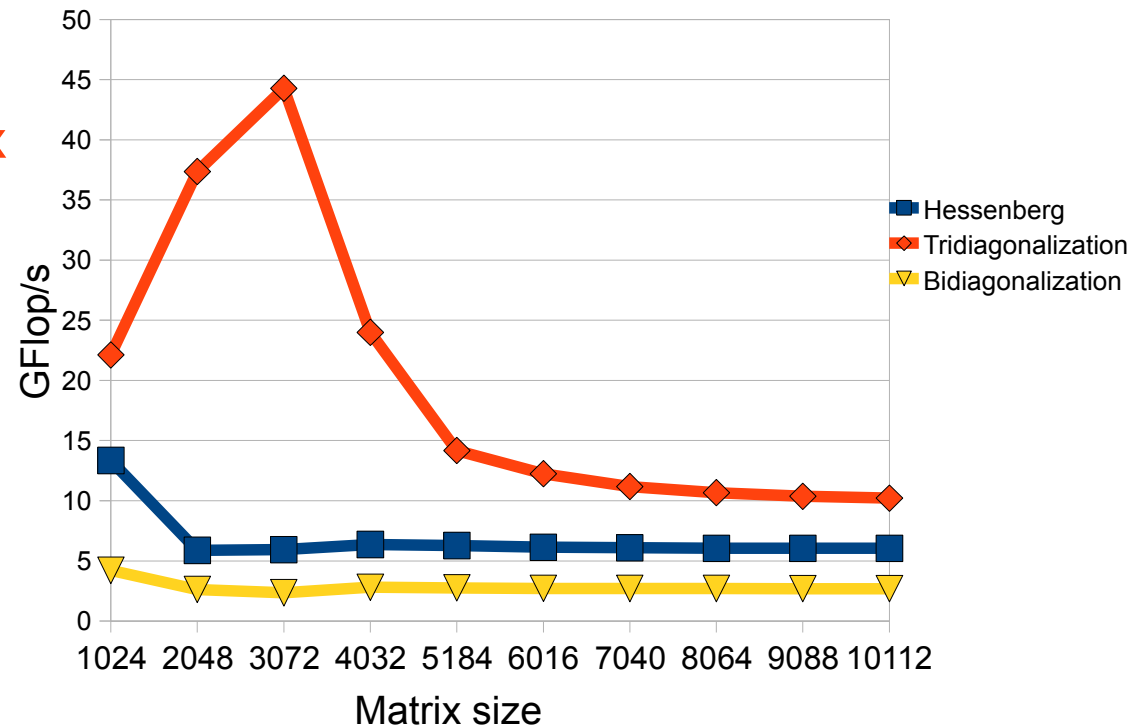
Two-sided factorizations

(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)

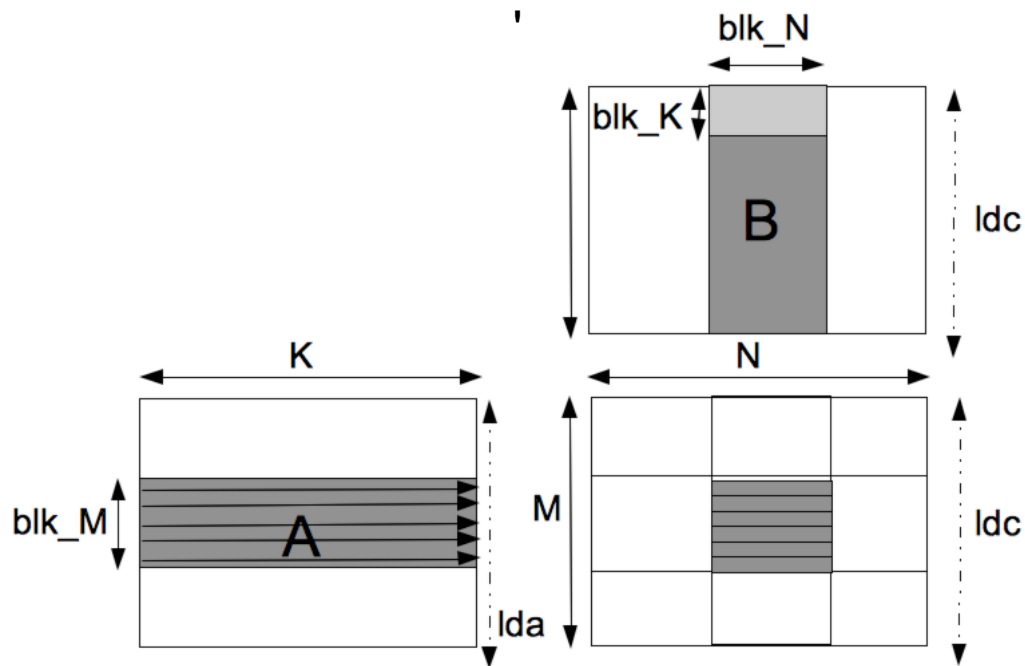
GPU BLAS : CUBLAS 2.3, dgemm peak: 75 GFlop/s
CPU BLAS : MKL 10.0 , dgemm peak: 65 GFlop/s

