MAGMA – LAPACK for HPC on Heterogeneous Architectures

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

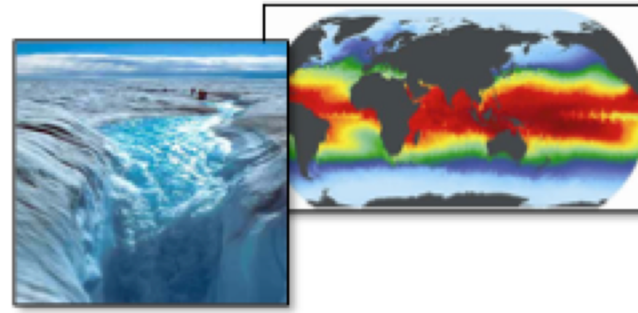
Titan Summit
Oak Ridge Leadership Computing Facility (OLCF)
Oak Ridge National Laboratory, TN
August 15, 2011

Outline

- Motivation
- MAGMA – LAPACK for GPUs
    - Overview
    - Methodology
    - MAGMA with various schedulers
- MAGMA BLAS
- Current & future work directions

Science and Engineering Drivers

- *Climate Change*: Understanding, mitigating and adapting to the effects of global warming
  - Sea level rise
  - Severe weather
  - Regional climate change
  - Geologic carbon sequestration
- *Energy*: Reducing U.S. reliance on foreign energy sources and reducing the carbon footprint of energy production
  - Reducing time and cost of reactor design and deployment
  - Improving the efficiency of combustion energy sources
- *National Nuclear Security*: Maintaining a safe, secure and reliable nuclear stockpile
  - Stockpile certification
  - Predictive scientific challenges
  - Real-time evaluation of urban nuclear detonation

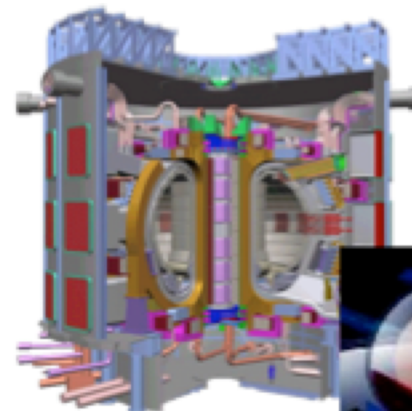


Accomplishing these missions requires exascale resources.

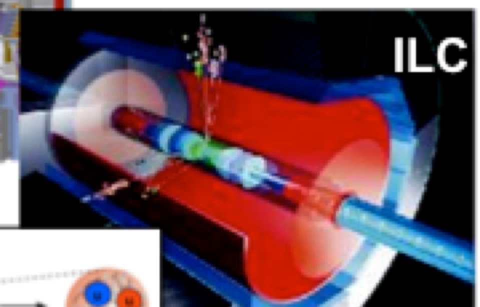
Simulation enables fundamental advances in basic science

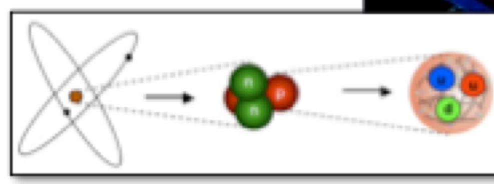
- **Nuclear Physics**
  - Quark-gluon plasma & nucleon structure
  - Fundamentals of fission and fusion reactions
- **Facility and experimental design**
  - Effective design of accelerators
  - Probes of dark energy and dark matter
  - ITER shot planning and device control
- **Materials / Chemistry**
  - Predictive multi-scale materials modeling: observation to control
  - Effective, commercial, renewable energy technologies, catalysts and batteries
- **Life Sciences**
  - Better biofuels
  - Sequence *to* structure *to* function

These breakthrough scientific discoveries and facilities require exascale applications and resources.

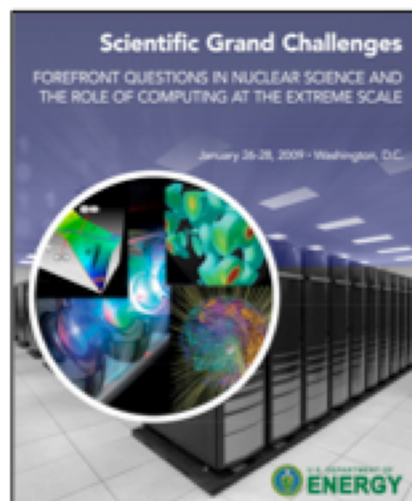


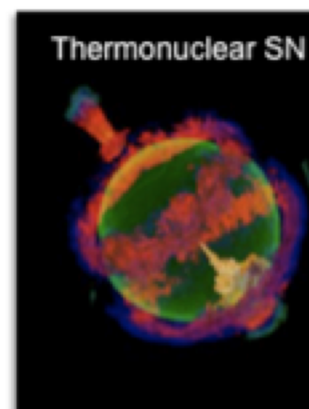
ITER





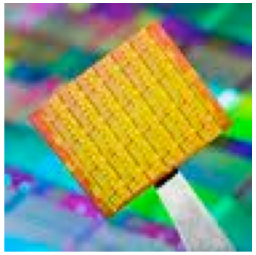
Structure of nucleons



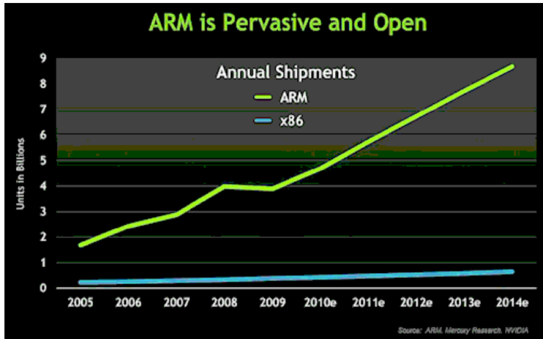


Future Computer Systems

- 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.



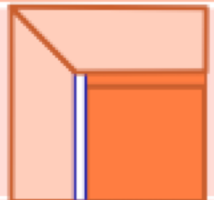
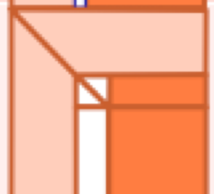
AMD
The future is fusion



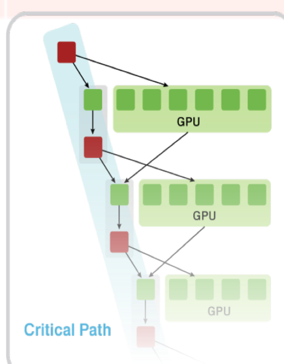
Major change to Software

- 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 Next Generation of Software

