Linear Algebra Software On Accelerated Multicore Systems

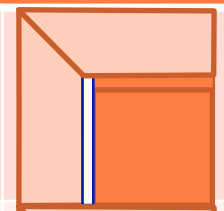
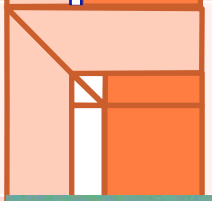
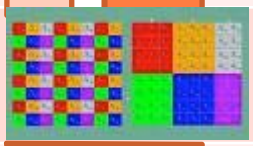
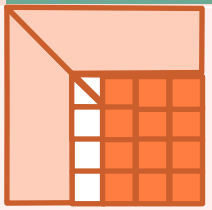
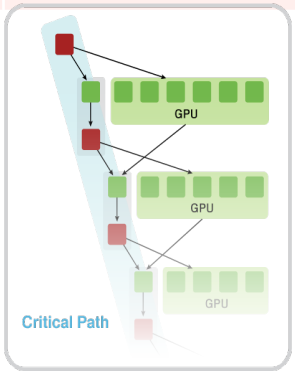
Piotr Luszczek
Jack Dongarra
Mark Gates
Tim Dong
Ichitaro Yamazaki
Simplice Donfack
Stanimire Tomov
Azzam Haidar
Hartwig Anzt
University of Tennessee, Knoxville

PLC Workshop 2014
Phoenix, AZ
May 19, 2014

Outline

- Methodology
- Dense linear system and eigen-problem solvers
- Multi-GPU algorithms
  - Dynamic scheduling
  - Distributed MAGMA
- MAGMA Sparse
- Future Directions

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

MAGMA: LAPACK for GPUs

- MAGMA
  - Matrix algebra for GPU and multicore architecture
  - The LAPACK/ScaLAPACK on hybrid architectures
  - http://icl.cs.utk.edu/magma/
- MAGMA 1.4.1
  - For NVIDIA CUDA GPUs on shared memory systems
  - Hybrid dense linear algebra (for CPUs and GPUs)
    - One-sided factorizations and linear system solvers
    - Two-sided factorizations and eigenproblem solvers
    - A subset of BLAS and auxiliary routines in CUDA
- MAGMA developers & collaborators
  - UTK, UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  - Community effort, similarly to LAPACK/ScaLAPACK

Key Features of MAGMA 1.4.1

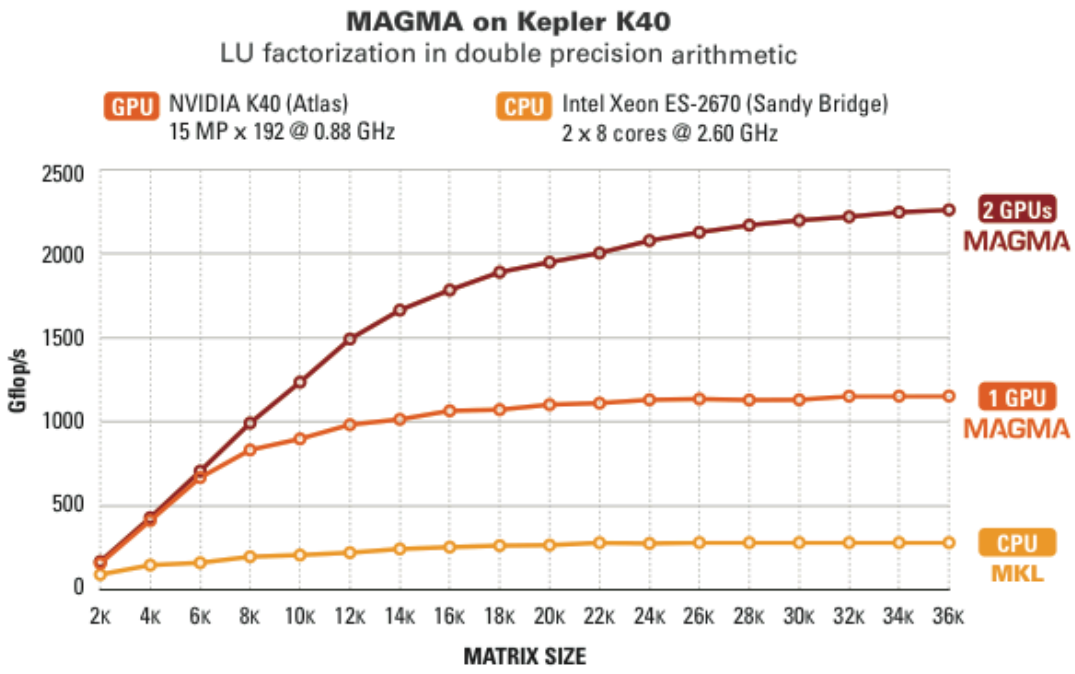
- High performance
- Multiple precision support
  (32-bit/64-bit, real/complex, and mixed)
- Hybrid algorithms
- Out-of-GPU memory algorithms
- MultiGPU support

Key Features of MAGMA 1.4.1

HYBRID ALGORITHMS

MAGMA uses a hybridization methodology where algorithms of interest are split into tasks of varying granularity and their execution scheduled over the available hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level 3 BLAS, are scheduled on the GPU.

PERFORMANCE

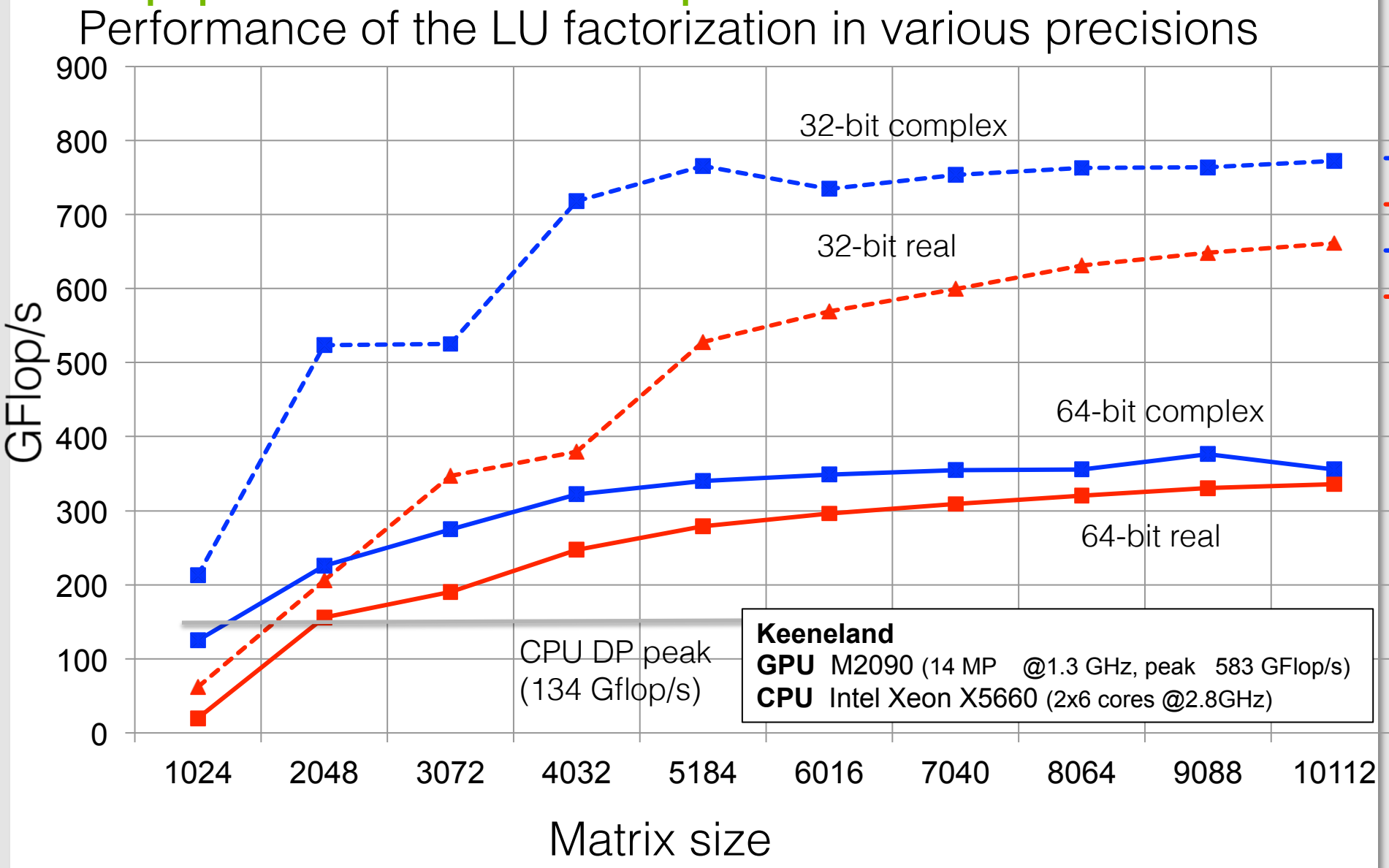


FEATURES AND SUPPORT

- MAGMA 1.4.1 FOR CUDA
- clMAGMA 1.1 FOR OpenCL
- MAGMA MIC 1.1 FOR Intel Xeon Phi

CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●	●	Eigenvalue problem solvers
●	●		Auxiliary BLAS
●	●	●	CPU Interface
●	●	●	GPU Interface
●	●	●	Multiple precision support
●			Non-GPU-resident factorizations
●	●	●	Multicore and multi-GPU support
●	●	●	LAPACK testing
●	●	●	Linux
●			Windows
●			Mac OS

Support for Multiple Precisions

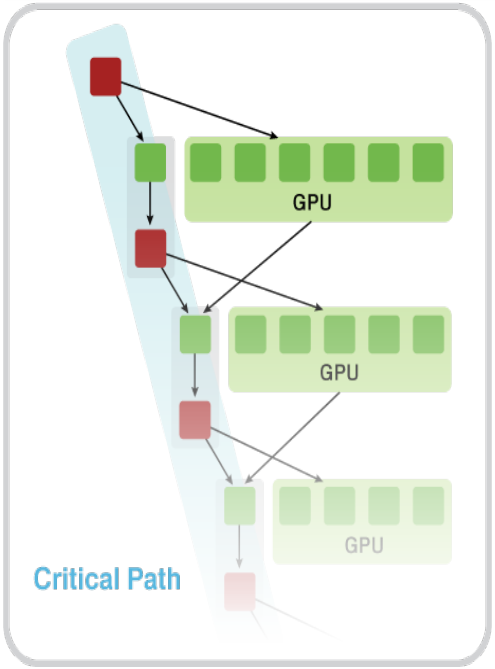


Methodology Overview

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

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



- Successfully applied to fundamental linear algebra algorithms
  - One- and two-sided factorizations and solvers
  - Iterative linear- and eigensolvers

