The Future of Computing: Software Libraries

Stanimire Tomov and **Jack Dongarra**
Research Director
Innovative Computing Laboratory
University of Tennessee, Knoxville

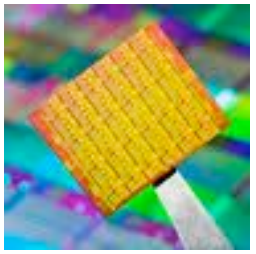
DOD CREATE Developers' Review
Savannah, Georgia
February 28, 2012

Outline

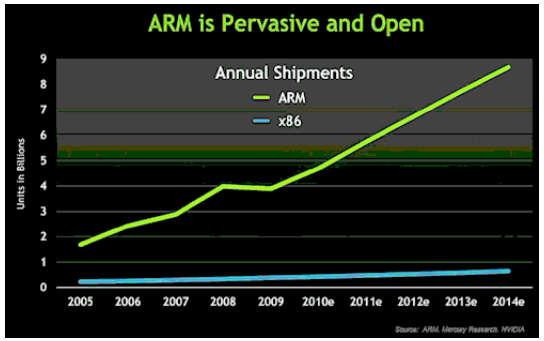
- **Motivation**
- **Challenges**
- **Current approaches**
- **Specific examples**
- **Conclusions**

Future Computer Systems

- Most likely to 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 Changes to Software

- **Must rethink the design of our algorithms and software**
  - **Manycore architectures are another disruptive technology**
    - **Similar to what happened with cluster computing and message passing**
  - **Rethink and rewrite the applications, algorithms, and software**
  - **Data movement is expensive**
  - **Flops are cheap**

Software Libraries at DoD HPCMP

[must provide support for manycore and hybrid architectures]

- **LAPACK** (including vendor optimized)
- **ScaLAPACK**
- **BLAS** (ATLAS, GotoBLAS, vendor)
- PAPI, ScaLASCA, TAU
- PETSc
- SuperLU
- …

Dense Linear Algebra

[more at http://www.ors.hpc.mil/software/]

The Need for HP Linear Algebra

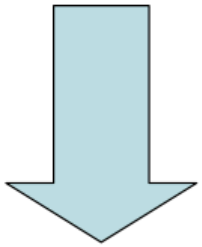
Electronic structure calculations

- 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

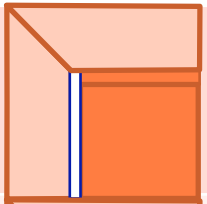
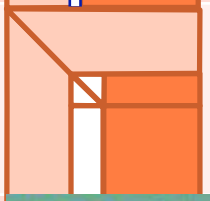
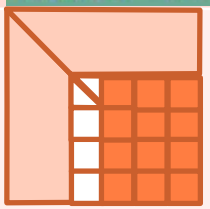
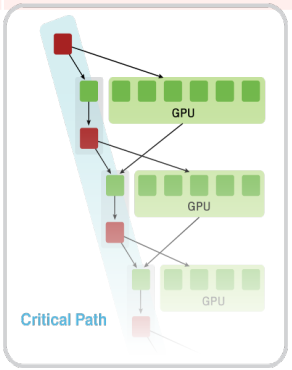
· ρ is the density (of the original many-body system)

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

where V_{XC} depends only on ρ

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

A Next Generation of DLA 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

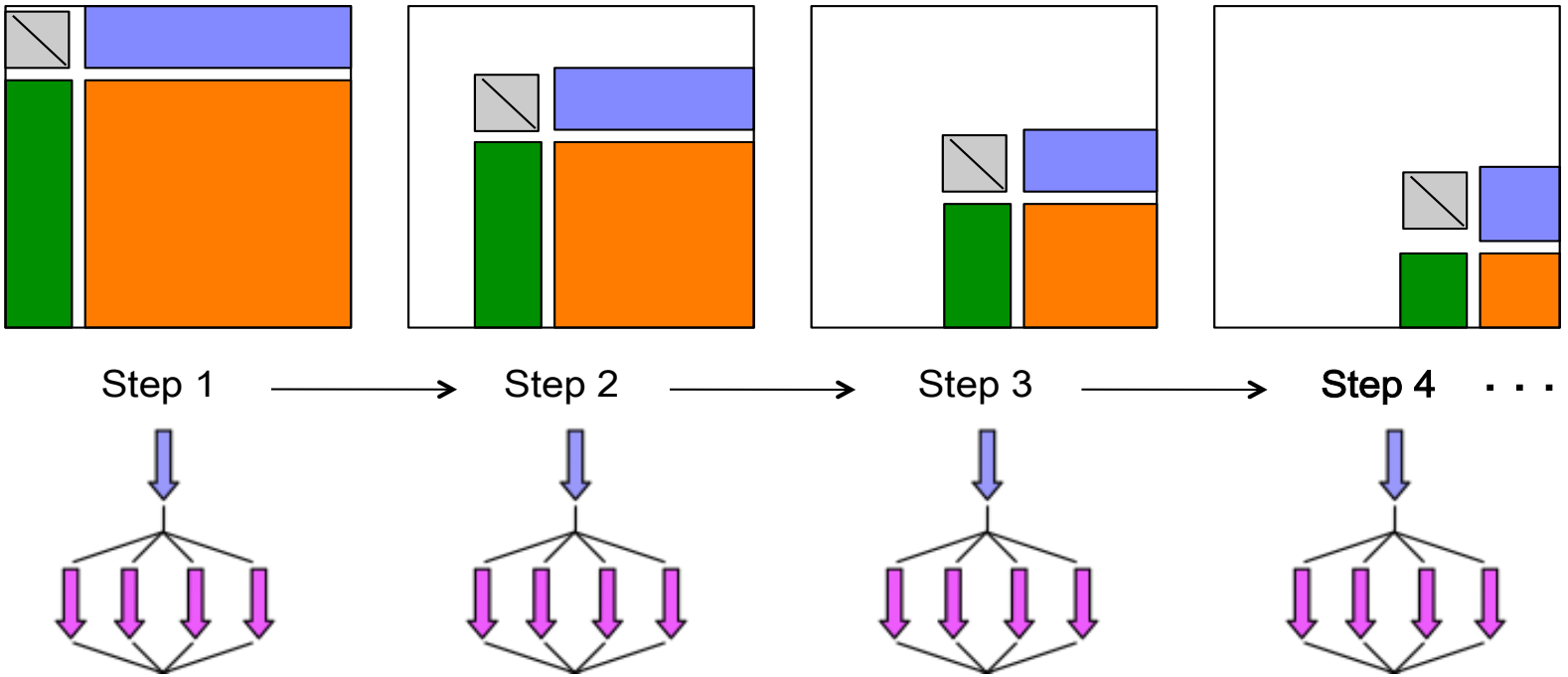
Challenges for Software Libraries

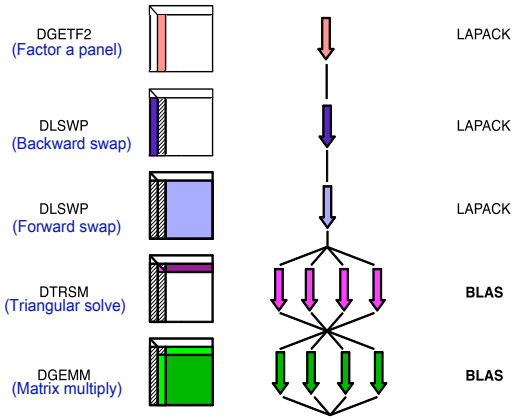
1. **Synchronization**
   - Break Fork-Join model
2. **Communication**
   - Use methods which have lower bound on communication
3. **Mixed precision methods**
   - 2x speed of ops and 2x speed for data movement
4. **Autotuning**
   - Today's machines are too complicated, build "smarts" into software to adapt to the hardware
5. **Fault resilient algorithms**
   - Implement algorithms that can recover from failures/bit flips
6. **Reproducibility of results**