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

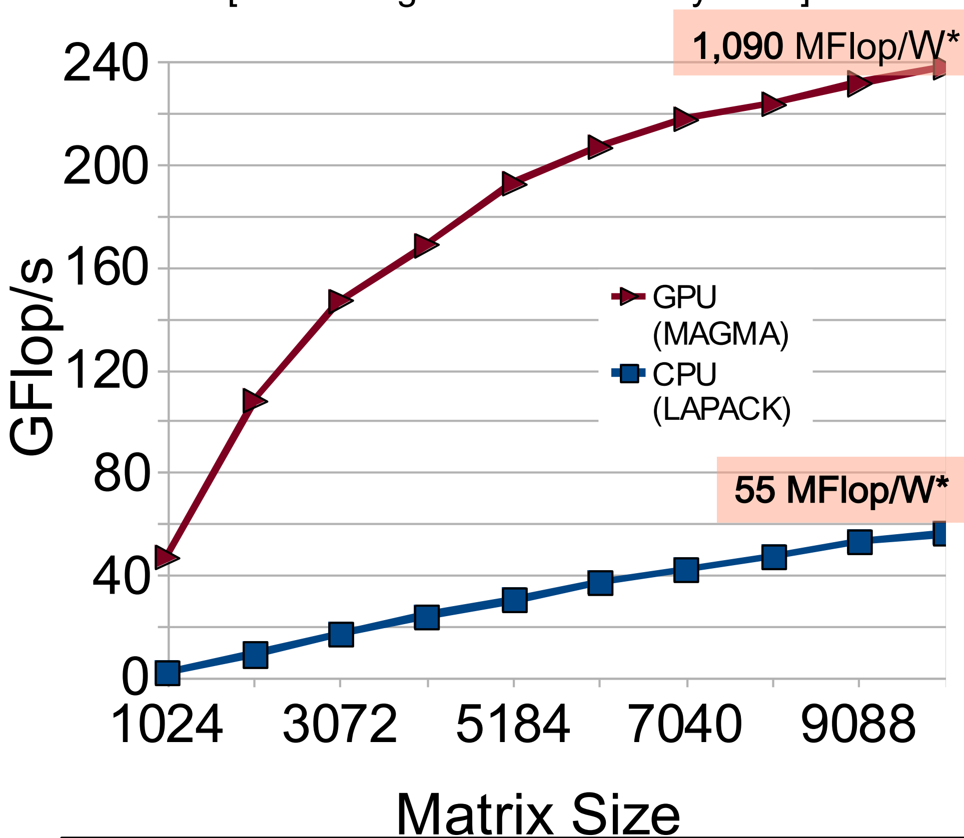
MAGMA
Hybrid Algorithms
(heterogeneity friendly)



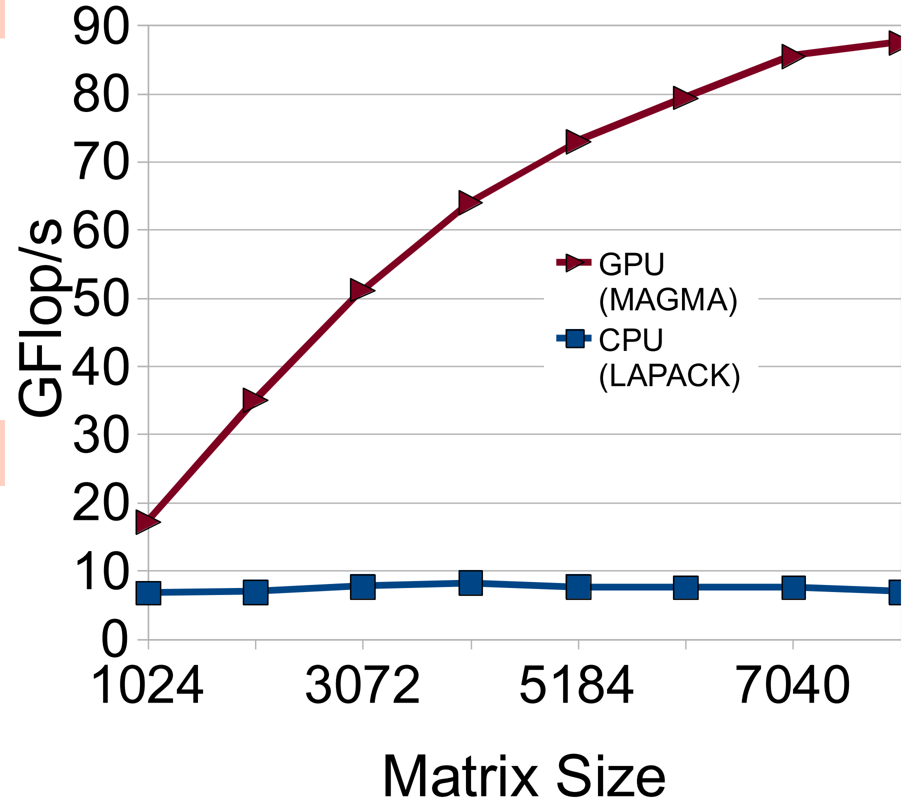
Rely on
- hybrid scheduler (of DAGs)
- hybrid kernels
  (for nested parallelism)
- existing software infrastructure

HPC @ 1/10th the cost & 1/20th the power

LU Factorization in double precision (DP)
[for solving a dense linear system]



Hessenberg factorization in DP
[for the general eigenvalue problem]



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]

	GPU	CPU
DP peak	**515 + 40** GFlop/s	**538** GFlop/s
System cost	~ $**3,000**	~ $**30,000**
Power *	~ **220** W	~ **1,022** W

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

Matrix Algebra on GPU and Multicore Architectures (MAGMA)

- MAGMA: **a collection of next 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/

- **Key features**
  - **Top performance and high accuracy (LAPACK compliant)**
  - **Multiple precision arithmetic (S/D/C/Z & mixed)**
  - **Hybrid algorithms using both multicore CPUs and accelerators (GPUs)**

- **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]

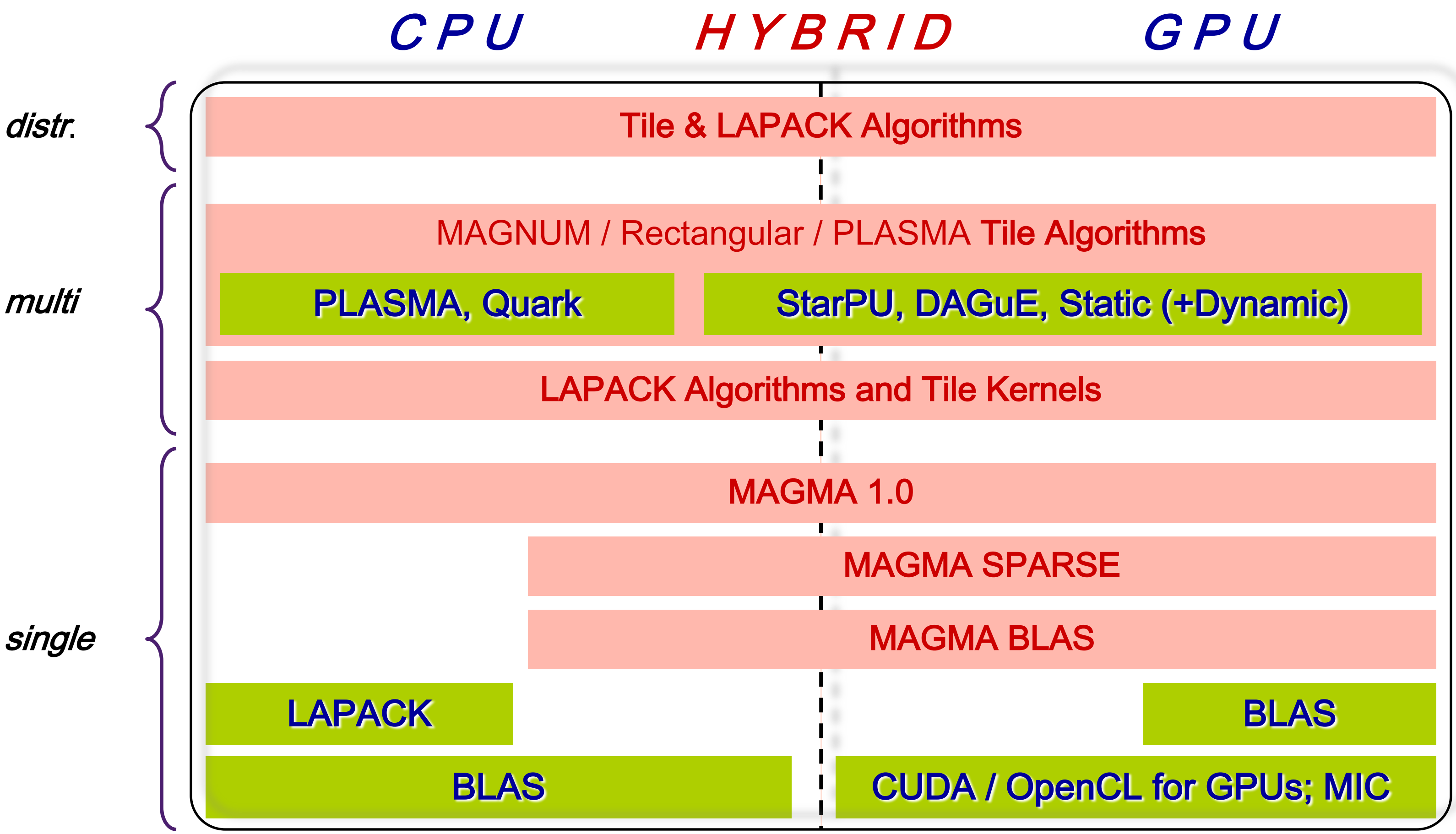
Challenges of using GPUs

- **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]