- Productivity
  - 1) High level; 2) Leveraging prior developments; 3) Exceeding in performance homogeneous solutions

Example of Hybrid Algorithm

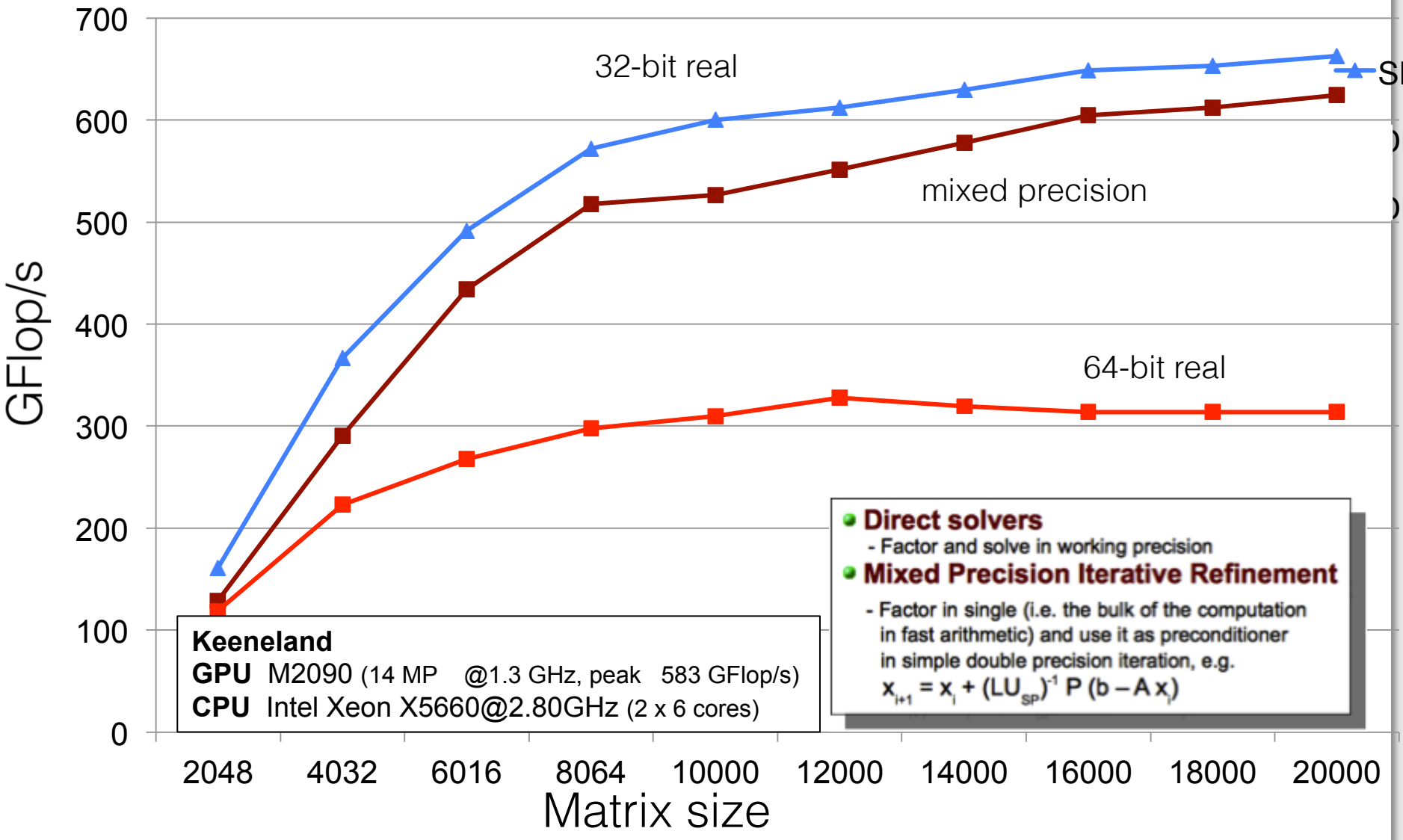
- Left-looking hybrid Cholesky factorization in MAGMA

```
1   for ( j=0; j<n; j += nb)  {
2         jb = min(nb, n – j);
3         magma_zherk( MagmaUpper, MagmaConjTrans,
                       jb, j, m_one, dA(0, j), ldda, one, dA(j, j), ldda, queue );
4         magma_zgetmatrix_async( jb, jb, dA(j,j), ldda, work, 0, jb, queue, &event );
5         if ( j+jb < n )
6             magma_zgemm( MagmaConjTrans, MagmaNoTrans, jb, n-j-jb, j, mz_one,
                           dA(0, j ), ldda, dA(0, j+jb), ldda, z_one,  dA(j, j+jb), ldda, queue );
7         magma_event_sync( event );
8         lapackf77_zpotrf( MagmaUpperStr, &jb, work, &jb, info );
9         if ( *info != 0 )
10            *info += j;
11        magma_zsetmatrix_async( jb, jb, work, 0, jb, dA(j,j), ldda, queue, &event );
12        if ( j+jb < n ) {
13            magma_event_sync( event );
14            magma_ztrsm( MagmaLeft, MagmaUpper, MagmaConjTrans, MagmaNonUnit,
                           jb, n-j-jb, z_one, dA(j, j), ldda, dA(j, j+jb), ldda, queue );
      }
  }
```

- The difference with LAPACK – the 4 additional lines in red
- Line 8 (done on CPU) is overlapped with the work on the GPU (from line 6)

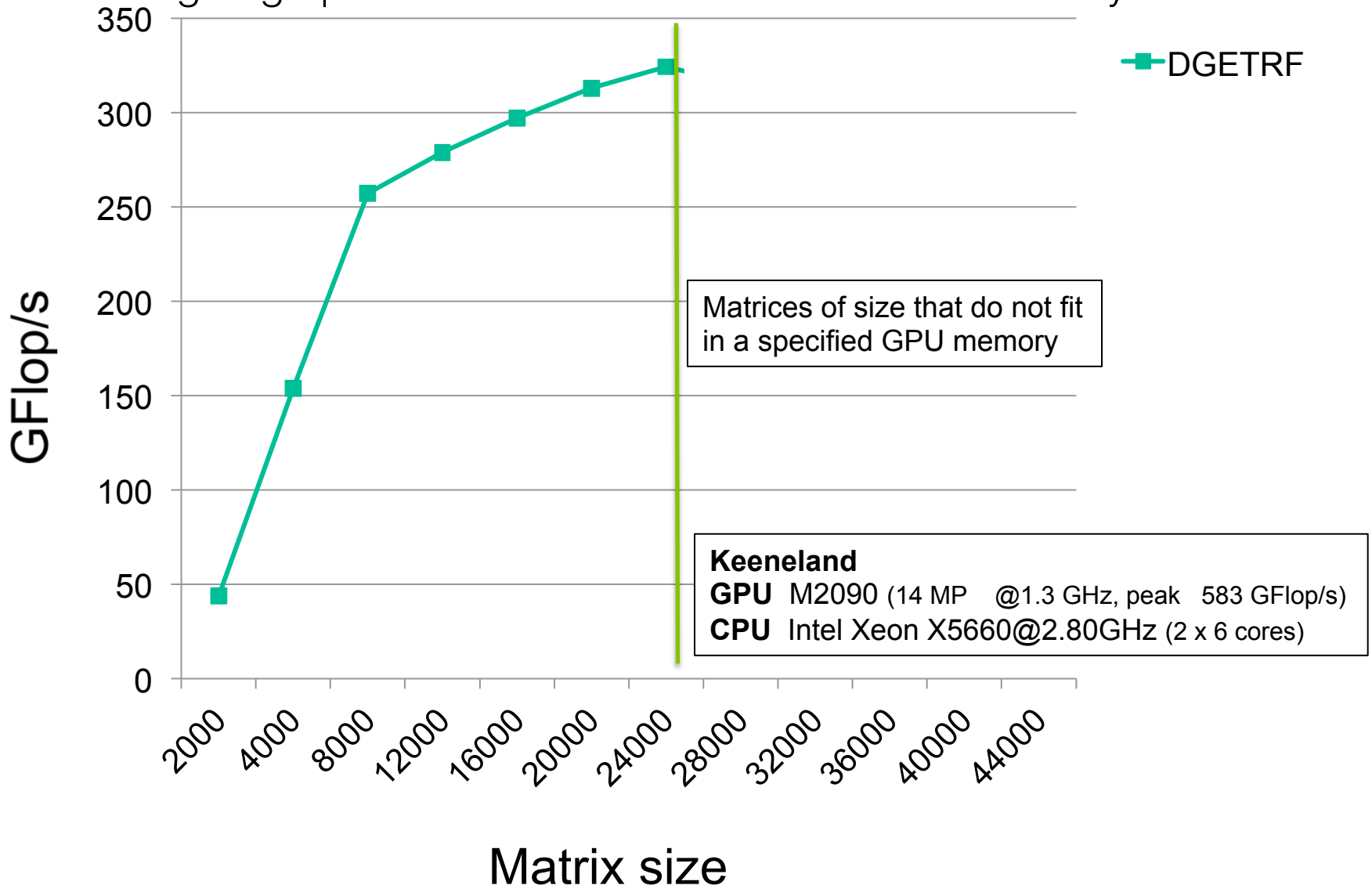
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