Real Crisis with HPC is with the Software

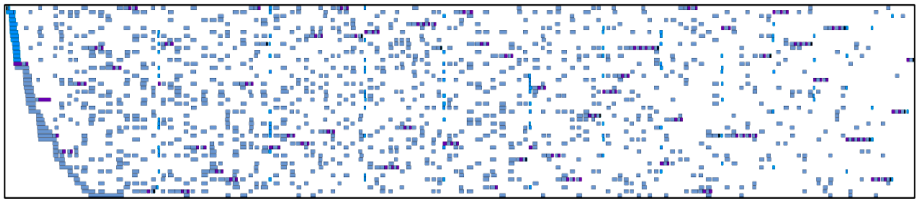
- **Our ability to configure the next hardware system is without question just a matter of time and $$**
- **A supercomputer application and software are usually much more long-lived than a hardware**
  - **Hardware life typically five years at most…. Apps 20-30 years**
  - **Fortran and C are the main programming models (still!!)**
- **The REAL CHALLENGE is Software**
  - **Programming hasn't changed since the 70's**
  - **HUGE manpower investment**
    - **MPI… is that all there is?**
  - **Often requires HERO programming**
  - **Investments in the entire software stack is required (OS, libs, etc.)**
- **Software is a major cost component of modern technologies**
  - **The tradition in HPC system procurement is to assume that the software is free… SOFTWARE COSTS (over and over)**

1. Synchronization (in LAPACK LU)





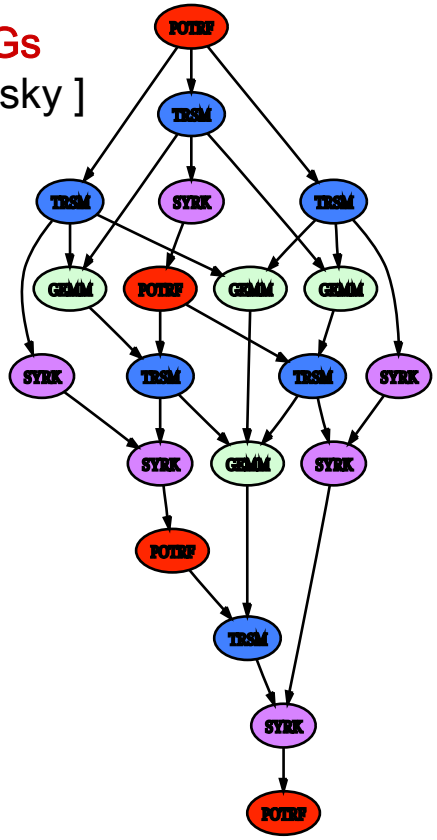
- fork join
- bulk synchronous processing



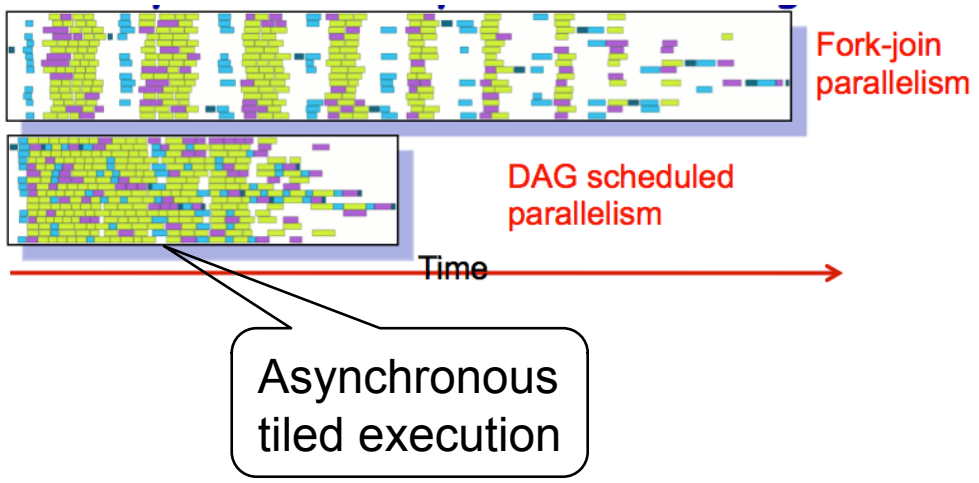
Synchronization-avoiding (PLASMA)

- **Idea:** break into smaller tasks and remove dependencies
- **Objectives:** high utilization of each core, scaling to large number of cores
- **Methodology:** Arbitrary DAG scheduling, Fine granularity / block data layout

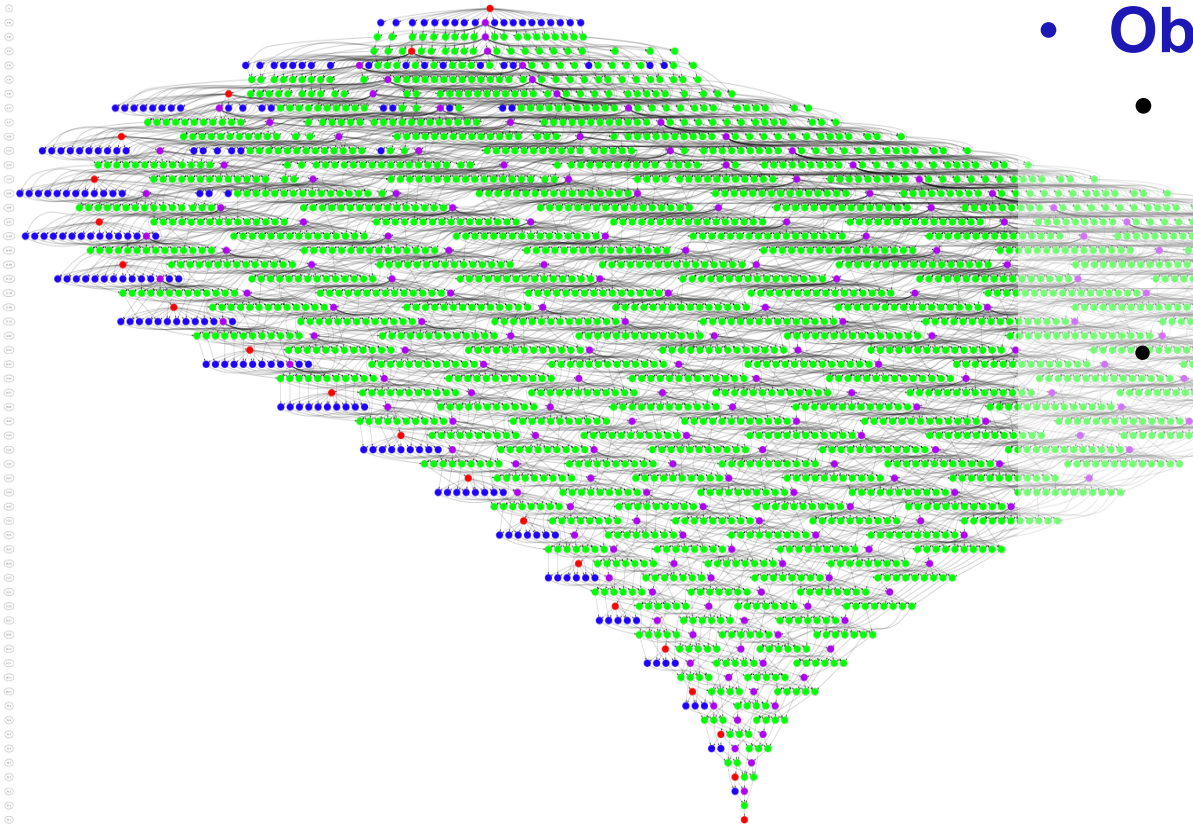
Algorithms as DAGs
[example – Cholesky]