MAGMA Software Stack



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

MAGMA 1.0

- 35+ algorithms are developed (total of 130+ routines)
  - Every algorithm is in 4 precisions (s/c/d/z, denoted by X)
    - One-sided factorizations and solvers
    - Two-sided factorizations and eigen/singular-value solvers
  - There are 3 mixed precision algorithms (zc & ds, denoted by XX)
  - These are hybrid algorithms
    - Expressed in terms of BLAS
- Support is for single CUDA-enabled NVIDIA GPU, either Tesla or Fermi
- MAGMA BLAS
  - A subset of GPU BLAS, optimized for Tesla and Fermi GPUs

MAGMA 1.0

One-sided factorizations

1. Xgetrf	LU factorization; CPU interface
2. Xgetrf_gpu	LU factorization; GPU interface
3. Xgetrf_mc	LU factorization on multicore (no GPUs)
4. Xpotrf	Cholesky factorization; CPU interface
5. Xpotrf_gpu	Cholesky factorization; GPU interface
6. Xpotrf_mc	Cholesky factorization on multicore (no GPUs)
7. Xgeqrf	QR factorization; CPU interface
8. Xgeqrf_gpu	QR factorization; GPU interface; with T matrices stored
9. Xgeqrf2_gpu	QR factorization; GPU interface; without T matrices
10. Xgeqrf_mc	QR factorization on multicore (no GPUs)
11. Xgeqrf2	QR factorization; CPU interface
12. Xgeqlf	QL factorization; CPU interface
13. Xgelqf	LQ factorization; CPU interface

MAGMA 1.0

Linear solvers

14. Xgetrs_gpu	Work precision; using LU factorization; GPU interface
15. Xpotrs_gpu	Work precision; using Cholesky factorization; GPU interface
16. Xgels_gpu	Work precision LS; GPU interface
17. XXgetrs_gpu	Mixed precision iterative refinement solver; Using LU factorization; GPU interface
18. XXpotrs_gpu	Mixed precision iterative refinement solver; Using Cholesky factorization; GPU interface
19. XXgeqrsv_gpu	Mixed precision iterative refinement solver; Using QR on square matrix; GPU interface

MAGMA 1.0

Two-sided factorizations

20. Xgehrd	Reduction to upper Hessenberg form; with T matrices stored; CPU interface
21. Xgehrd2	Reduction to upper Hessenberg form; Without the T matrices stored; CPU interface
22. Xhetrd	Reduction to tridiagonal form; CPU interface
23. Xgebrd	Reduction to bidiagonal form; CPU interface

MAGMA 1.0

Generating/applying orthogonal matrices

24. Xungqr	Generates Q with orthogonal columns as the product of elementary reflectors (from Xgeqrf); CPU interface
25. Xungqr_gpu	Generates Q with orthogonal columns as the product of elementary reflectors (from Xgeqrf_gpu); GPU interface
26. Xunmtr	Multiplication with the orthogonal matrix, product of elementary reflectors from Xhetrd; CPU interface
27. Xunmtr_gpu	Multiplication with the orthogonal matrix, product of elementary reflectors from Xhetrd; GPU interface
28. Xunmqr	Multiplication with orthogonal matrix, product of elementary reflectors from Xgeqrf; CPU interface
29. Xunmqr_gpu	Multiplication with orthogonal matrix, product of elementary reflectors from Xgeqrf_gpu; GPU interface
30. Xunghr	Generates Q with orthogonal columns as the product of elementary reflectors (from Xgehrd); CPU interface
31. Xunghr_gpu	Generates Q with orthogonal columns as the product of elementary reflectors (from Xgehrd); GPU interface

MAGMA 1.0

Eigen/singular-value solvers

32. Xgeev	Solves the non-symmetric eigenvalue problem; CPU interface
33. Xheevd	Solves the Hermitian eigenvalue problem; Uses devide and conquer; CPU interface
34. Xgesvd	SVD; CPU interface

- **Currently, these routines have GPU-acceleration for the**
  - **two-sided factorizations used and the**
  - **Orthogonal transformation related to them (matrix generation/application from the previous slide)**

MAGMA BLAS

- Subset of BLAS for a single NVIDIA GPU
- Optimized for MAGMA specific algorithms
- To complement CUBLAS on special cases

MAGMA BLAS

Level 2 BLAS

1. Xgemv_tesla	General matrix-vector product for Tesla
2. Xgemv_fermi	General matrix-vector product for Fermi
3. Xsymv_ tesla	Symmetric matrix-vector product for Tesla
4. Xsymv_fermi	Symmetric matrix-vector product for Fermi

MAGMA BLAS

Level 3 BLAS

5. Xgemm_tesla	General matrix-matrix product for Tesla
6. Xgemm_fermi	General matrix-matrix product for Fermi
7. Xtrsm_ tesla	Solves a triangular matrix problem on Tesla
8. Xtrsm_fermi	Solves a triangular matrix problem on Fermi
9. Xsyrk_tesla	Symmetric rank k update for Tesla
10. Xsyr2k_tesla	Symmetric rank 2k update for Tesla

- **CUBLAS GEMMs for Fermi are based on the MAGMA implementation**
- **Further improvements**
  - Autotuned GEMM for Fermi (J.Kurzak)
  - ZGEMM from 308 Gflop/s is now 341 Gflop/s

MAGMA BLAS

Other routines

11. Xswap	LU factorization; CPU interface
12. Xlacpy	LU factorization; GPU interface
13. Xlange	LU factorization on multicore (no GPUs)
14. Xlanhe	Cholesky factorization; CPU interface
15. Xtranspose	Cholesky factorization; GPU interface
16. Xinplace_transpose	Cholesky factorization on multicore (no GPUs)
17. Xpermute	QR factorization; CPU interface
18. Xauxiliary	QR factorization; GPU interface; with T matrices stored