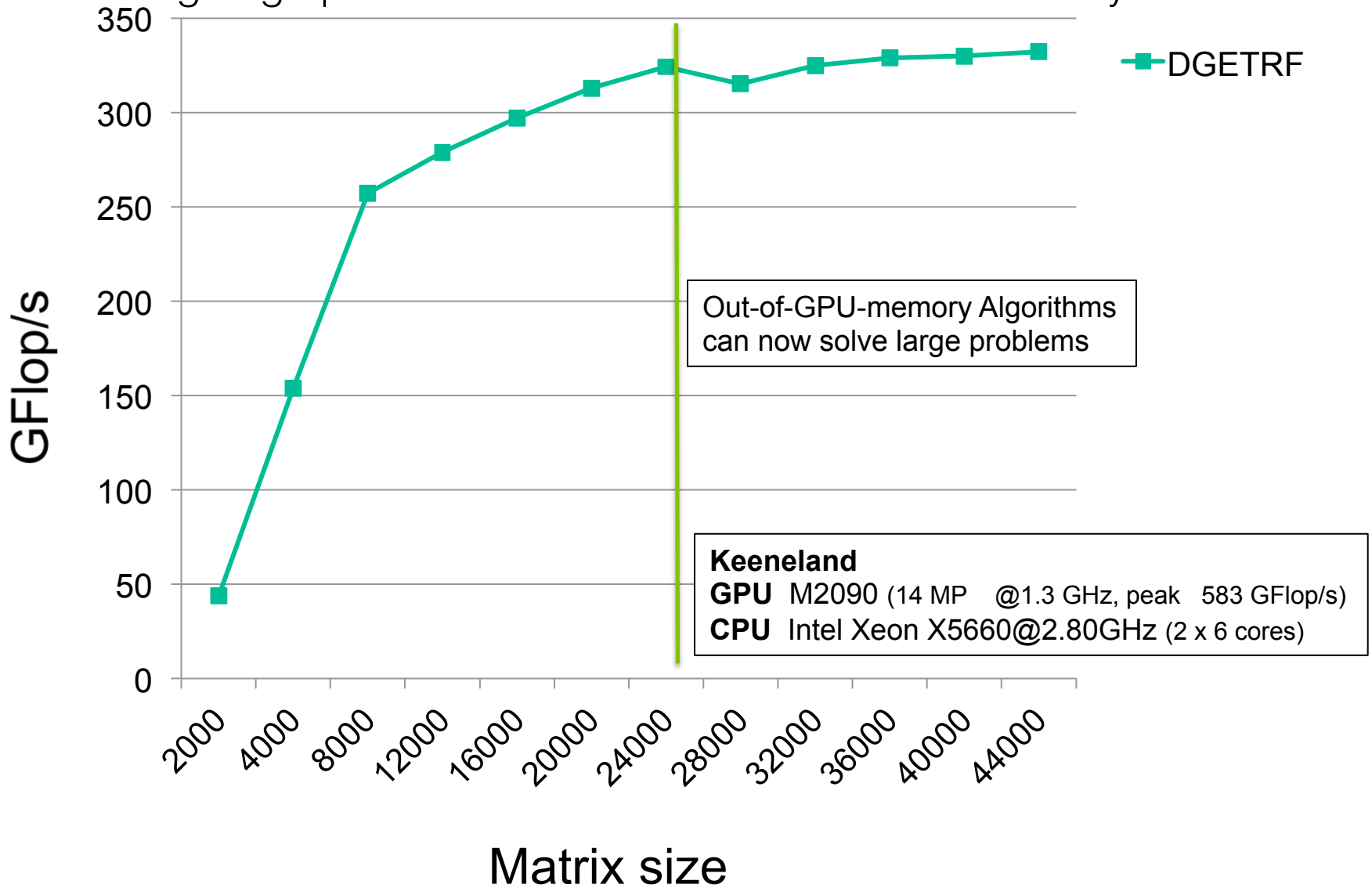
Out of GPU Memory Algorithms

Solving large problems that do not fit in the GPU memory



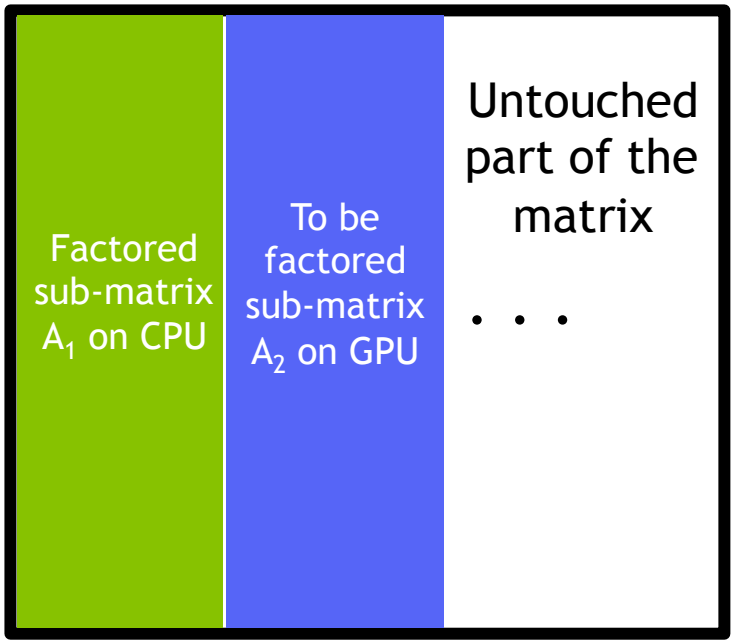
Out of GPU Memory Algorithms

Solving large problems that do not fit in the GPU memory



Out of GPU Memory Algorithm

- Perform left-looking factorizations on sub-matrices that fit in the GPU memory (using existing algorithms)
- The rest of the matrix stays on the CPU
- Left-looking versions minimize writing on the CPU



1) Copy A_2 to the GPU
2) Update A_2 using A_1
   a) 1st panel of A_1
   b) 2nd panel of A_1 …
3) Factor the updated A_2 using existing hybrid code
4) Copy factored A_2 to the CPU

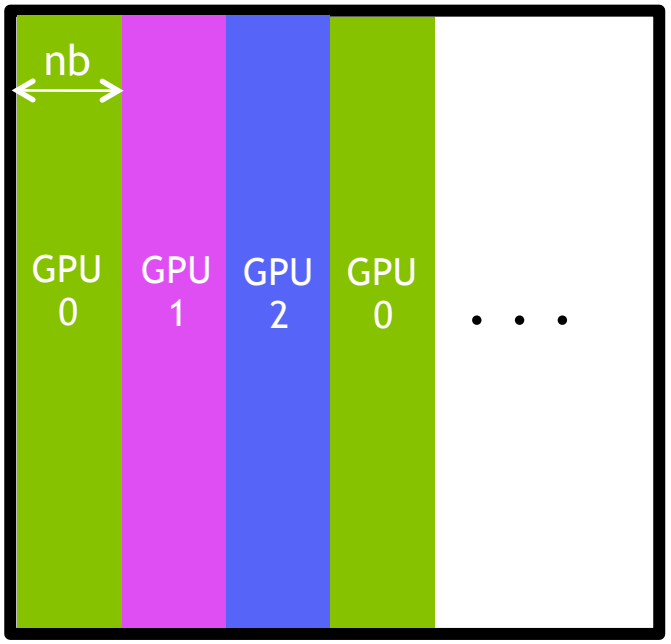
Trivially extended to multiple GPUs:

- A_2 is "larger" – spans all GPUs
- 1-D block cyclic distribution for load balance,