Execution trace example
[reduces idle time]



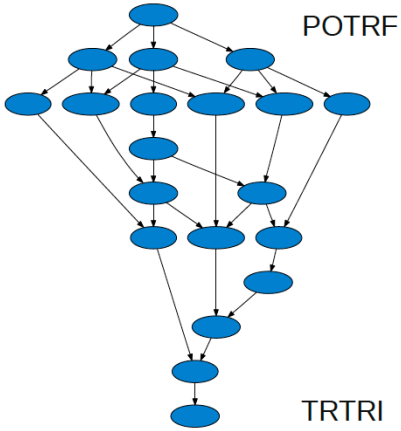
Algorithms as DAGs

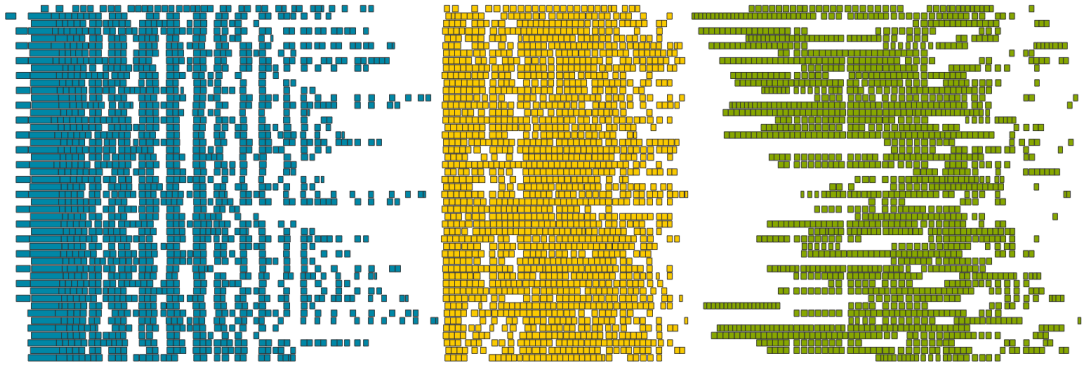


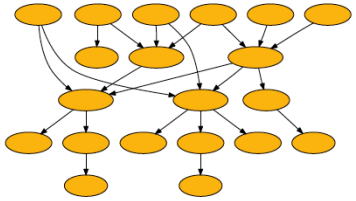
[example – a Cholesky factorization DAG]

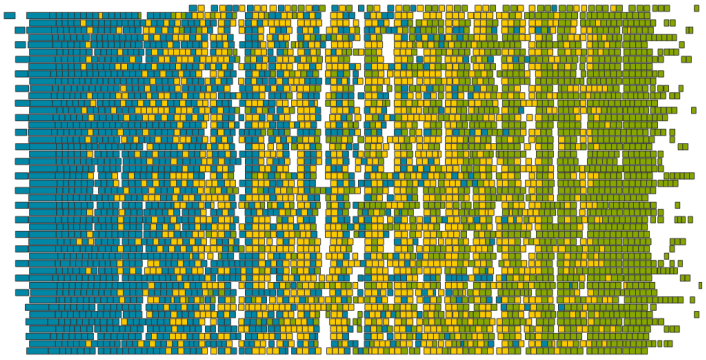
- **Observations**
  - DAG too large to be generated ahead of time
    - Generate it dynamically
  - HPC is about distributed heterogeneous resources
    - Have to get involved in message passing
    - Distributed management of the scheduling
    - Dynamically deal with heterogeneity

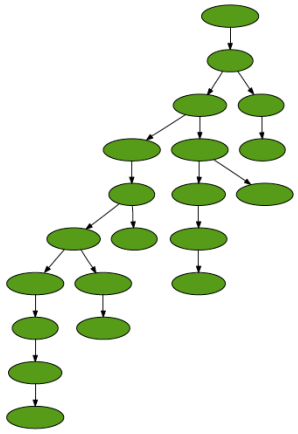
Scheduling Algorithms as DAGs



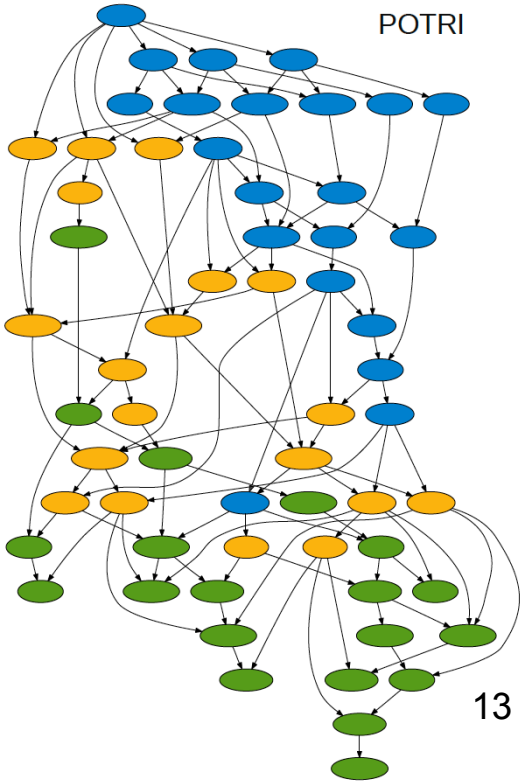








48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,



2. Communication

- **Exponentially growing gaps with time**

Annual improvements			
Time per flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