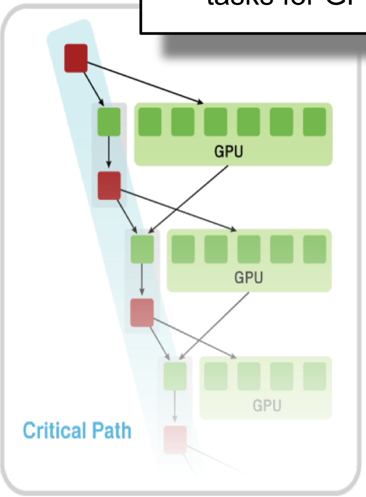
Methodology overview

- **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

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



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

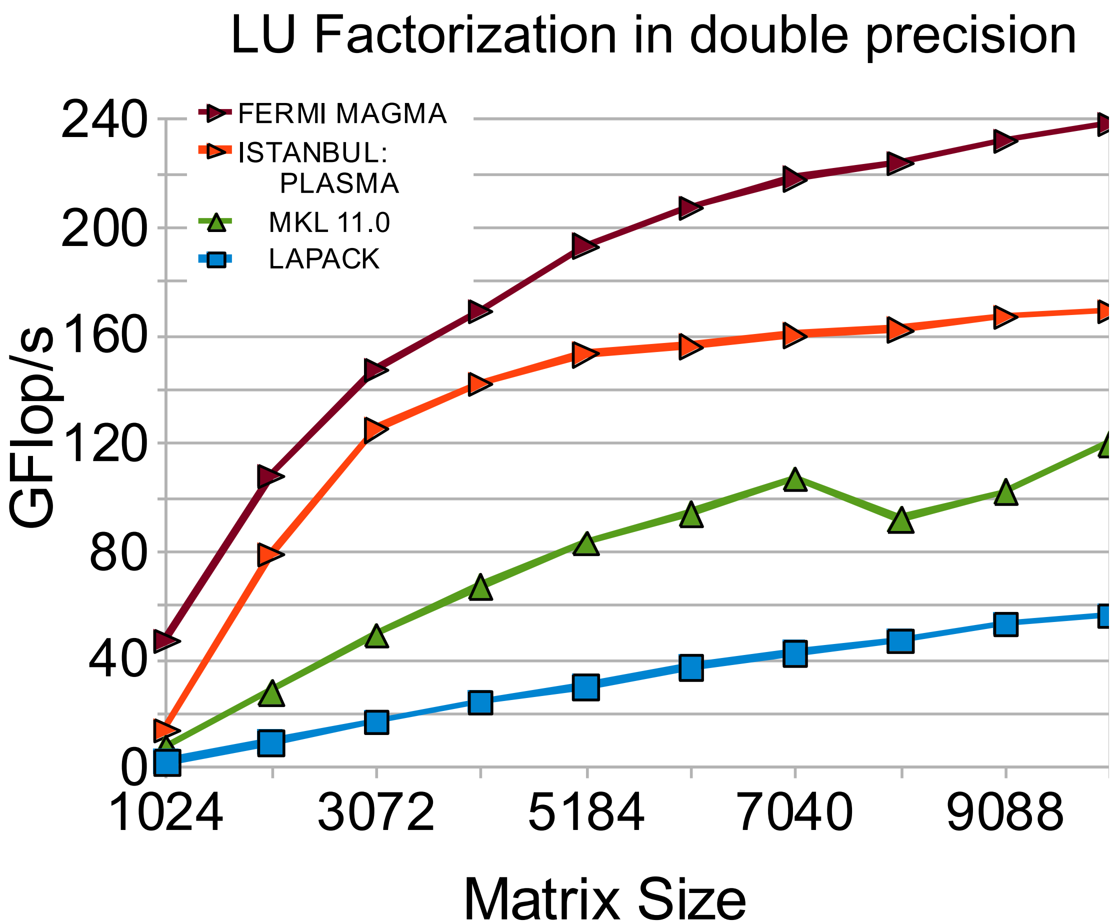
A hybrid algorithm example

- 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)

Results – one sided factorizations



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

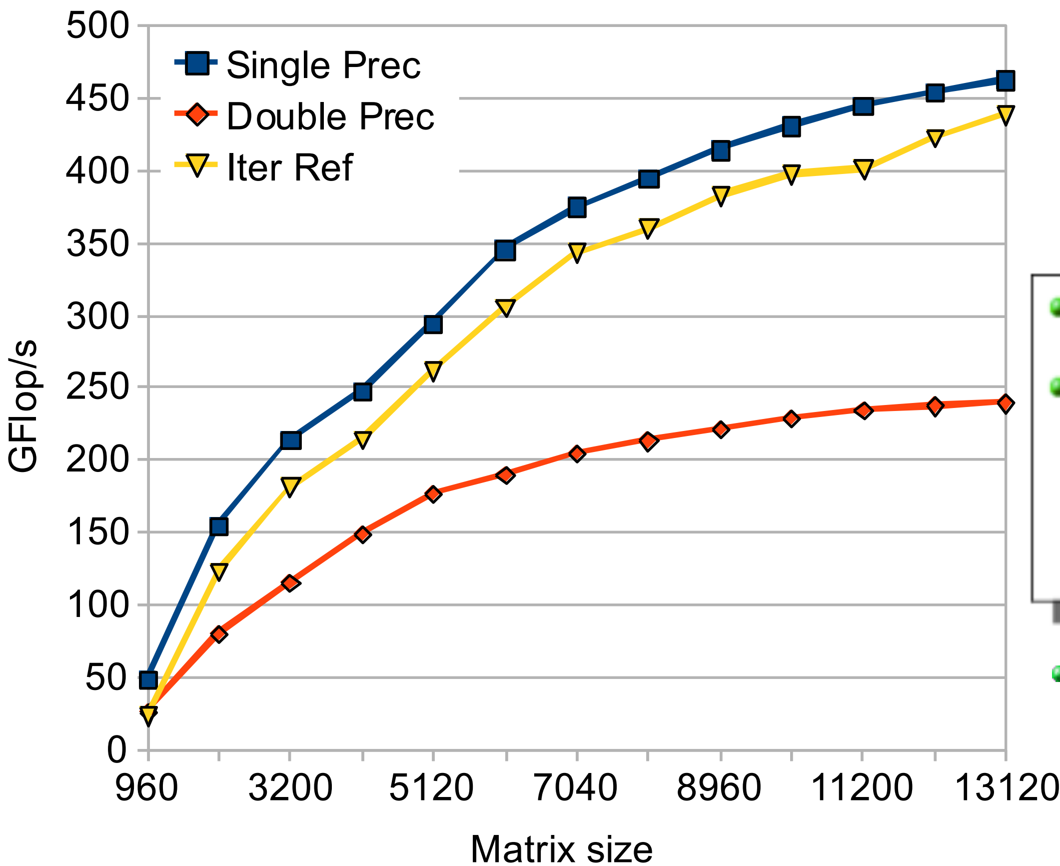
ISTANBUL AMD 8 socket 6 core (48 cores) @2.8GHz
SP/DP peak is 1075 / 538 GFlop/s

- Similar results for Cholesky & QR
- Fast solvers (several innovations)
  - in working precision, and
  - mixed-precision iter. refinement

  based on the one-sided factor.