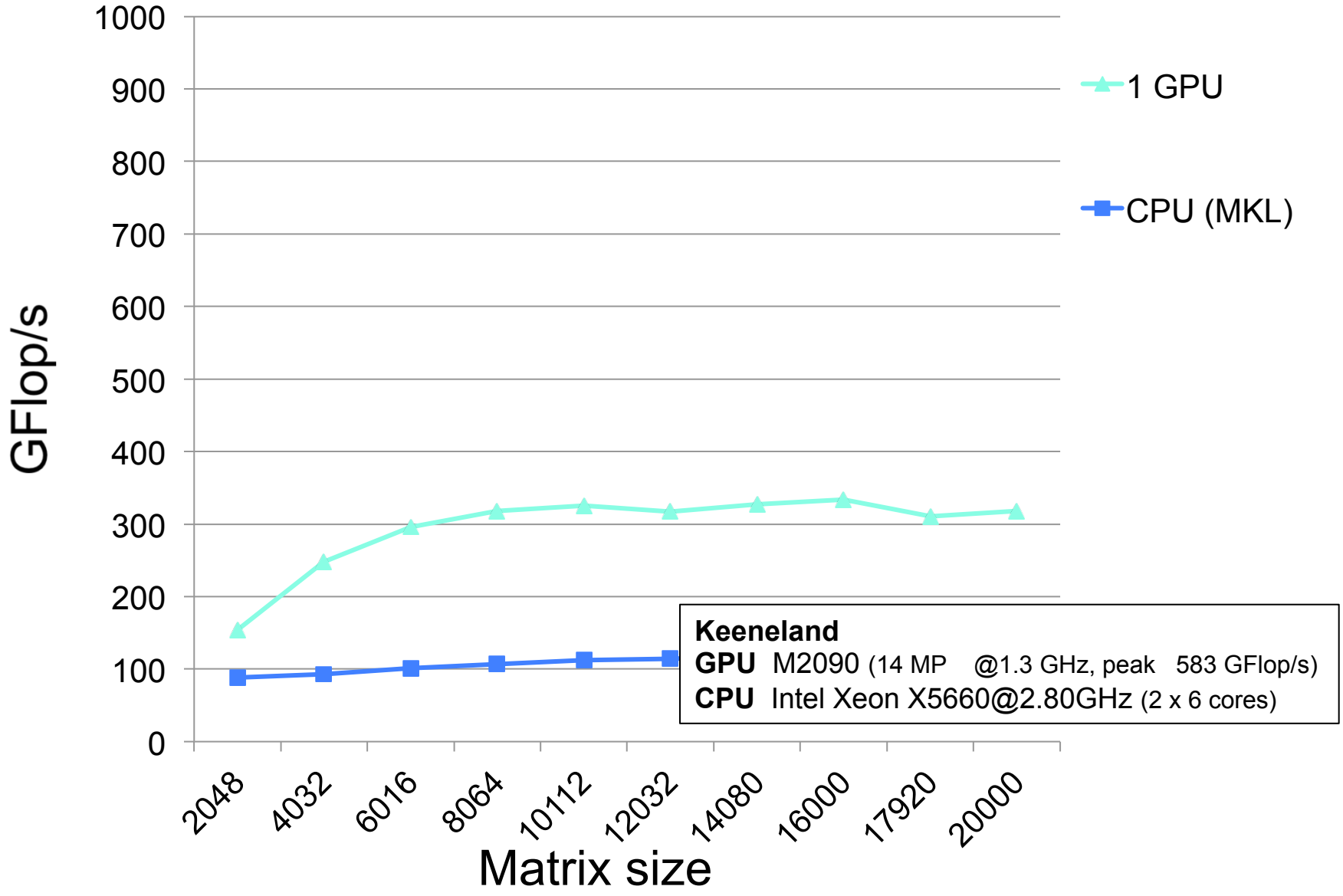
Each GPU uses existing algorithms

MultiGPU Support

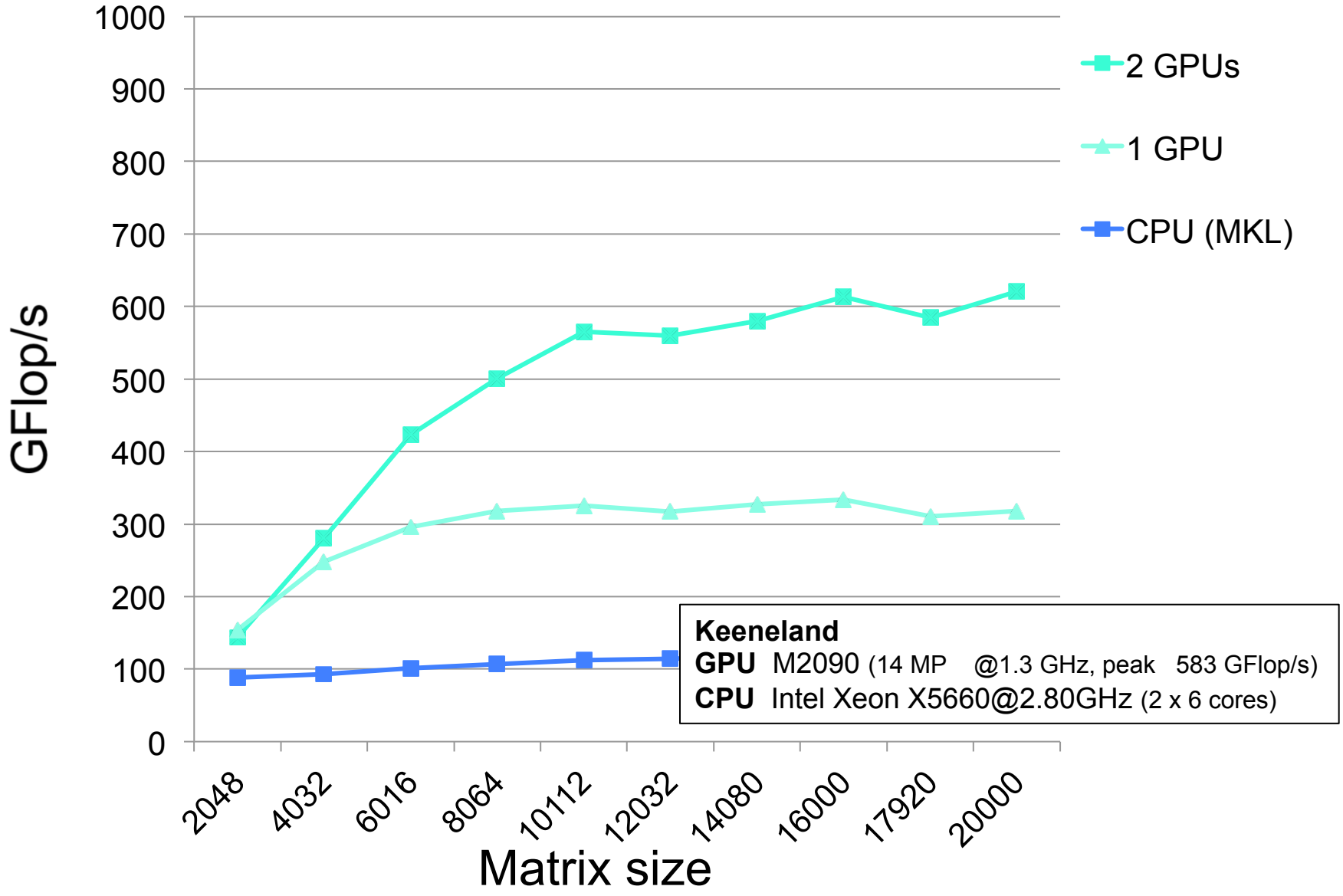
- Data distribution
  - 1-D block-cyclic distribution
- Algorithm
  - GPU holding current panel is sending it to CPU
  - All updates are done in parallel on the GPUs
  - Look-ahead is done with GPU holding the next panel



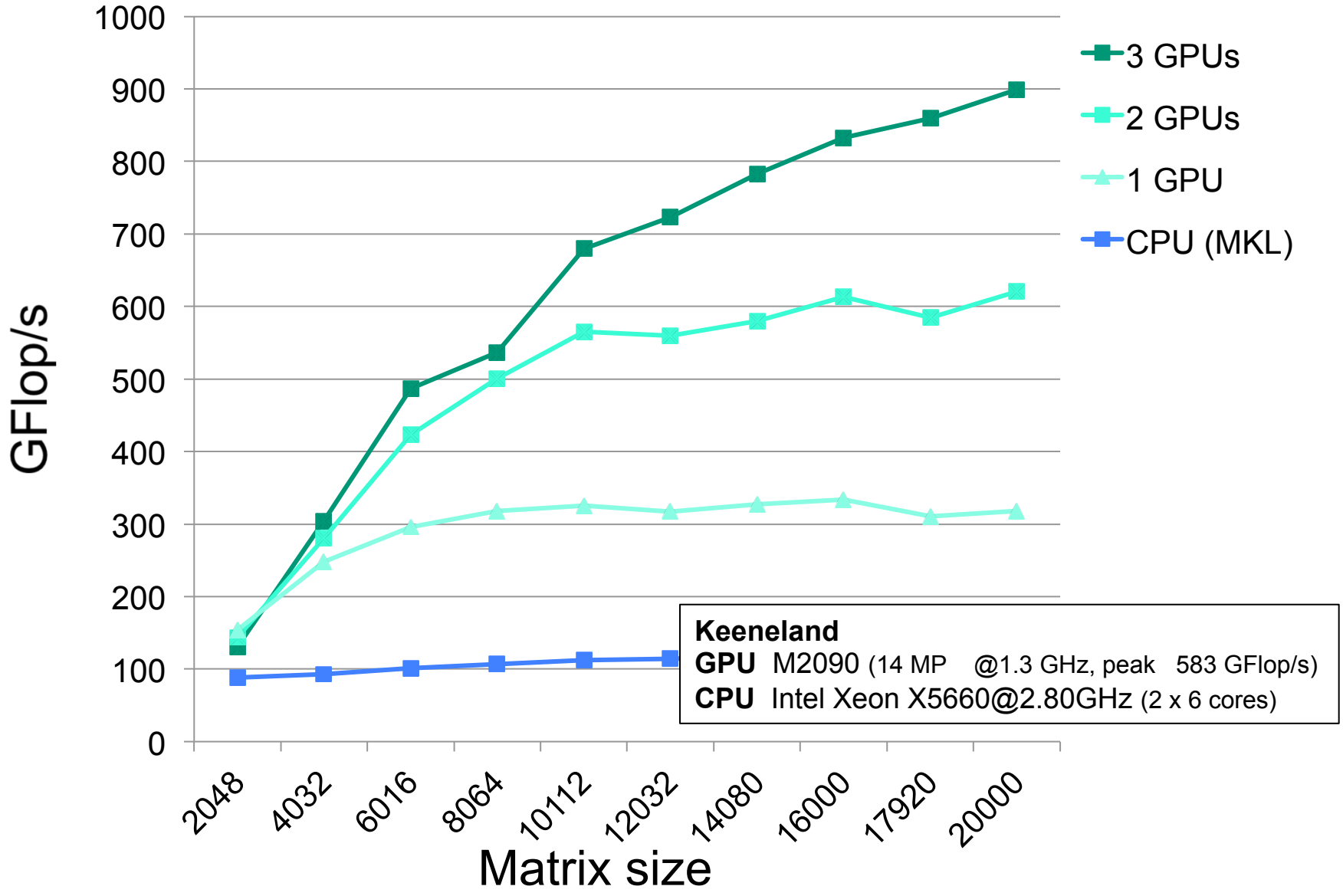
LU on multiple GPUs in DP



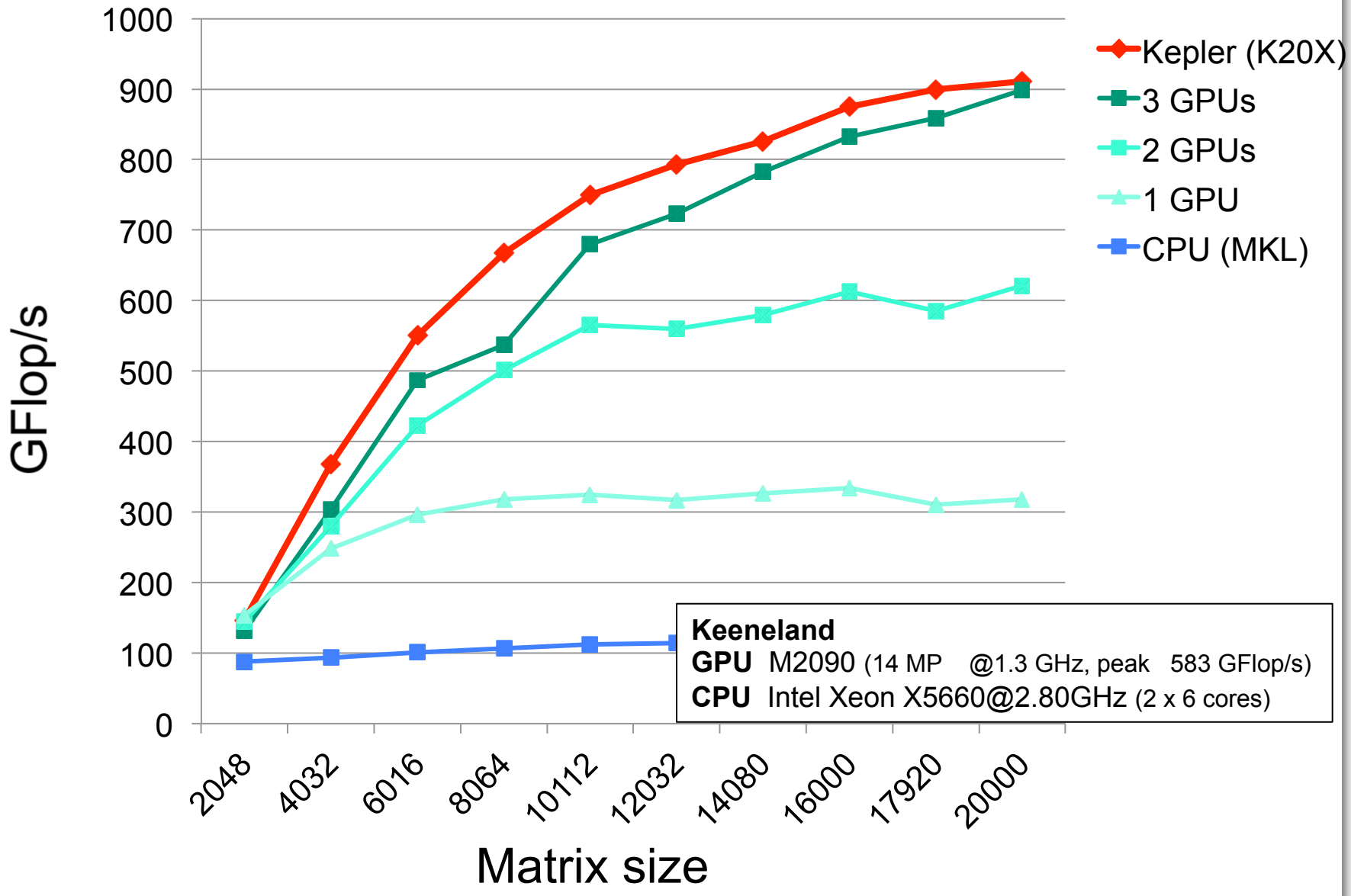
LU on multiGPUs in DP



LU on multiGPUs in DP



LU on Kepler in DP

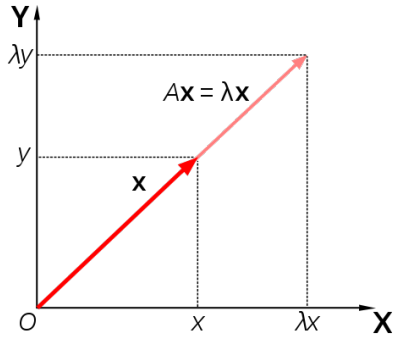


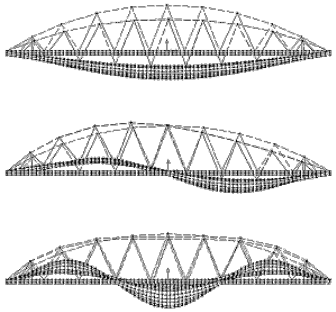
Eigenproblem Solvers in MAGMA

$A x = \lambda x$

- Quantum mechanics (Schrödinger equation)
- Quantum chemistry
- Principal component analysis (in data mining)
- Vibration analysis (of mechanical structures)
- Image processing, compression, face recognition
- Eigenvalues of graph, e.g., in Google's page rank

. . .