Auto-tuning: move to FERMI GPU

- **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
 - Shared memory is slower in Fermi
 - Register blocking is performed
 - Both A and B are brought into shared memory
 - Extra parameter: shape of the register blocking, square gave good results (see next slides)
 - **Auto-tuning becomes even more important ... and trickier**

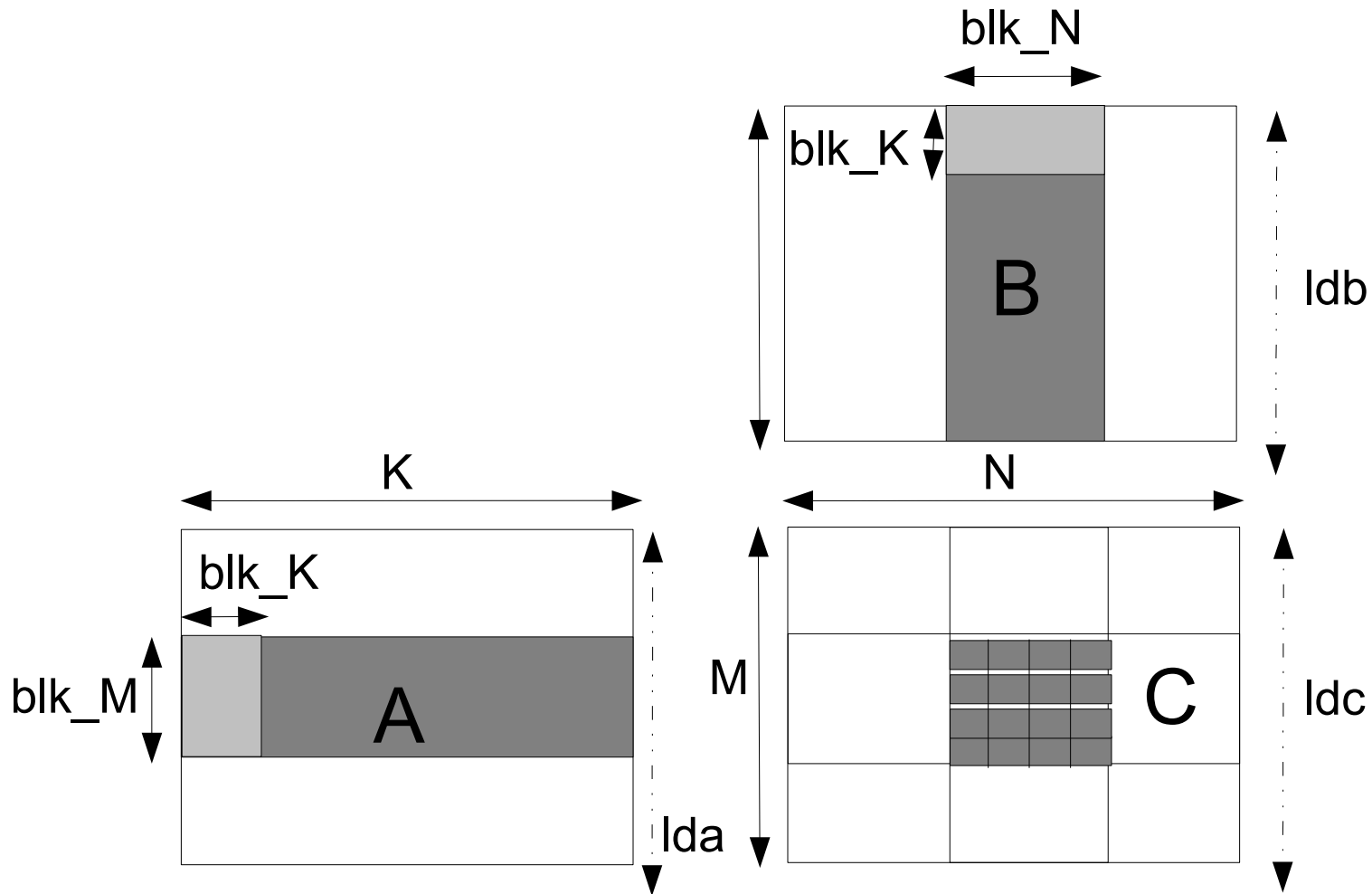
xGEMM kernel (GTX280 and Tesla C1060)