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

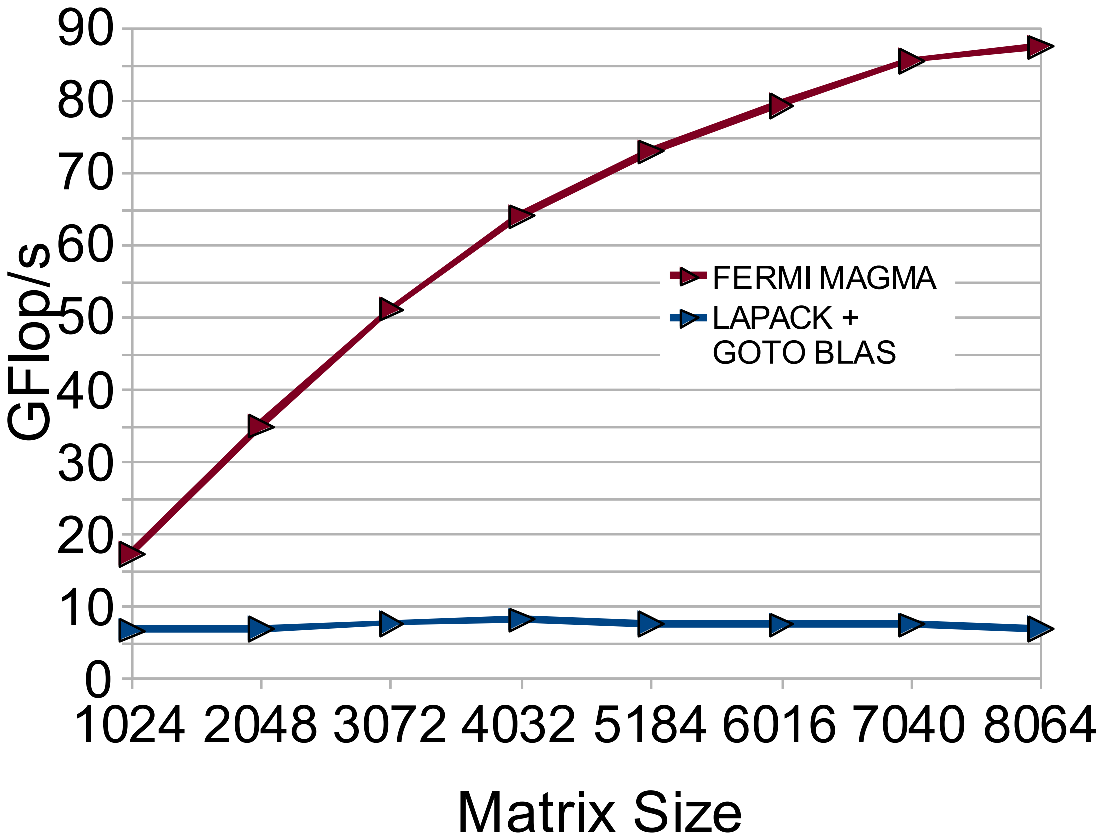
- **Direct solvers**
  - Factor and solve in working precision
- **Mixed Precision Iterative Refinement**
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g.

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

- Similar results for Cholesky & QR

Results – two sided factorizations

Hessenberg Factorization in double precision
[for the general eigenvalue problem]



FERMI Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 Gflop/s
[system cost ~ $3,000]

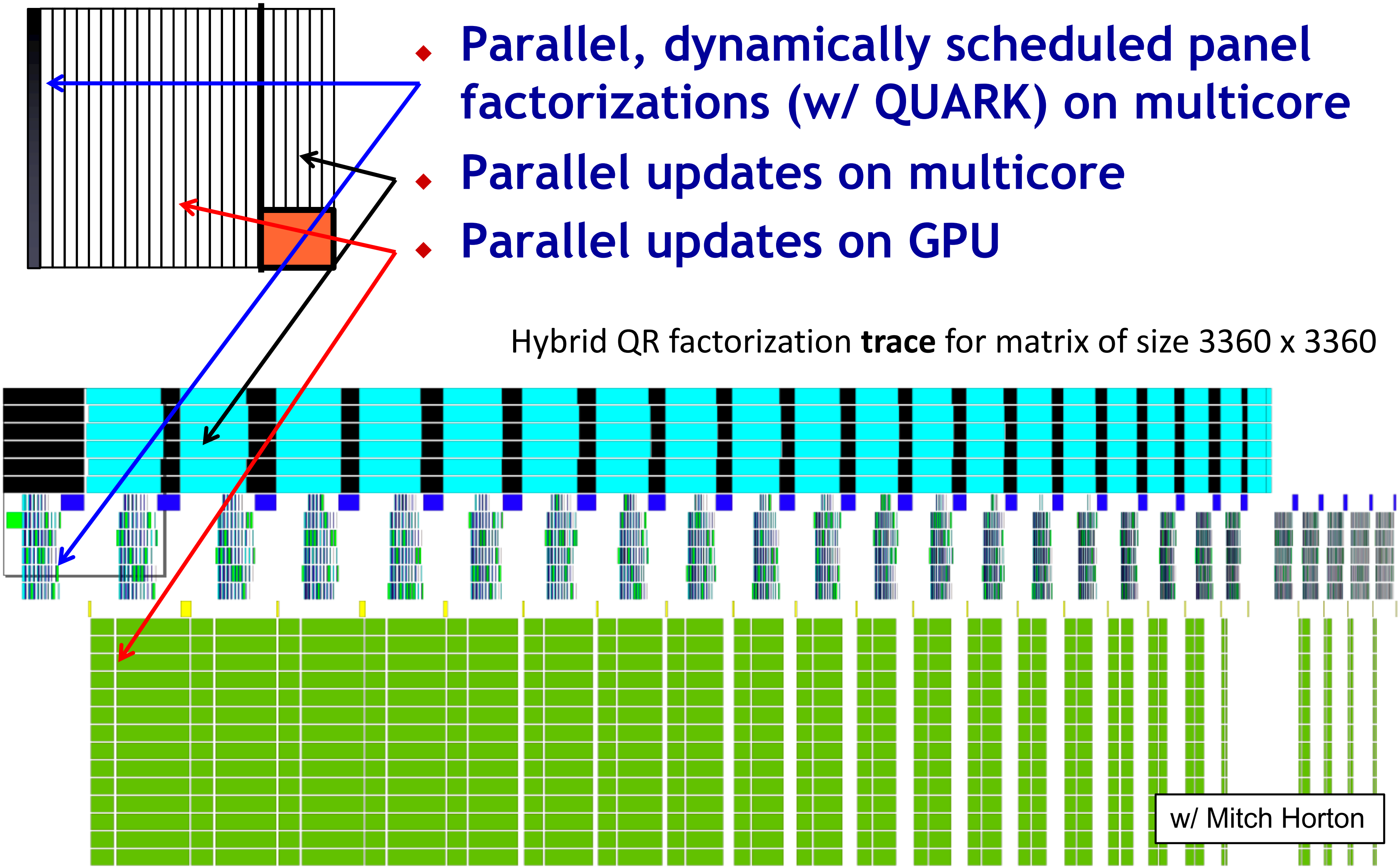
ISTANBUL AMD 8 socket 6 core (48 cores) @2.8GHz
SP/DP peak is 1075 / 538 Gflop/s
[system cost ~ $30,000]

- Similar accelerations for the bidiagonal factorization [for SVD] & tridiagonal factorization [for the symmetric eigenvalue problem]
- Similar acceleration (exceeding 10x) compared to other top-of-the-line multicore systems (including Nehalem-based) and libraries (including MKL, ACML)