- Need to solve it fast

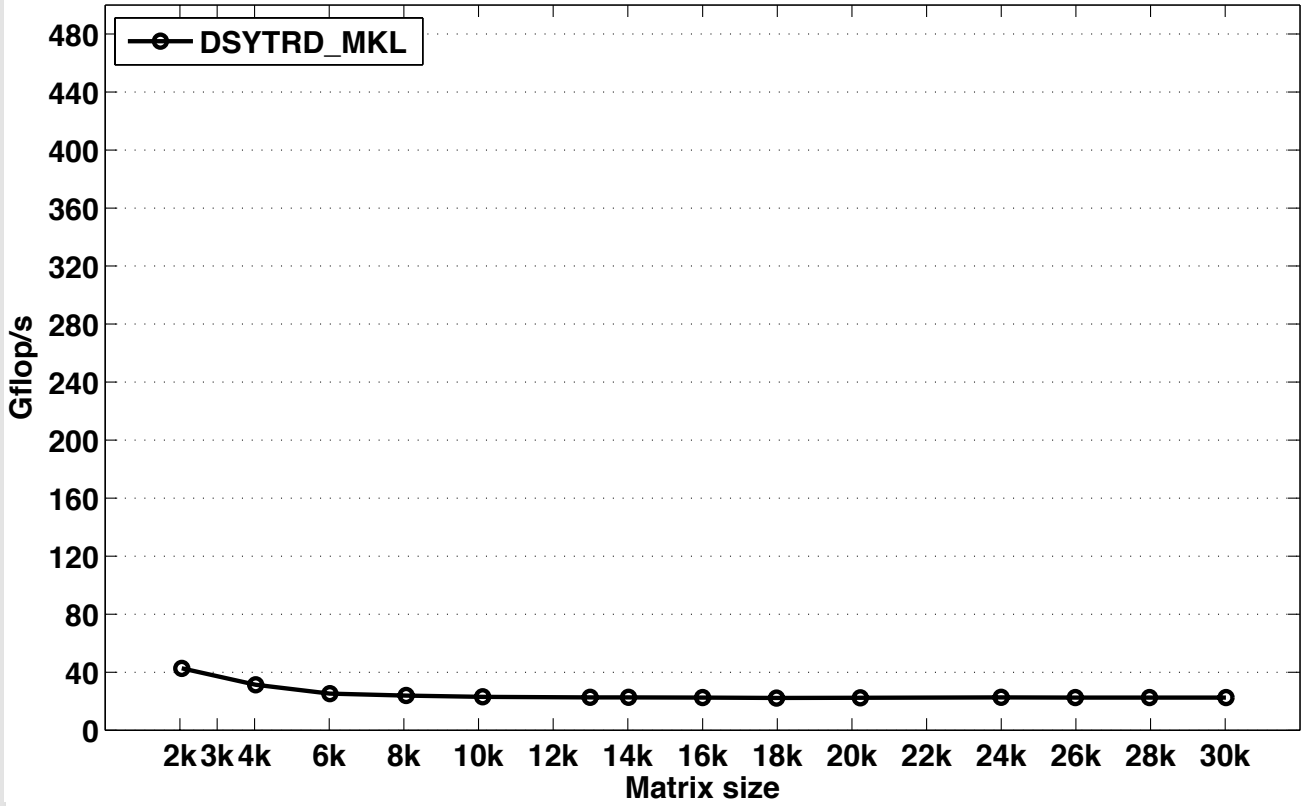
  Current MAGMA results:

  MAGMA with 1 GPU can be 12x faster vs. vendor libraries on state-of-the-art multicore systems

T. Dong, J. Dongarra, S. Tomov, I. Yamazaki, T. Schulthess, and R. Solca, *Symmetric dense matrix-vector multiplication on multiple GPUs and its application to symmetric dense and sparse eigenvalue problems*, ICL Technical report, 03/2012.

J. Dongarra, A. Haidar, T. Schulthess, R. Solca, and S. Tomov, *A novel hybrid CPU- GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks*, ICL Technical report, 03/2012.

Toward Fast Eigensolvers



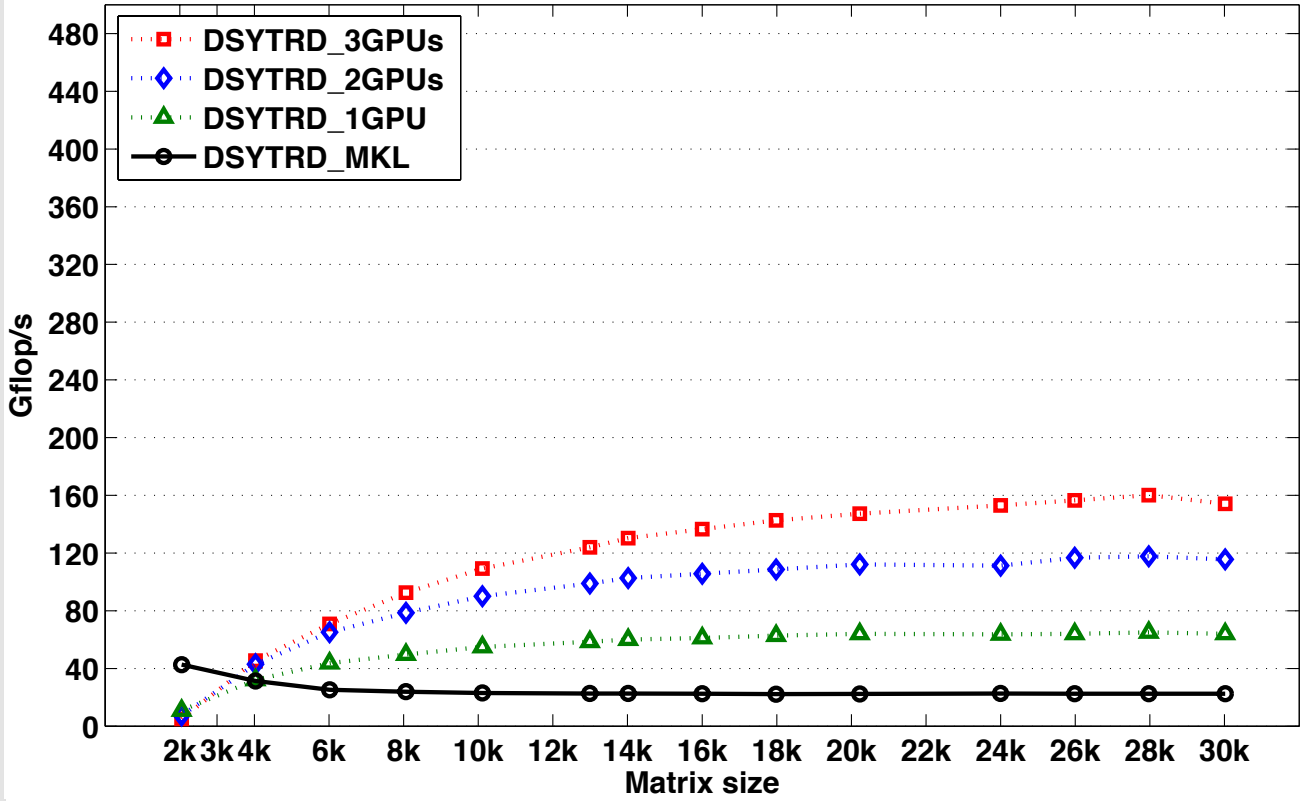
flops formula: $n^3/3/time$
Higher is faster

Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

- Characteristics
  - Too many Level 2 BLAS operations
  - Relies on panel factorization
  - ➔Bulk synchronous phases
  - ➔Memory-bound algorithm

A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks,* ICL Technical report, 03/2012.