Principle (reminder)

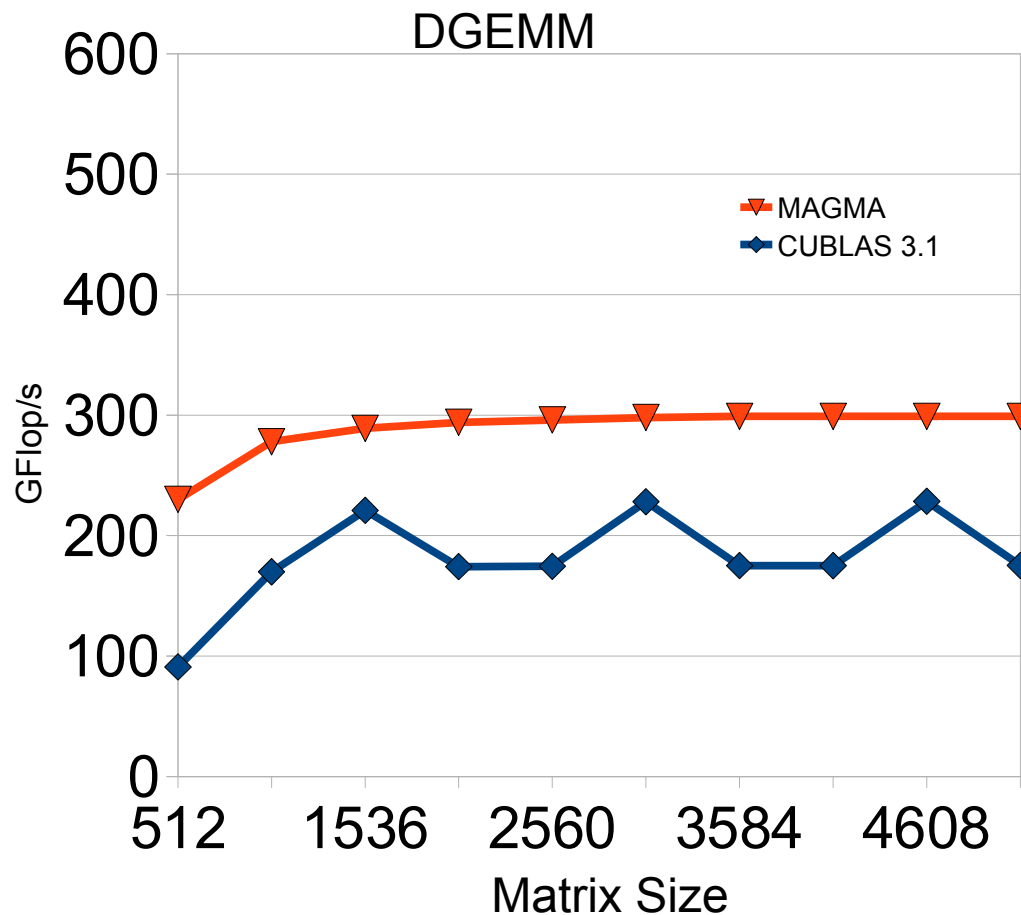
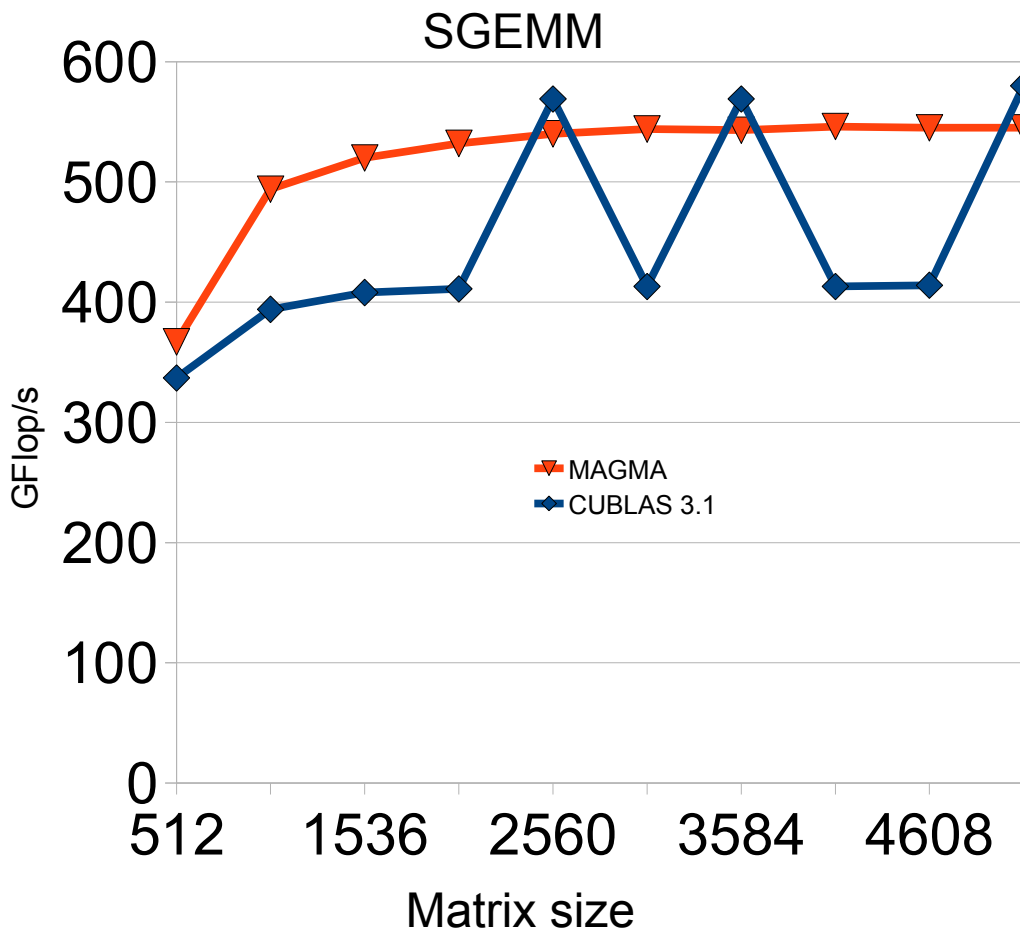


- ◆ A thread block computes a block of matrix C
- ◆ Each thread computes a row of the block submatrix of C
- ◆ Part of matrix B is loaded into shared memory and computations are done in terms of axpy

xGEMM for Fermi (1/2) (Tesla C2050)