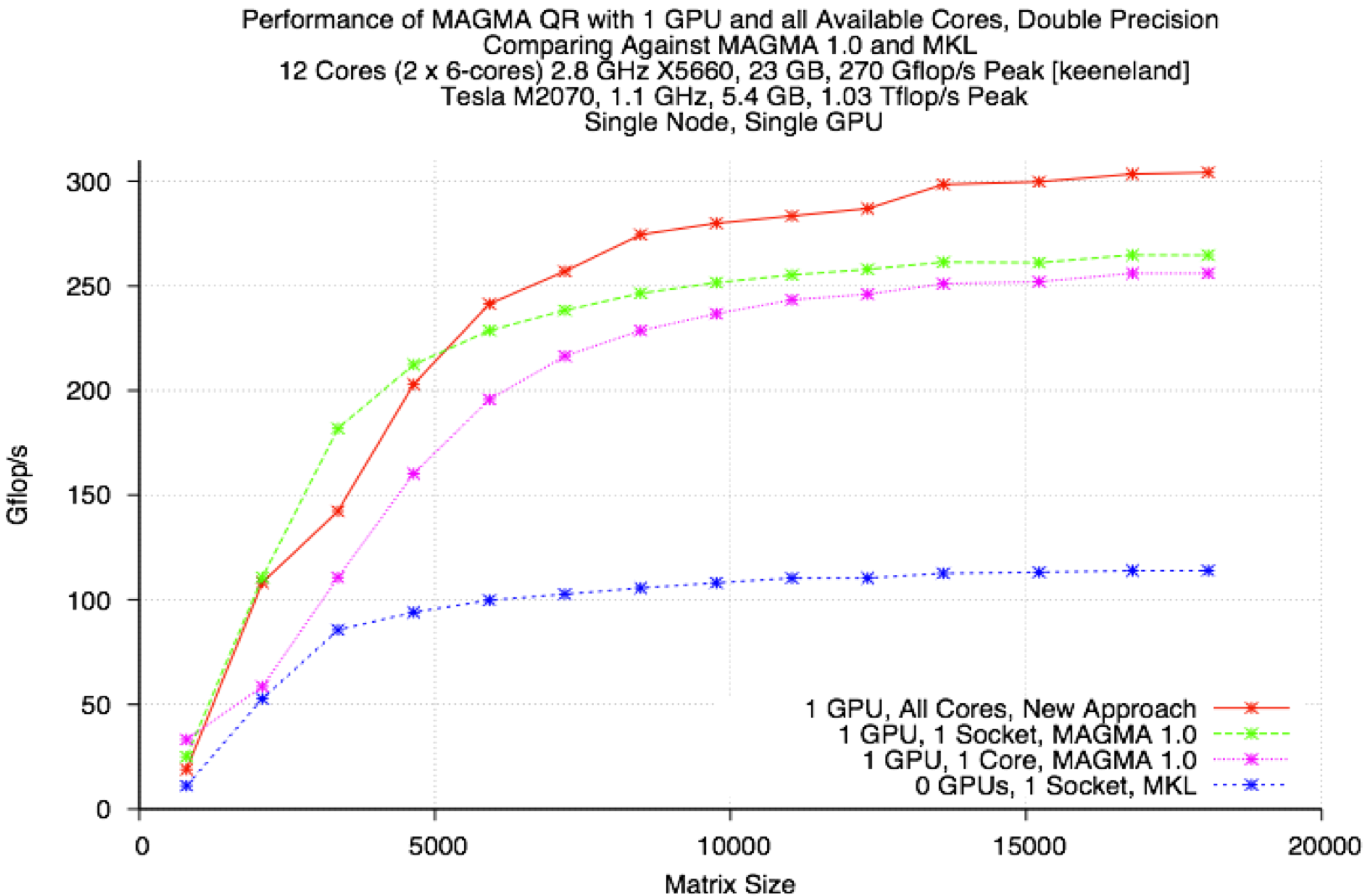
GPU + Multicore one-sided factorizations

- Parallel, dynamically scheduled panel factorizations (w/ QUARK) on multicore
- Parallel updates on multicore
- Parallel updates on GPU

Hybrid QR factorization **trace** for matrix of size 3360 x 3360



A QR for GPU + Multicore



Multicore + multiGPU tiled algorithms

- Reuse already developed kernels
  - Hybrid MAGMA 1.0 for single GPU
  - PLASMA for multicore
- We have developed tiled one-sided factorization algorithms
- Use various run time systems to schedule the kernels' execution
  - StarPU (Dynamic scheduling on a node)
  - QUARK (Dynamic on multicore node from PLASMA)
  - Static + Dynamic scheduling
  - DAGuE

Tiled algorithms with StarPU

- Productivity – easy to develop parallel multicore & multiGPU algorithms from sequential algorithms

```
// 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])
```

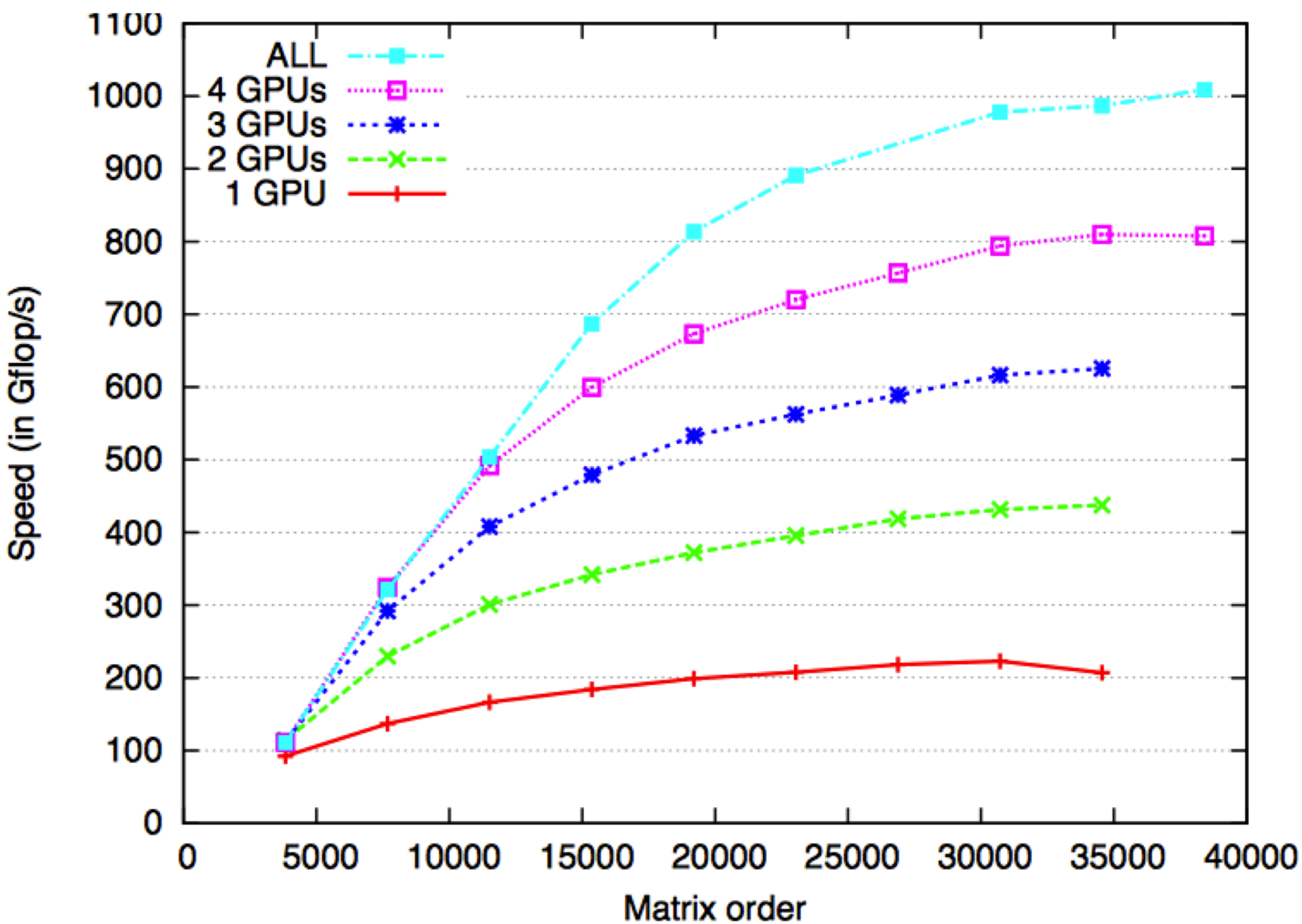
```
// Hybrid Tile Cholesky
FOR k = 0..TILES-1
    starpu_Insert_Task(DPOTRF, ...)
    FOR m = k+1..TILES-1
        starpu_Insert_Task(DTRSM, ...)
        FOR n = k+1..TILES-1
            starpu_Insert_Task(DSYRK, ...)
            FOR m = n+1..TILES-1
                starpu_Insert_Task(DGEMM, ...)
```