Toward Fast Eigensolvers



flops formula: $n^3/3/time$

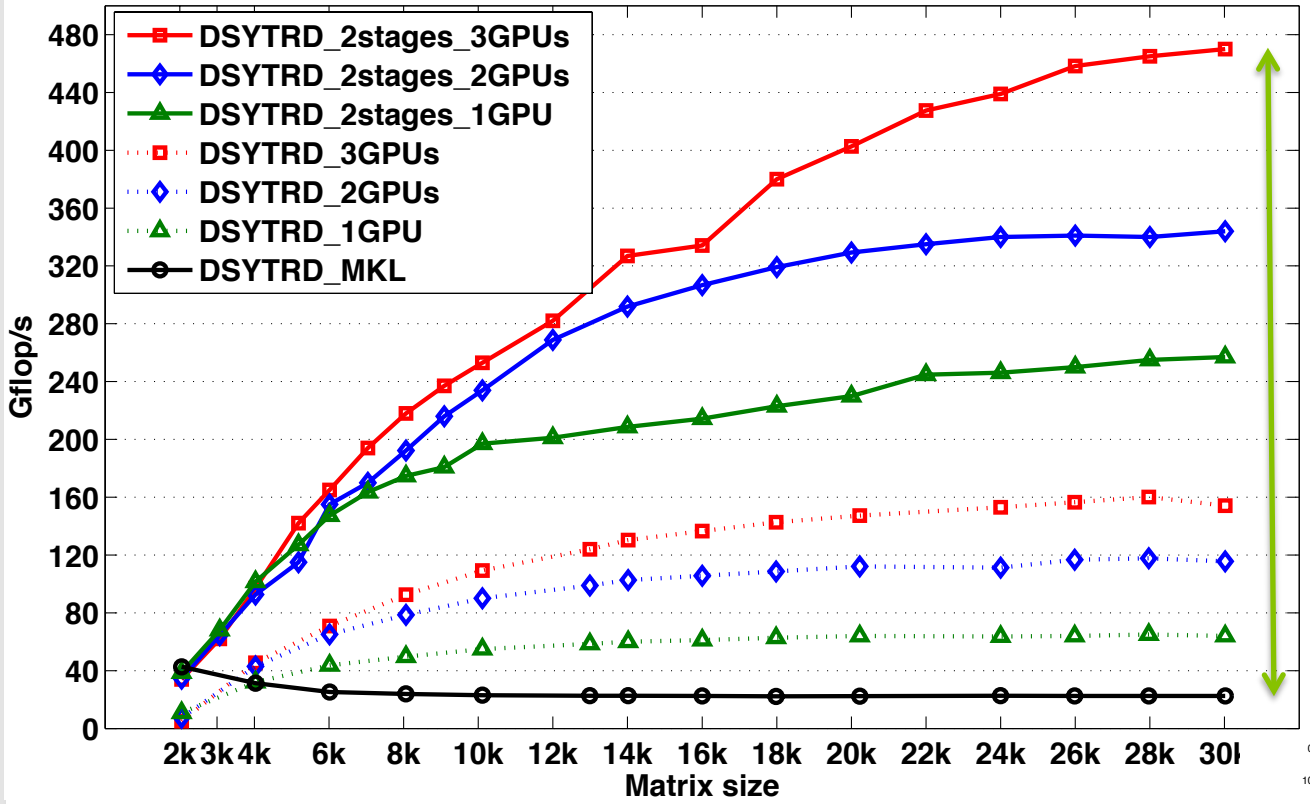
Higher is faster

Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

- Characteristics
  - Level 2 BLAS GEMV moved to the GPU,
  - Accelerate the algorithm by doing all Level 3 BLAS on the GPU,
  - ➔Bulk synchronous phases
  - ➔Memory-bound algorithm

A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks,* ICL Technical report, 03/2012.

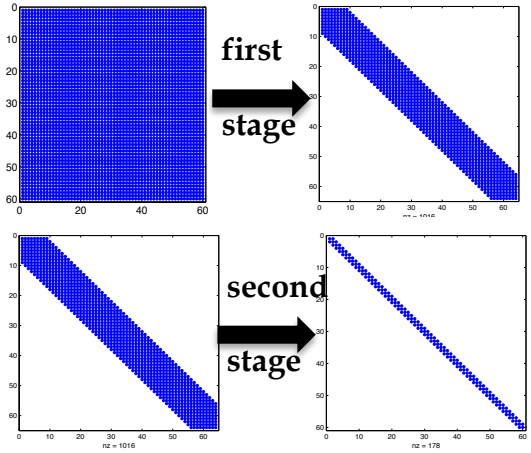
Toward Fast Eigensolvers



flops formula: $n^3/3/time$
Higher is faster

Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

Acceleration w/ 3 GPUs:
15x vs. 12 Intel cores



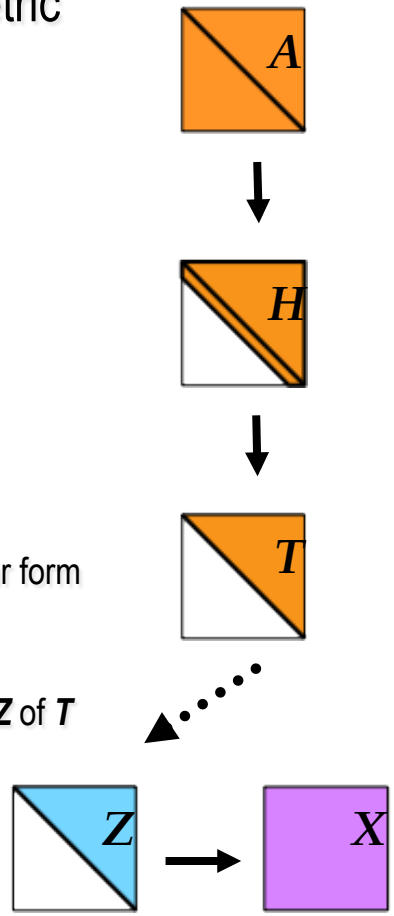
Characteristics

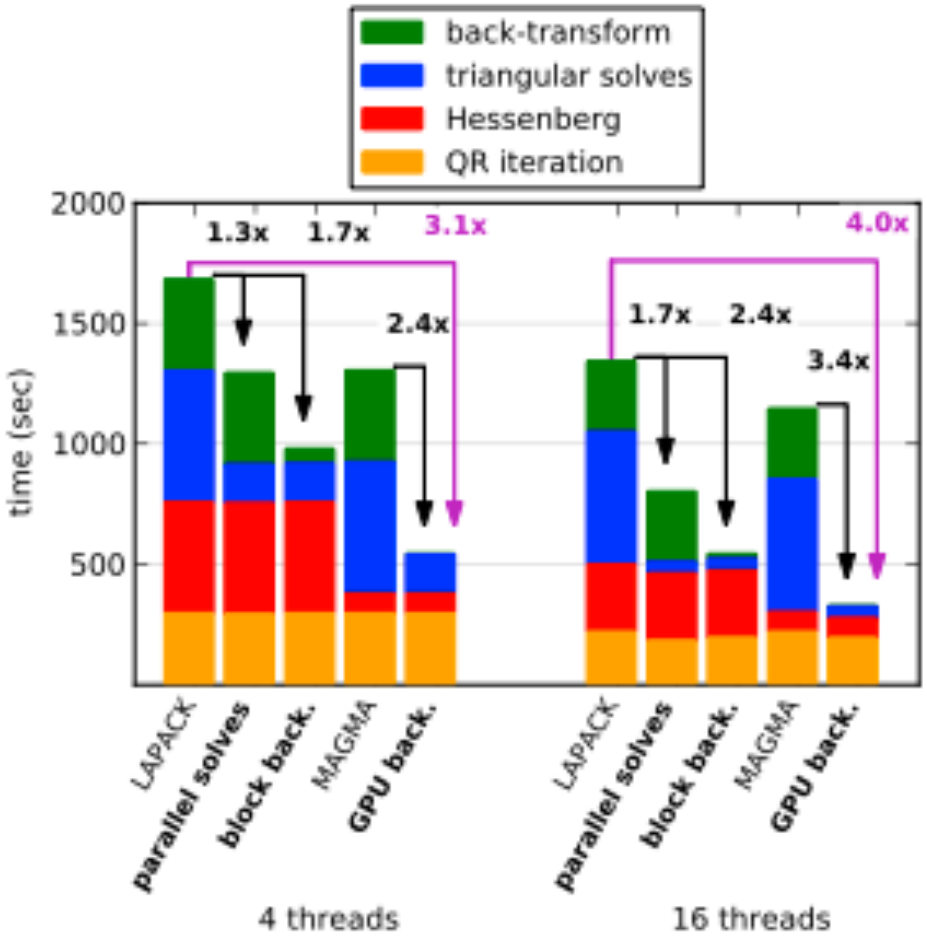
- Stage 1: BLAS-3, increasing computational intensity,
- Stage 2: BLAS-1.5, new cache friendly kernel,
- 4X/12X faster than standard approach,
- Bottelneck: if all Eigenvectors are required, it has 1 back transformation extra cost.

A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks,* ICL Technical report, 03/2012.

Toward Fast Eigensolvers for Non-Symmetric Matrices

- ***A*** is *n* x *n*, nonsymmetric
- $\boldsymbol{Ax} = \lambda \boldsymbol{x}$
- Three phases:
  - Hessenberg reduction
    $\boldsymbol{H} = \boldsymbol{Q}_1^T \boldsymbol{A}\, \boldsymbol{Q}_1$
  - QR iteration to triangular form
    $\boldsymbol{T} = \boldsymbol{Q}_2^T \boldsymbol{H}\, \boldsymbol{Q}_2$
  - Compute eigenvectors ***Z*** of ***T*** and back-transform to eigenvectors ***X*** of ***A***

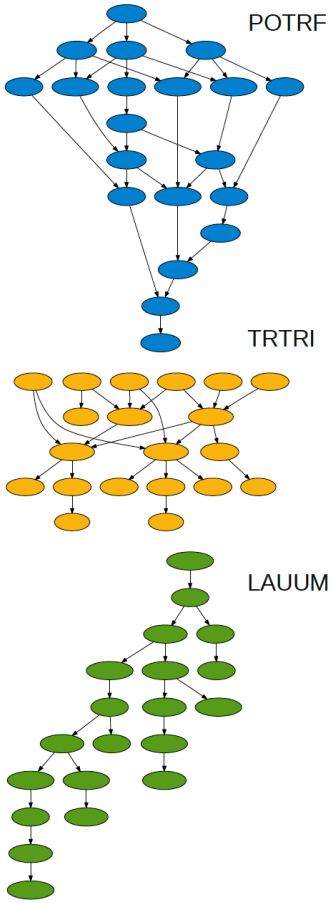


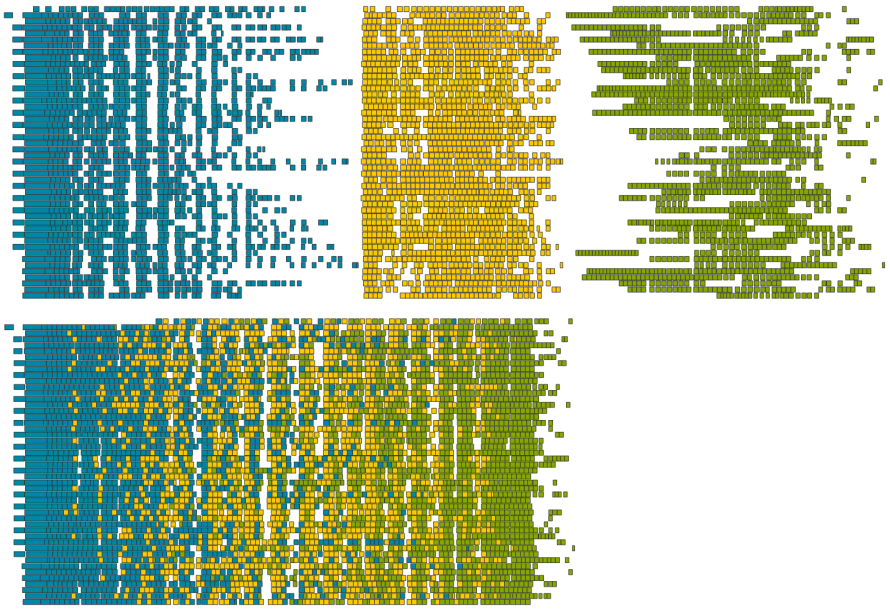


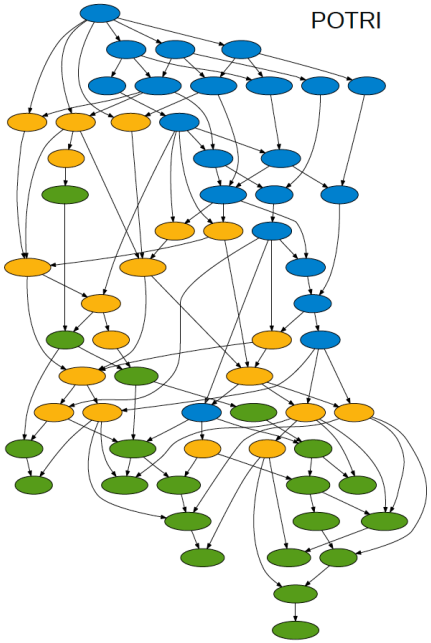
n = 16000, 2x8 core Intel Sandy Bridge, NVIDIA Kepler K40 GPU