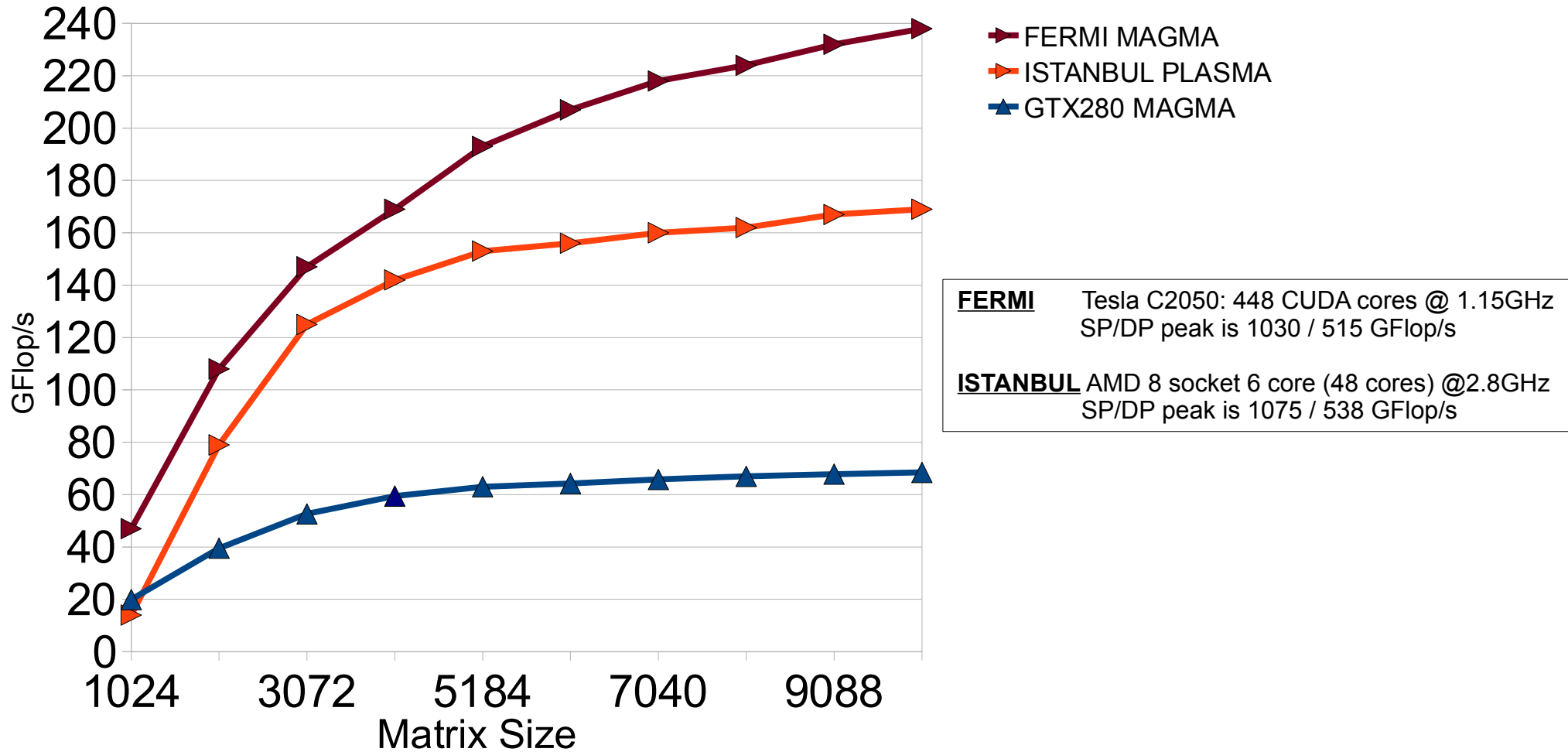


xGEMM for Fermi (2/2) (Tesla C2050)