- Developed are LU, QR, and Cholesky factorization algorithms
- The kernels needed are available to use w/ other schedulers

w/ E. Aggulo, C. Augonnet, M. Faverge, H. Ltaief

MAGMA with StarPU

- **QR factorization**
  - System: 16 CPUs (AMD) + 4 GPUs (C1060)



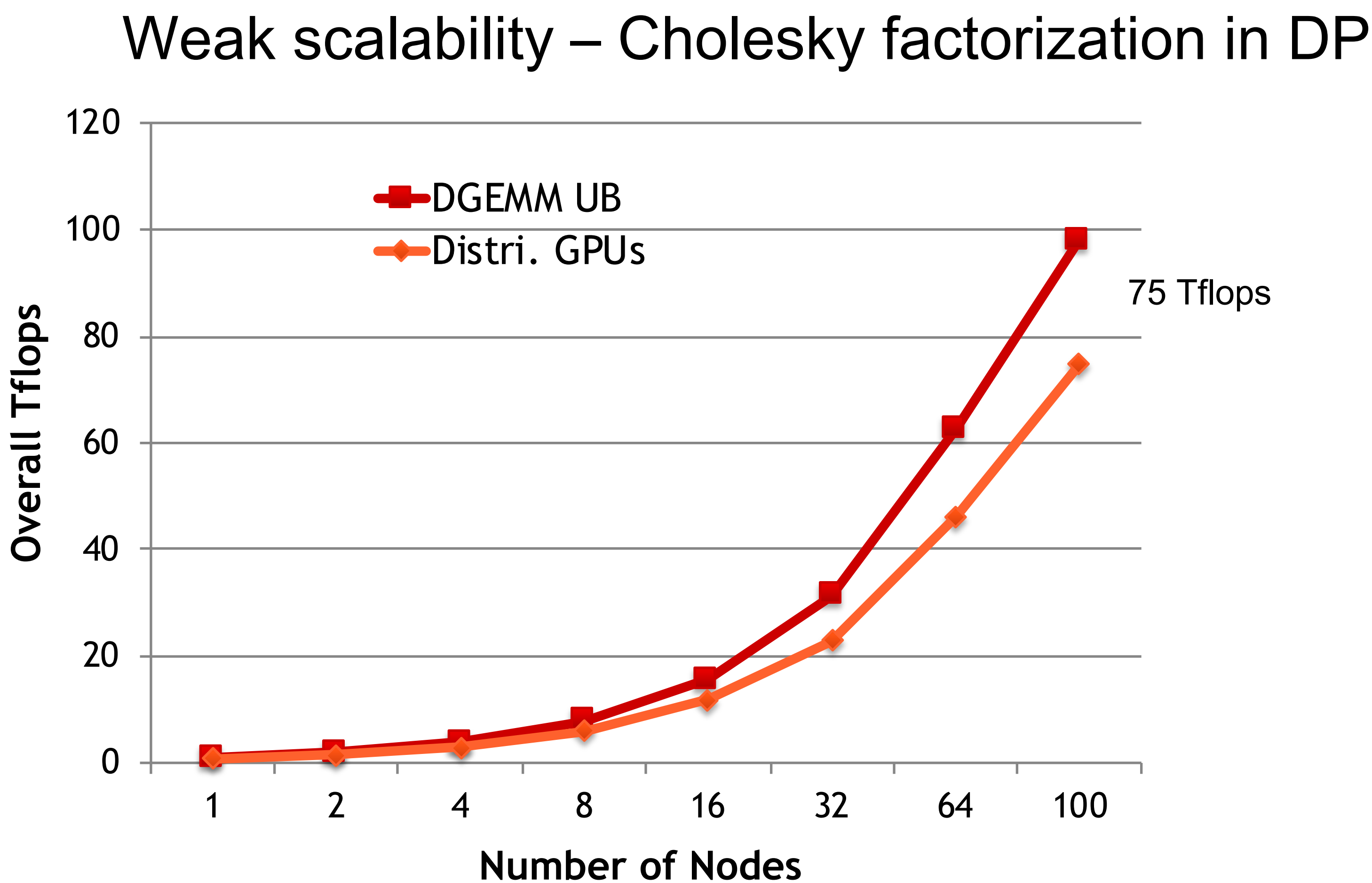
Static + Dynamic Scheduling

- Dynamic schedulers provide ease of development
- Static scheduling allows more flexible design, e.g., to minimize communication
- We have explored combination of
  - Static data distribution and communication between node
  - Dynamic scheduling within a node
- Developed are QR and Cholesky for single and distributed multicore and multiGPU nodes

w/ Fengguang Song

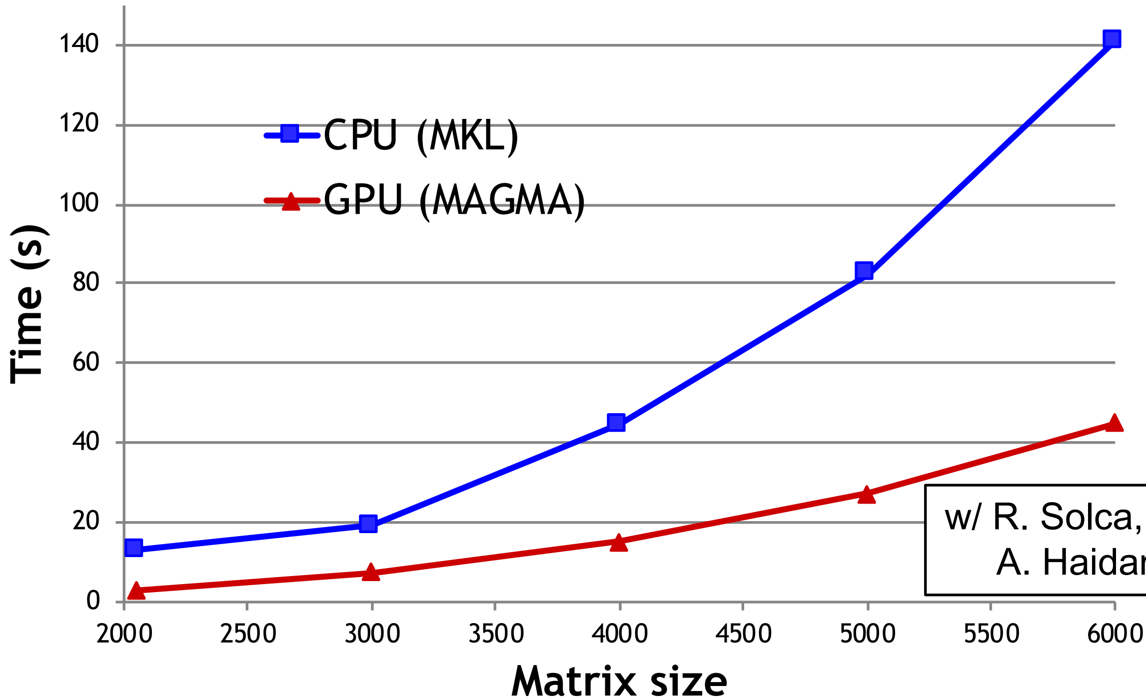
Distributed GPUs

on Keeneland nodes – 12 CPU cores and 3 Fermi GPUs



Complete Eigensolvers

Generalized Hermitian-definite eigenproblem solver (**A x = λ B x**)
[double complex arithmetic; based on Divide & Conquer; eigenvalues + eigenvectors]