Current Work: DAG Scheduling

- Schedule task execution using
  Dynamic Runtime Systems







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

Current Work: DAG Scheduling

High-productivity w/ Dynamic Runtime Systems
From Sequential Nested-Loop Code to Parallel Execution

```
for (k = 0; k < min(MT, NT); k++){
    zgeqrt(A[k;k], ...);
    for (n = k+1; n < NT; n++)
        zunmqr(A[k;k], A[k;n], ...);
    for (m = k+1; m < MT; m++){
        ztsqrt(A[k;k],,A[m;k], ...);
        for (n = k+1; n < NT; n++)
            ztsmqr(A[m;k], A[k;n], A[m;n], ...);
    }
}
```

Current Work: DAG Scheduling

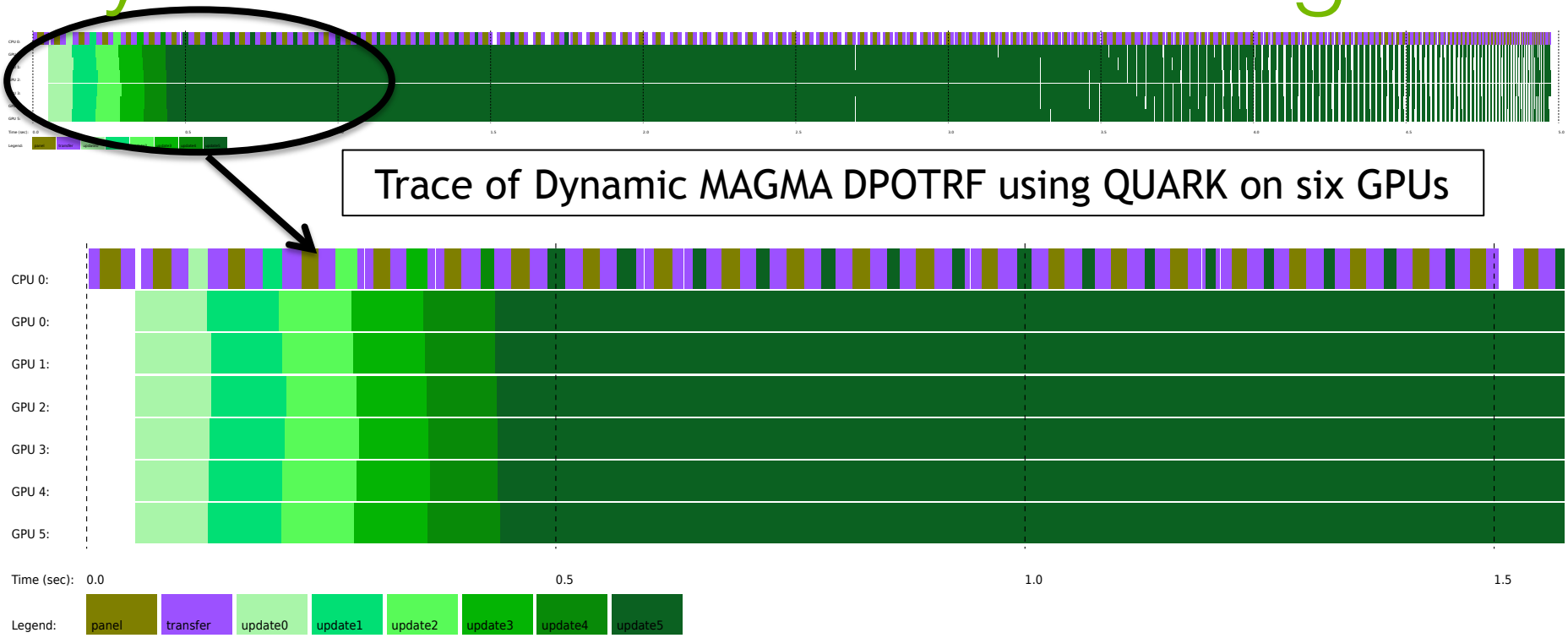
High-productivity w/ Dynamic Runtime Systems
From Sequential Nested-Loop Code to Parallel Execution

```
for (k = 0; k < min(MT, NT); k++){
    Insert_Task(&cl_zgeqrt, k , k, ...);
    for (n = k+1; n < NT; n++)
            Insert_Task(&cl_zunmqr, k, n, ...);
    for (m = k+1; m < MT; m++){
            Insert_Task(&cl_ztsqrt, m, k, ...);
            for (n = k+1; n < NT; n++)
                    Insert_Task(&cl_ztsmqr, m, n, k, ...);
    }
}
```

Various runtime systems can be used:

- **StarPU**
  http://icl.cs.utk.edu/projectsdev/morse
- **PaRSEC**
  https://icl.cs.utk.edu/parsec/
- **QUARK**
  http://icl.cs.utk.edu/quark/