- **A comparison**

- FPS-164 and VAX (1976)
  - 11 Mflop/s; transfer rate 44 MB/s
  - Ratio of flops to bytes of data movement: 1 flop per 4 bytes transferred
- Nvidia Fermi and PCI-X to host
  - 500 Gflop/s; transfer rate 8 GB/s
  - Ratio of flops to bytes of data movement: 62 flops per 1 byte transferred

- **Flops are cheap**
  Need algorithms of reduced communication

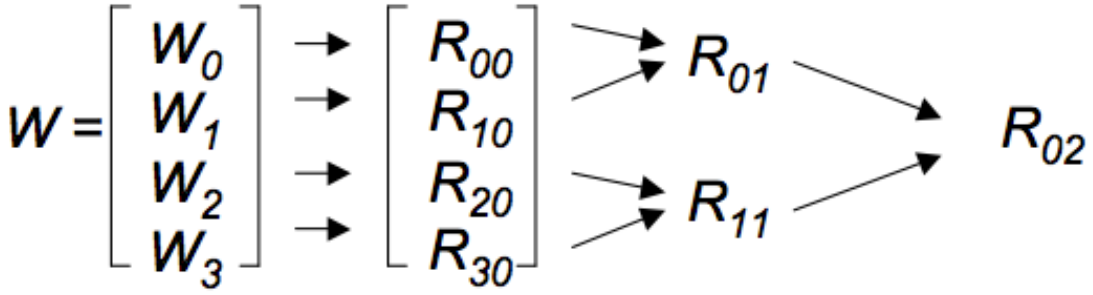
Ways to reduce communication

- **New algorithms**
  - To attain lower bounds on communication
  - Attain large speedups in theory and practice
- **Blocking for data reuse**
  - Split computation in tasks of small enough memory footprint to allow cache reuse (all levels of memory hierarchy)
- **Delayed update**
  - Accumulate inefficient transformations (e.g., Level 2 BLAS) into more efficient (e.g., Level 3 BLAS)
- **Mixed precision techniques**
  - E.g., mixed-precision for sparse iterative solvers

An Example

TSQR: QR factorization of a tall skinny matrix using Householder transformations

- QR decomposition of m x b matrix W, m >> b, on P processors
- Usual parallel algorithm (ScaLAPACK)
  - Compute Householder vector for each column
  - Number of massages ~ b log P
- Communication avoiding algorithm
  - Reduction operation, with QR as an operator
  - Number of massages ~ log P

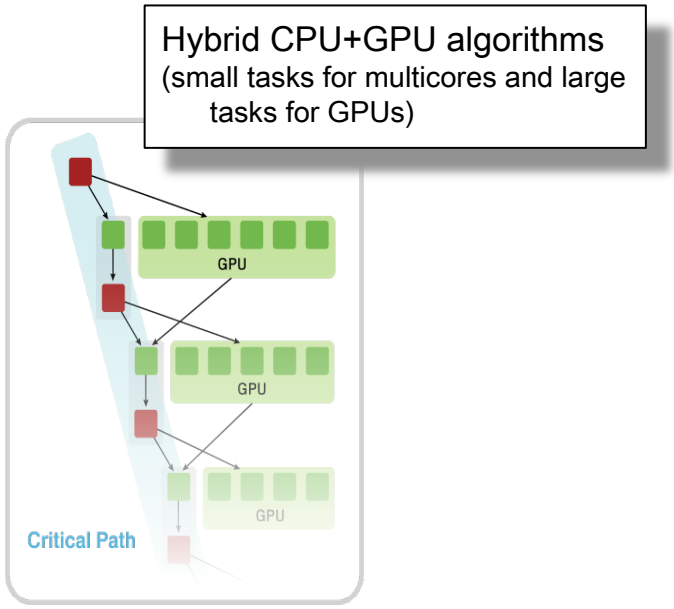


J. Demmel, L. Grigori, M. Hoemmen, J. Langou '08

Hybrid Algorithms (Challenges 1 & 2)

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



Hybrid Algorithms

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"**

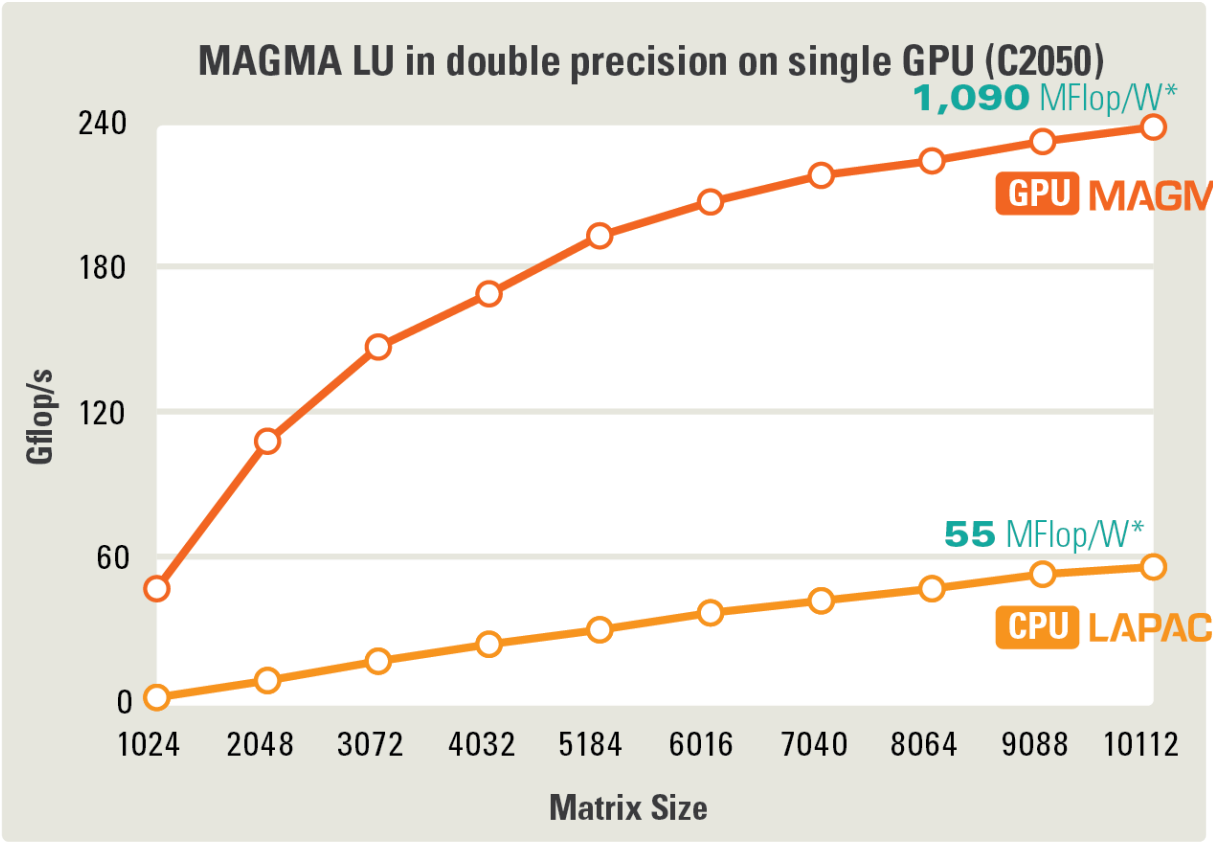
A hybrid algorithm example

- **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)