w/ R. Solca, T. Schulthess, W. Sawyer, A. Haidar, C. Vomel

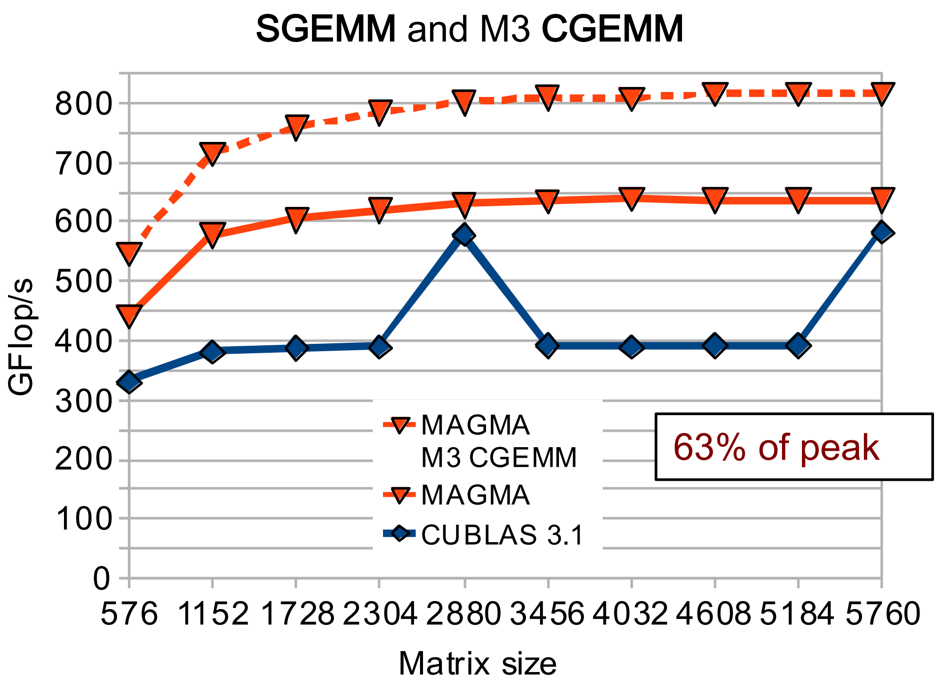
GPU Fermi C2050 [448 CUDA Cores @ 1.15 GHz]
+ Intel Q9300 [4 cores @ 2.50 GHz]

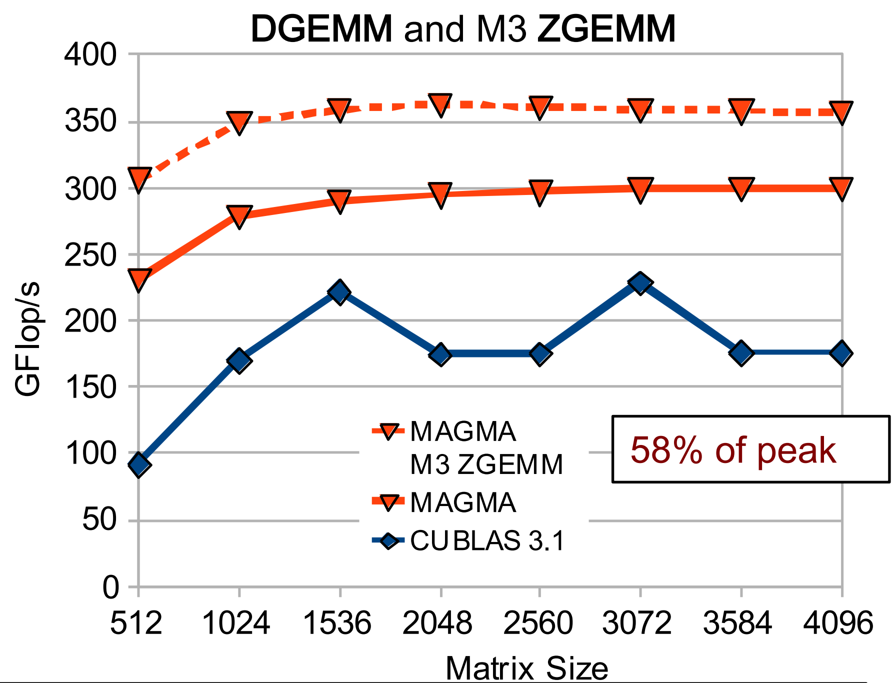
CPU AMD ISTANBUL
[8 sockets x 6 cores (48 cores) @2.8GHz]

	GPU	CPU
DP peak	**515 + 40** GFlop/s	**538** GFlop/s
System cost	~ $**3,000**	~ $**30,000**
Power *	~ **220** W	~ **1,022** W

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

MAGMA BLAS



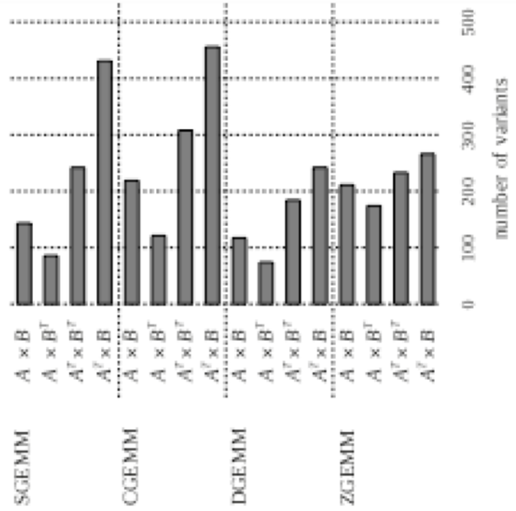


Tesla C2050 (Fermi): 448 CUDA cores @ 1.15GHz, theoretical SP peak is 1.03 Tflop/s, DP is 515 GFlop/s)

- Performance critically depend on BLAS
- On going efforts on developing highly optimized BLAS for NVIDIA GPUs in CUDA
  - CUBLAS 3.2 GEMM are based on the MAGMA kernels
  - TRSM and other Level 3 BLAS based on GEMM
  - HEMV / SYMV (accepted at SC11)
- **Auto-tuning** has become more important …

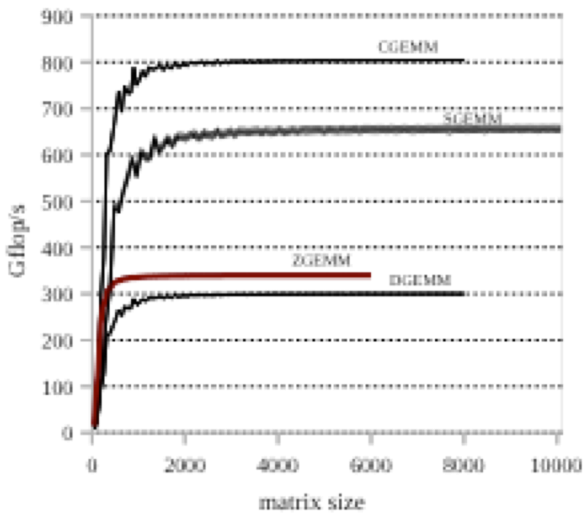
w/ R. Nath, P. Du, T. Dong

Autotuning



- **Autotuning framework**
  - will define stencils for kernels
  - Generate implementations while pruning the search space
  - Empirically find the fastest implementation

Performance of GEMM on Fermi (C2050)



An example – Autotuning GEMM on Fermi (C2050):

- 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/ **J. Kurzak**, P. Luszczek

Future directions

- **Hybrid algorithms**
  - Further expend functionality,
    including support for certain sparse LA algorithms
  - New highly parallel algorithms of optimized communication and synchronization
- **OpenCL support (to increase MAGMA's portability)**
  - To be derived from OpenCL BLAS
- **MIC support**
- **Autotuning**
  - On both high level algorithms & BLAS
- **Multi-GPU algorithms, including distributed**
  - Scheduling
  - Further expand functionality

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