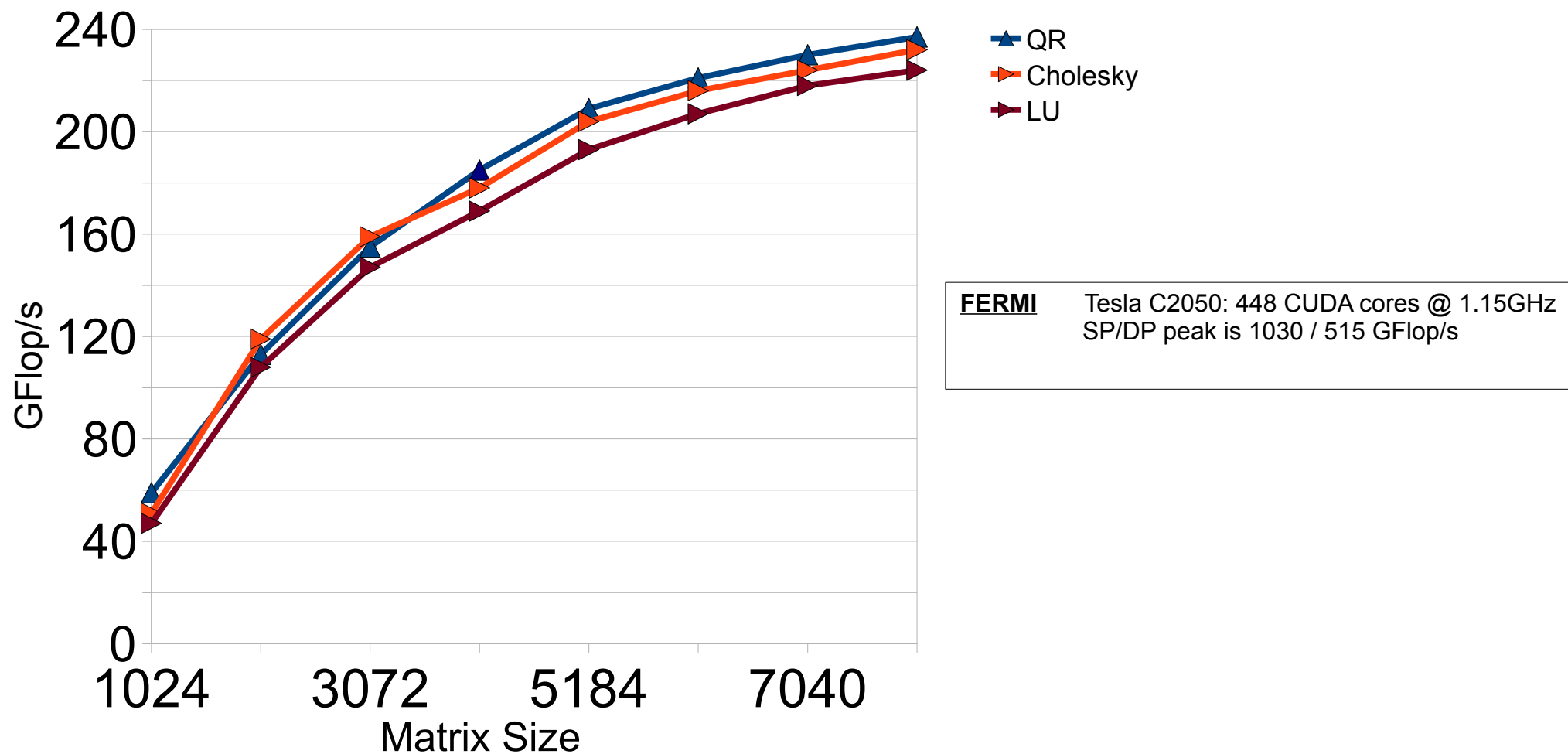


(Tesla C2050: 448 CUDA cores @ 1.15GHz, theoretical SP peak is 1.03 Tflop/s, DP peak 515 GFlop/s)

LU on Fermi in DP



LU, Cholesky, and QR on Fermi in DP



Conclusion

- The performance of high-level routines relies on highly optimized low-level kernels
- The auto-tuning framework is generic
 - But needs some adaptation to take into account new algorithmic schemes as hardware is evolving fast
 - Possible need to rewrite the kernels in assembly (GEMM, SYRK, TRSM, ...)
- Auto-tuning at higher level is of importance too (not discussed here)
 - Blocking size of the panel (1 gpu)
 - Magnum tile size (multi-gpu)

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 Bordeaux Sud Ouest, France