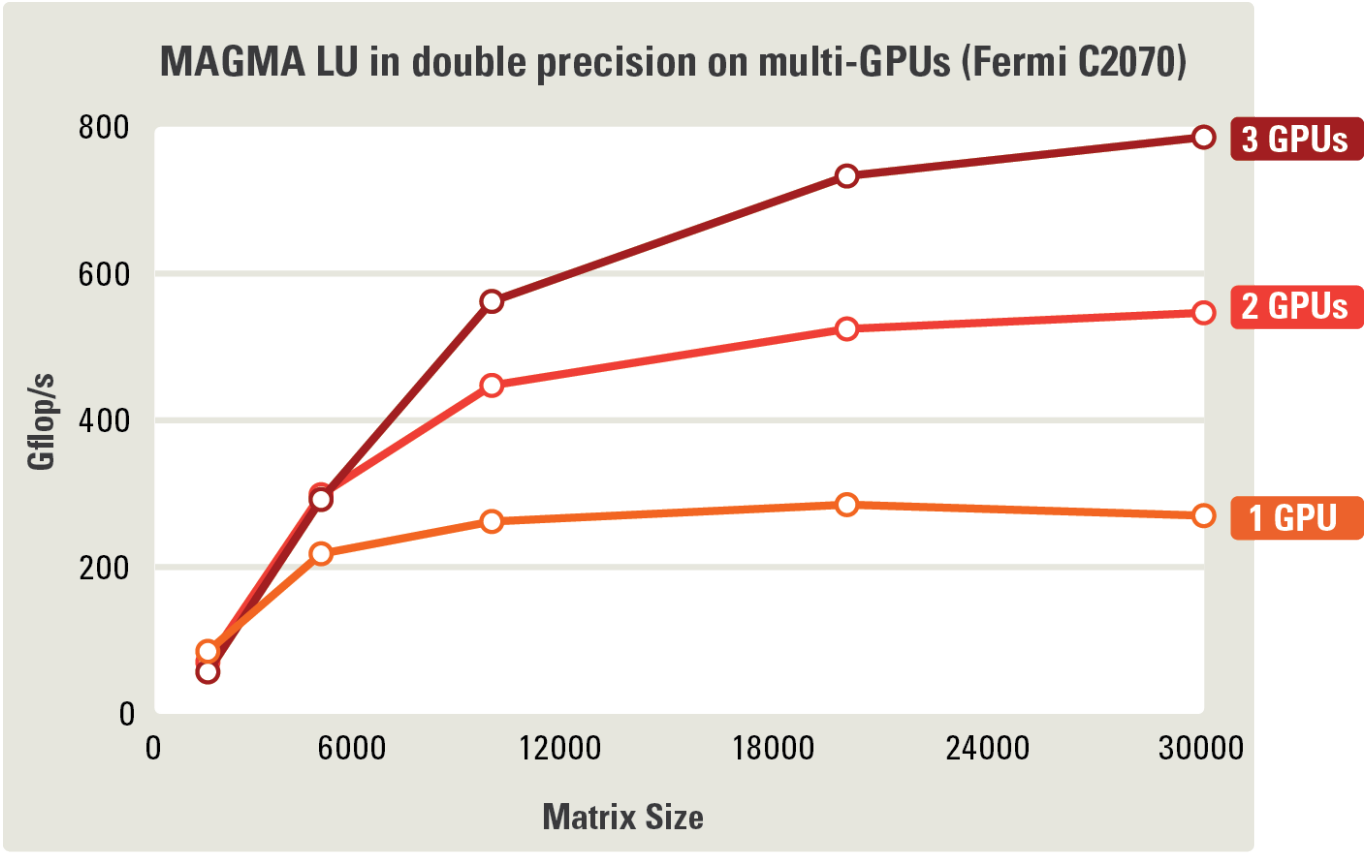


GPU Fermi C2050 (448 CUDA Cores @ 1.15 GHz)
+ Intel Q9300 (4 cores @ 2.50 GHz)
DP peak **515** + **40** GFlop/s
Power * ~**220** W

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)

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, …)
```

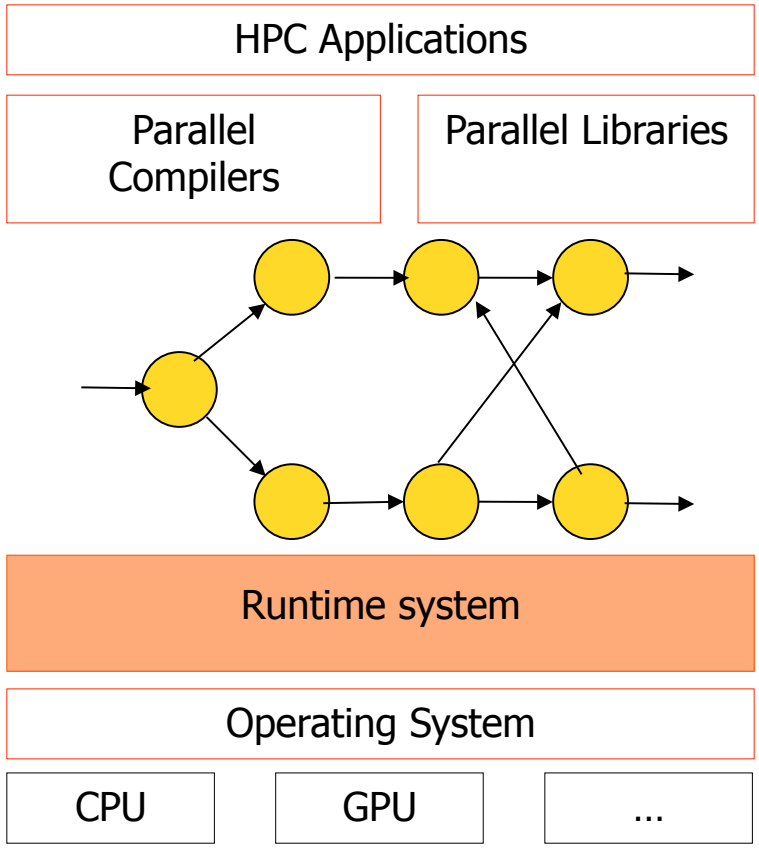
- **Tile kernels and one-sided factorizations and solvers (using StarPU) are released in MAGMA 1.1**

The StarPU runtime system

The need for runtime systems

- **do dynamically what would be difficult to do statically**
- **Library that provides**
  - **Task scheduling**
  - **Memory management**

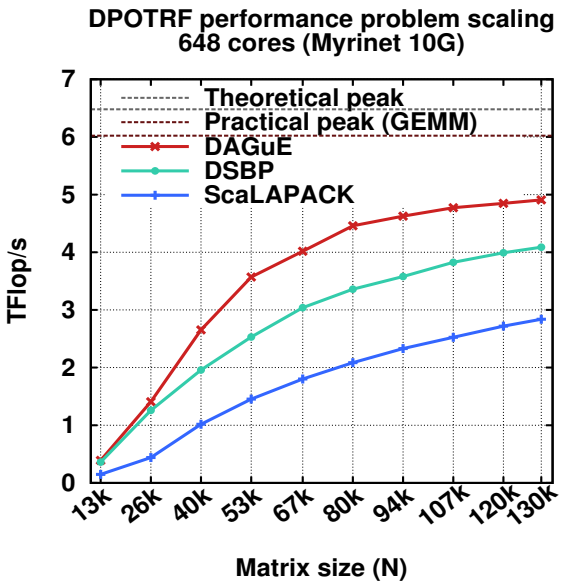
http://runtime.bordeaux.inria.fr/StarPU/



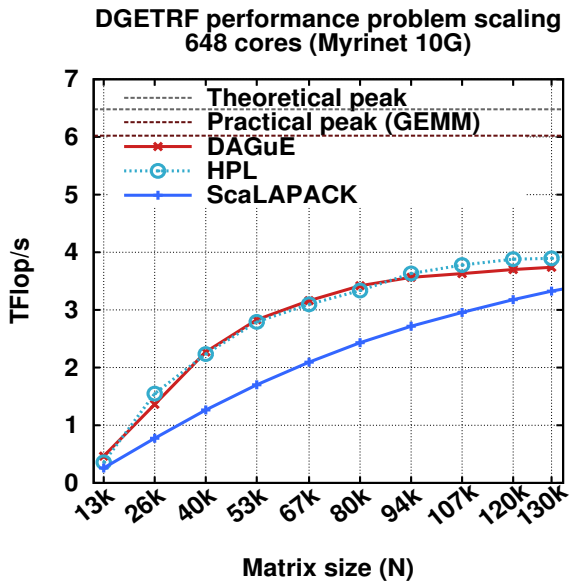
The DAGuE runtime system

- **Distribute the DAG analysis**
  - **The DAG is never completely unrolled**
  - **Each node only unrolls it's own portion of the DAG**
- **Minimize the data transfers**
- **Overlap communication and computations**
- **Let the user describe the algorithms based on data dependencies between tasks**

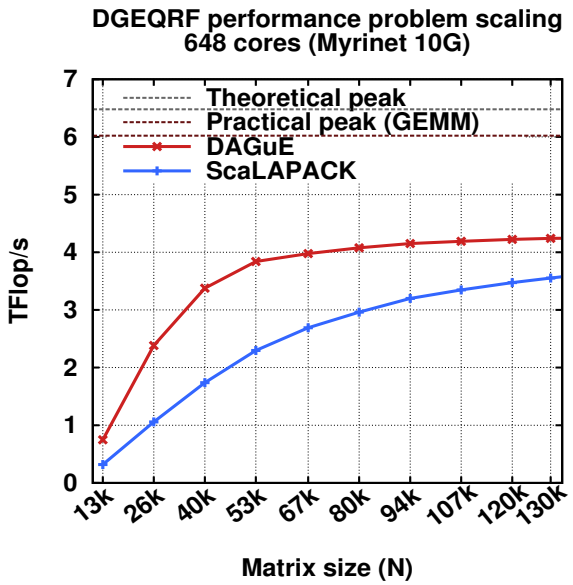
Performance



(a) Cholesky Factorization



(b) LU Factorization



(c) QR Factorization

Hardware: 81 dual socket Intel Xeon L5420 quad core nodes @2.5 GHz => 648 cores

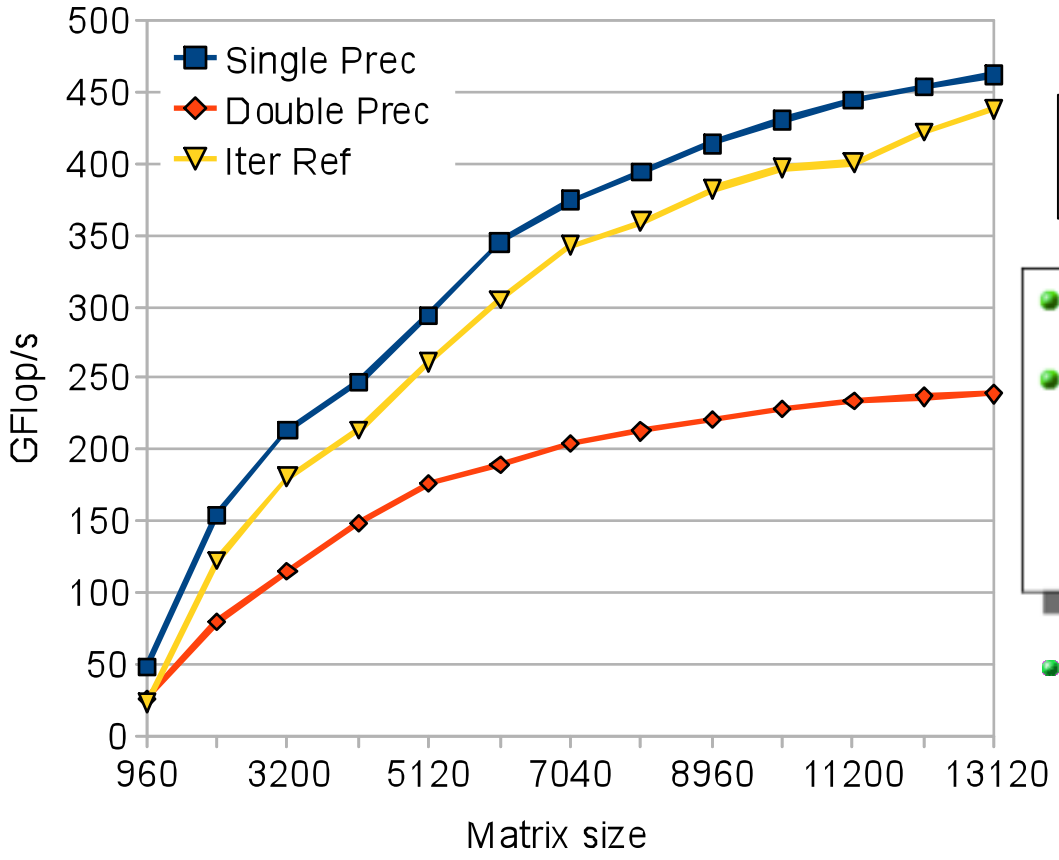
DAGuE & PLASMA teams @ ICL; For more information, see http://icl.cs.utk.edu/dague/;

3. 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 [Δx rather than x].**

Mixed Precision Solvers

MAGMA LU-based solvers on Fermi (C2050)



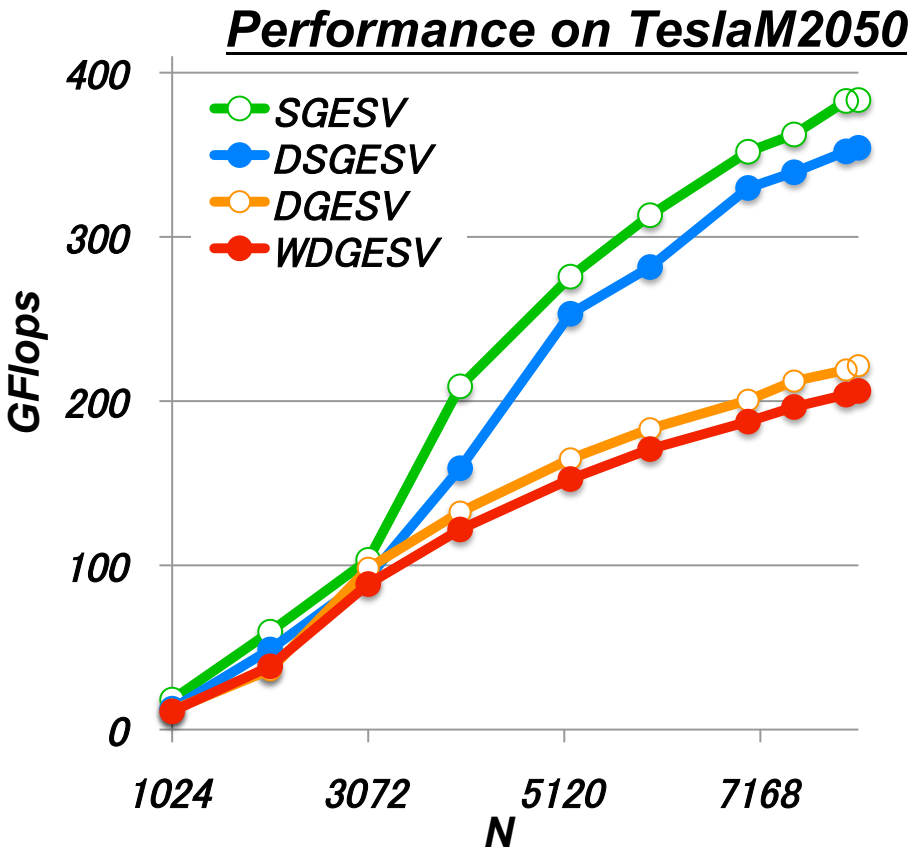
FERMI Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 GFlop/s

- **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

Mixed-precision solvers



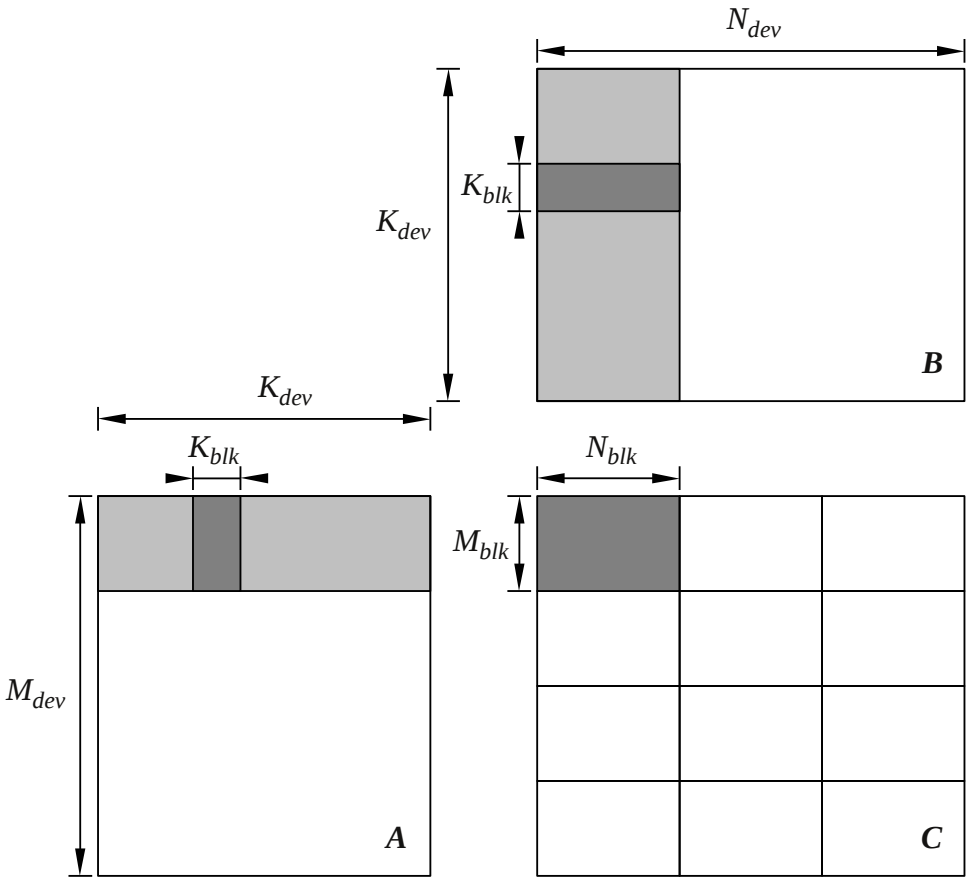
- **High-precision (quadruple): Double-double precision (WP)**
- **WP flops are expensive! 1 WP flop ~ 20 DP flops**
- **Up to 20x speedup over direct WP solver**

w/ Daichi Mukunoki
University of Tsukuba, Japan

- **Host: Xeon E5630 2.53GHz (4cores*2sockets), DDR3 6GB**
- **CentOS6.0, CUDA4.0**

4. Autotuning

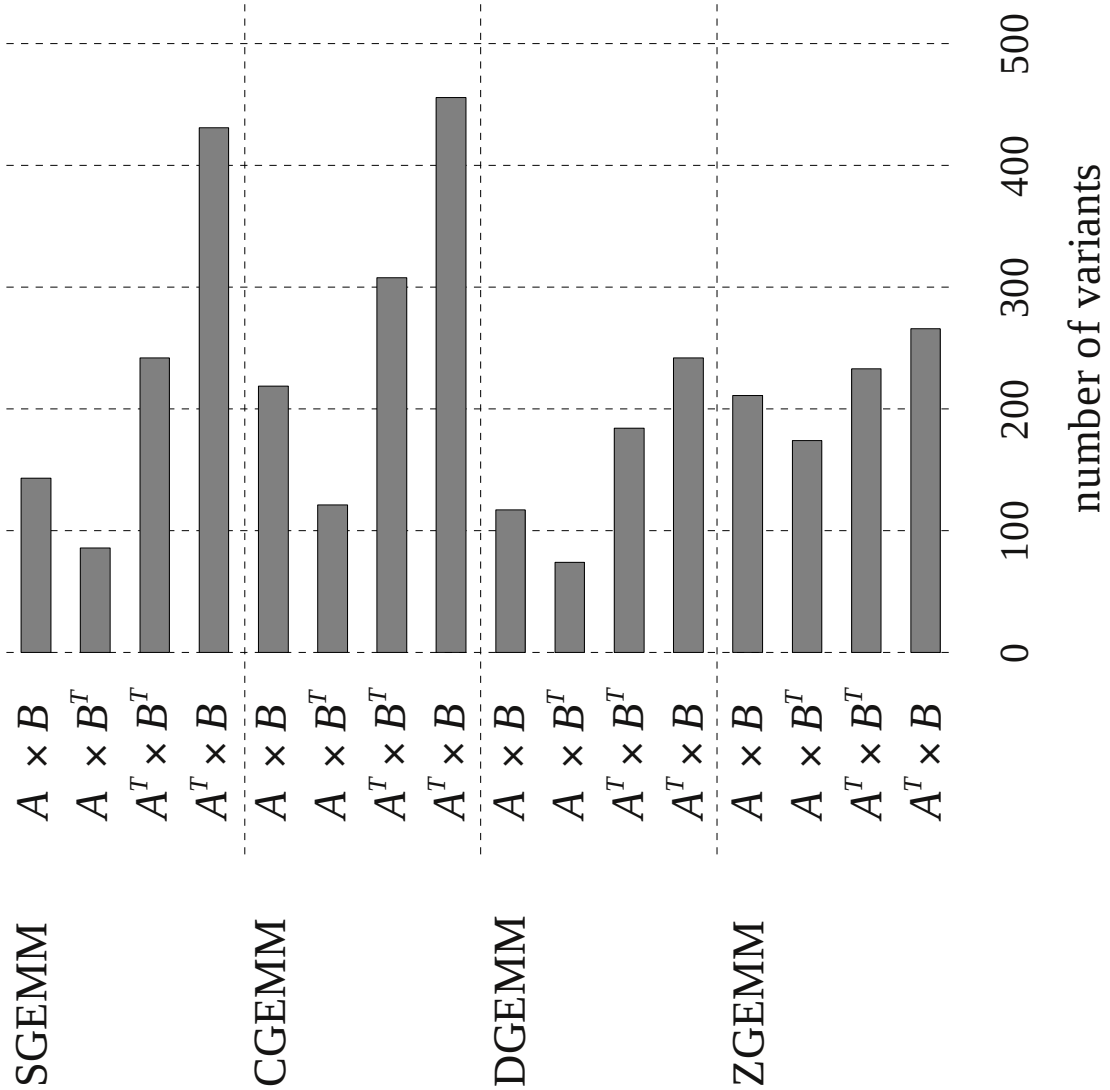
$$C = \alpha A B + \beta C$$



- **To empirically find best implementations**
- **Parameters influencing performance are selected**
- **Code is parameterized**
- **Search engine automatically finds best version**

Left figure: Example parameterization of matrix-matrix multiplication for NVIDIA GPUs

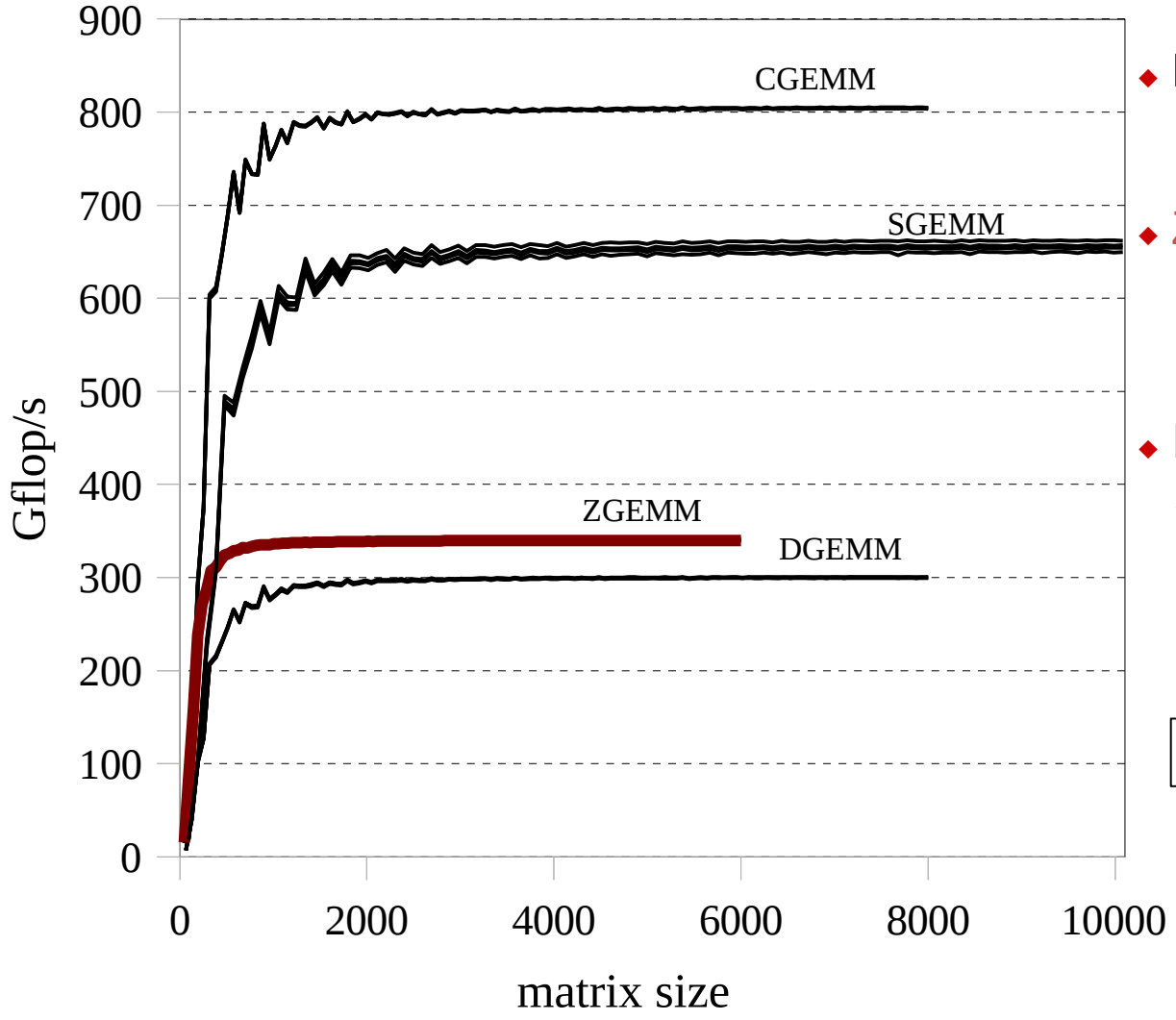
Autotuning in MAGMA 1.1



- 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

Conclusions

- **For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware**
- **This strategy needs to be rebalanced - barriers to progress are increasingly on the software side**
- **High Performance Ecosystem out of balance**
  - **Hardware, OS, Compilers, Software, Algorithms, Applications**
    - **No Moore's Law for software, algorithms and applications**

Collaborators / Support

- **MAGMA team**
  http://icl.cs.utk.edu/magma/
- **PLASMA team**
  http://icl.cs.utk.edu/plasma
- **DAGuE team**
  http://icl.cs.utk.edu/dague/
- **Collaborating partners**

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

  INRIA, France
  KAUST, Saudi Arabia