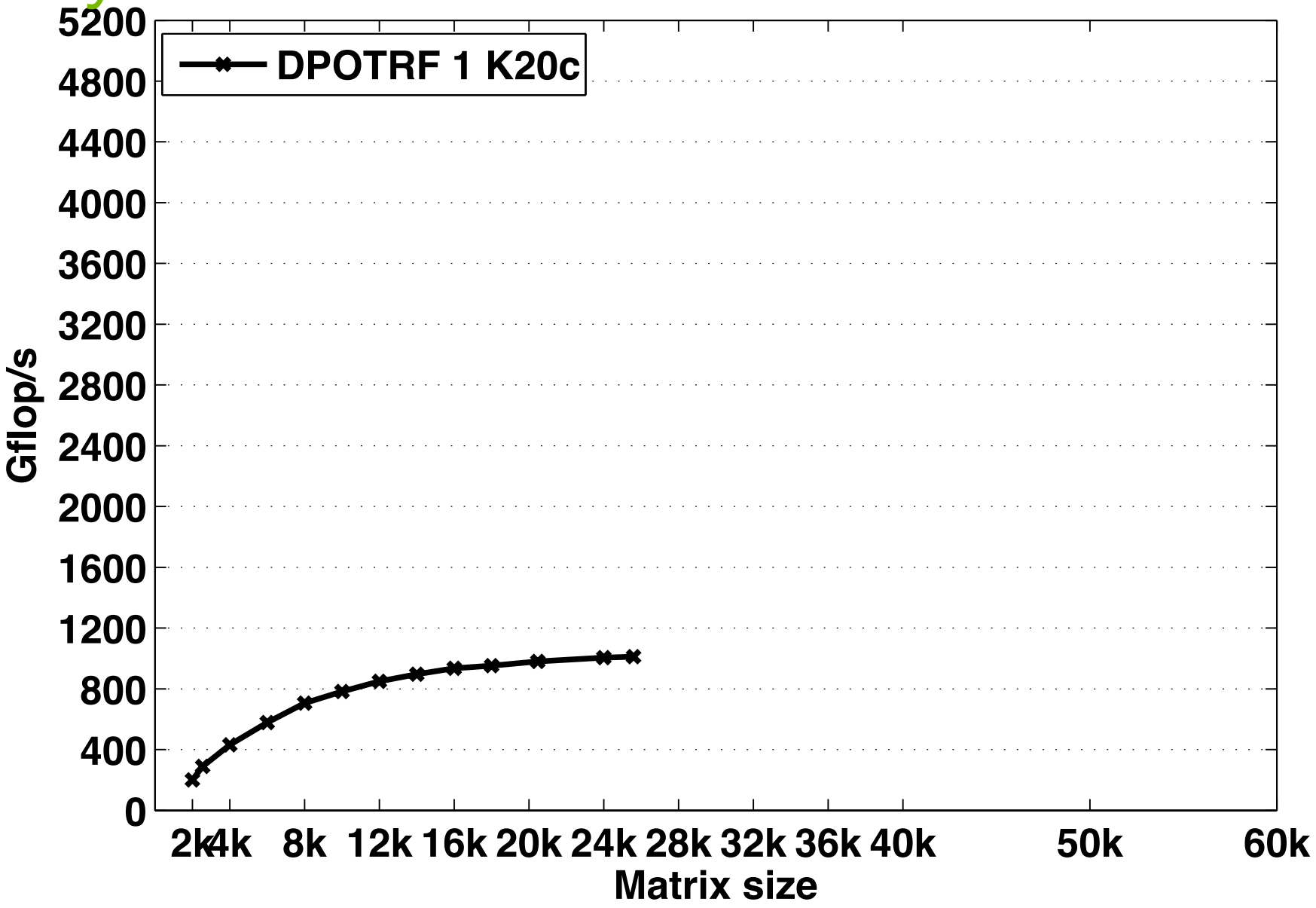
Dynamic MAGMA Scheduling



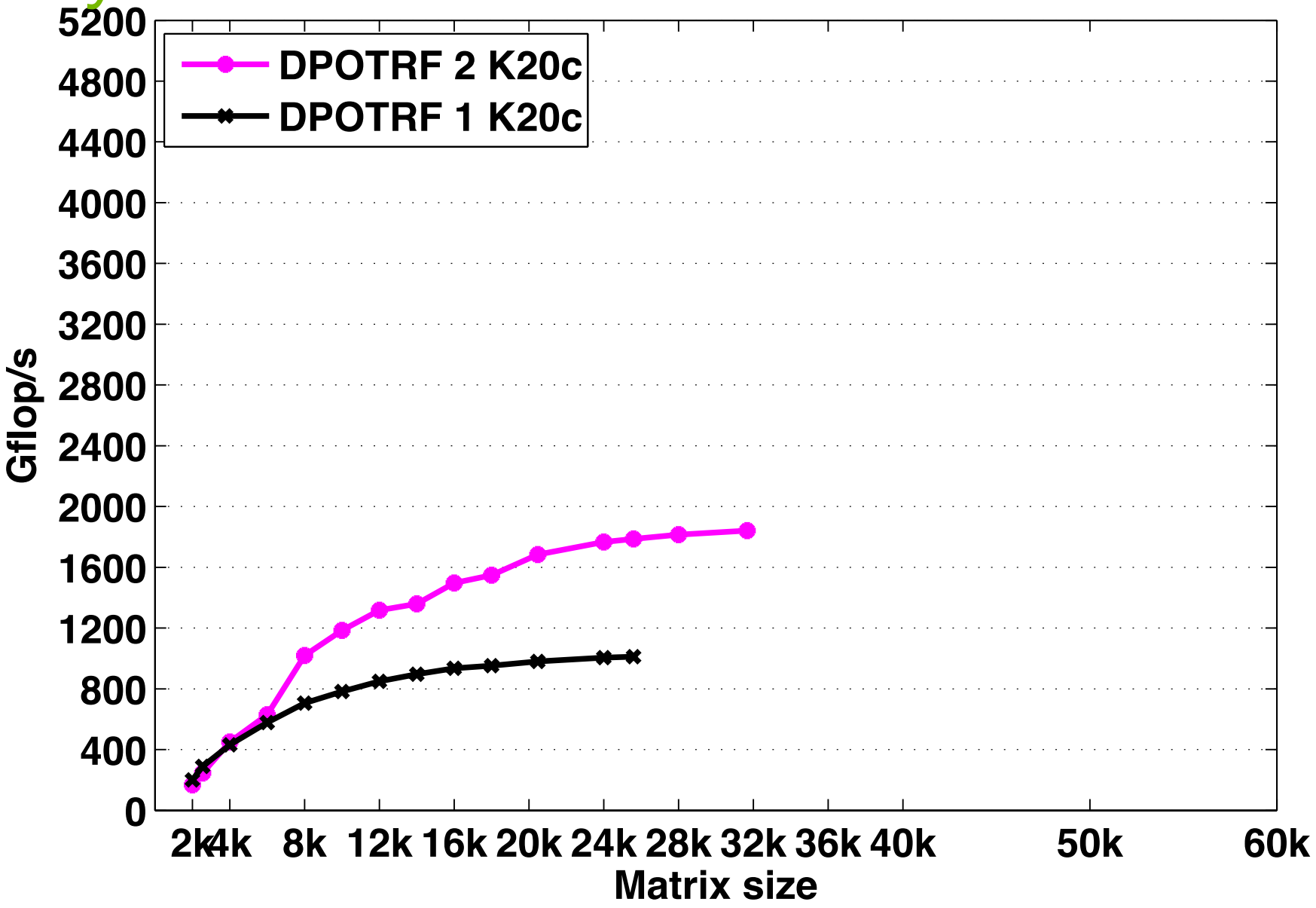
Scalability and efficiency :

- Snapshot of the execution trace of the Cholesky factorization for a matrix of size 40K using six GPUs K20c.
- As expected, the pattern of the trace looks compressed which means that our implementation is able to schedule and balance the tasks on the six GPU devices.

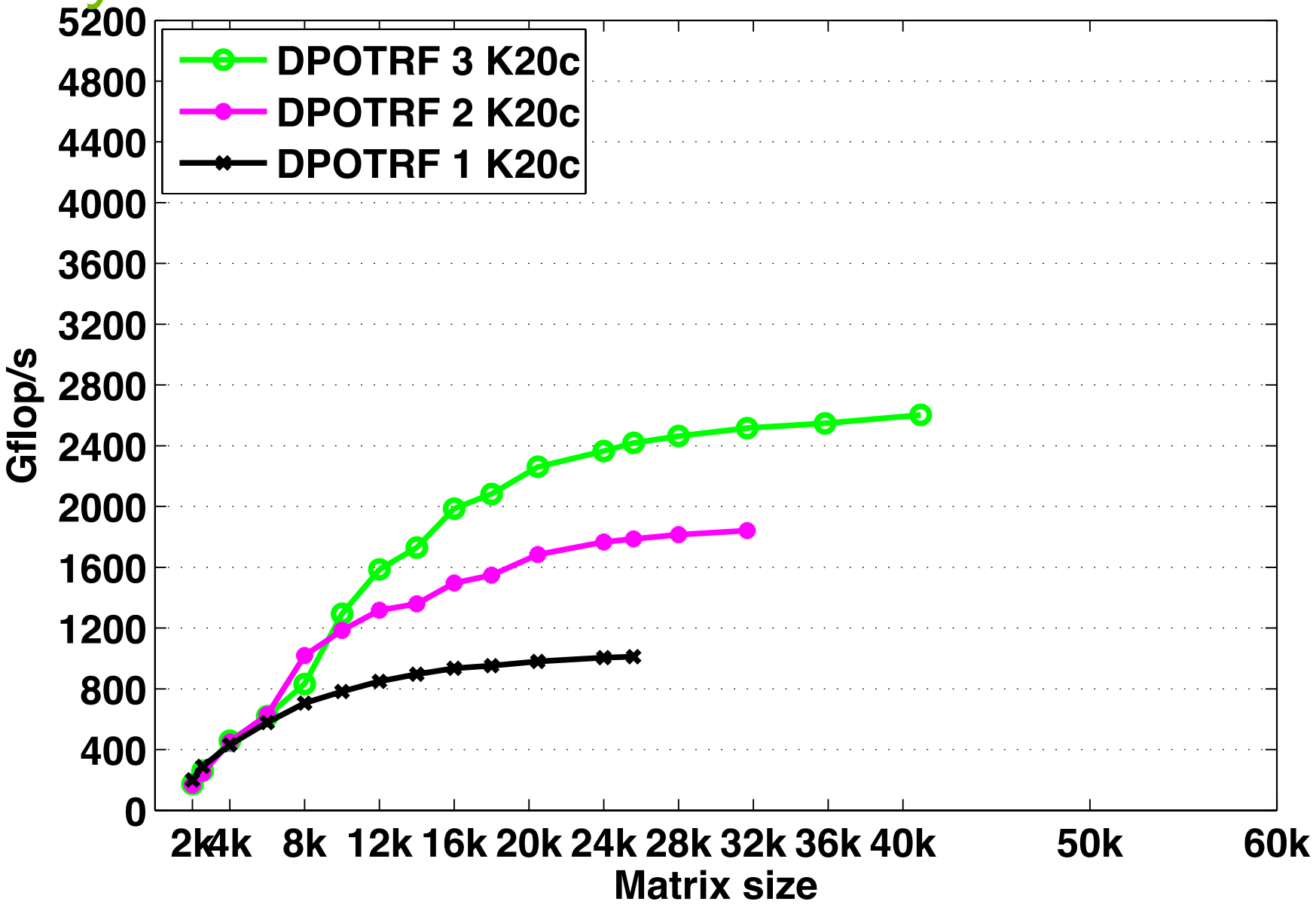
Dynamic MAGMA with QUARK



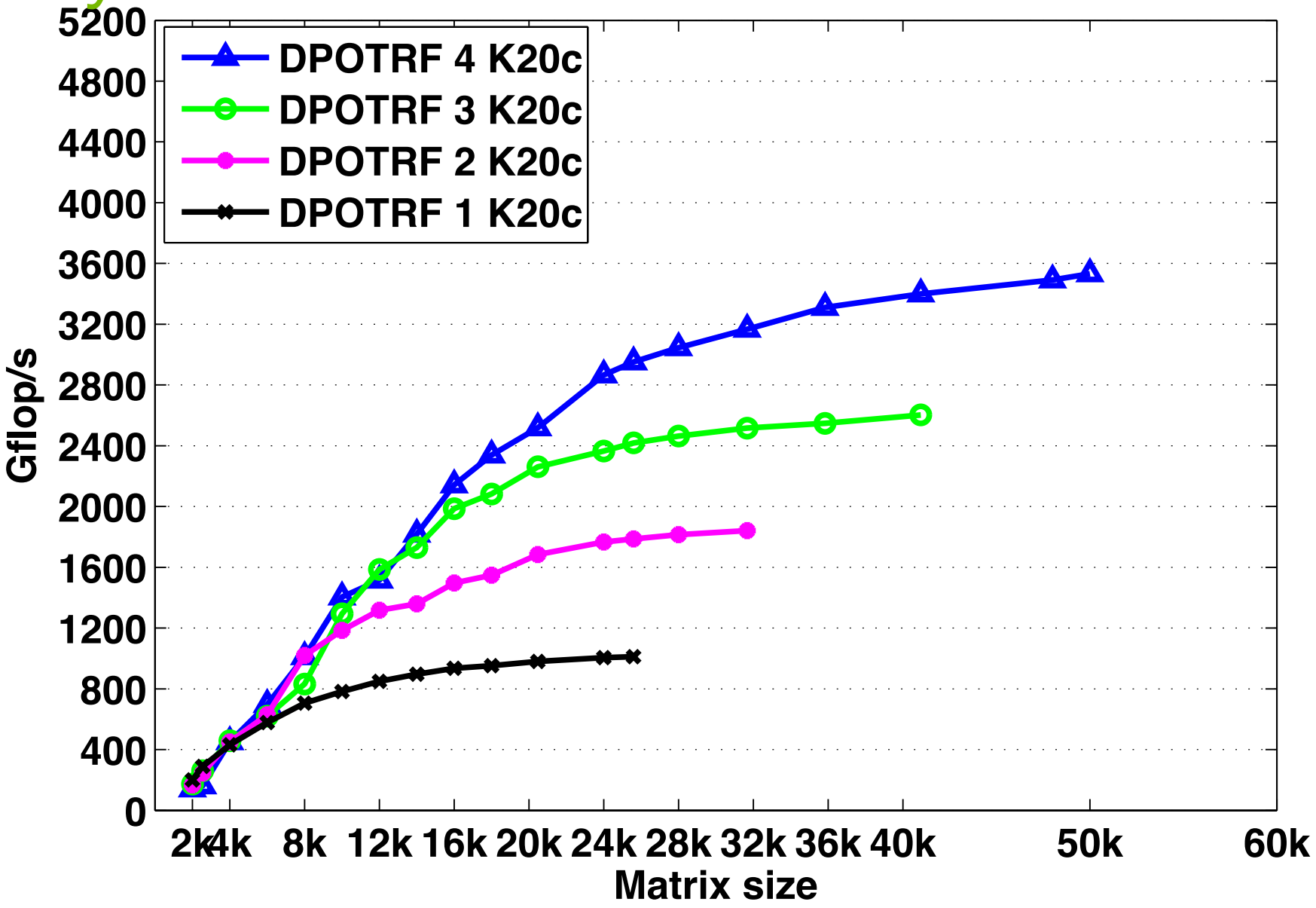
Dynamic MAGMA with QUARK



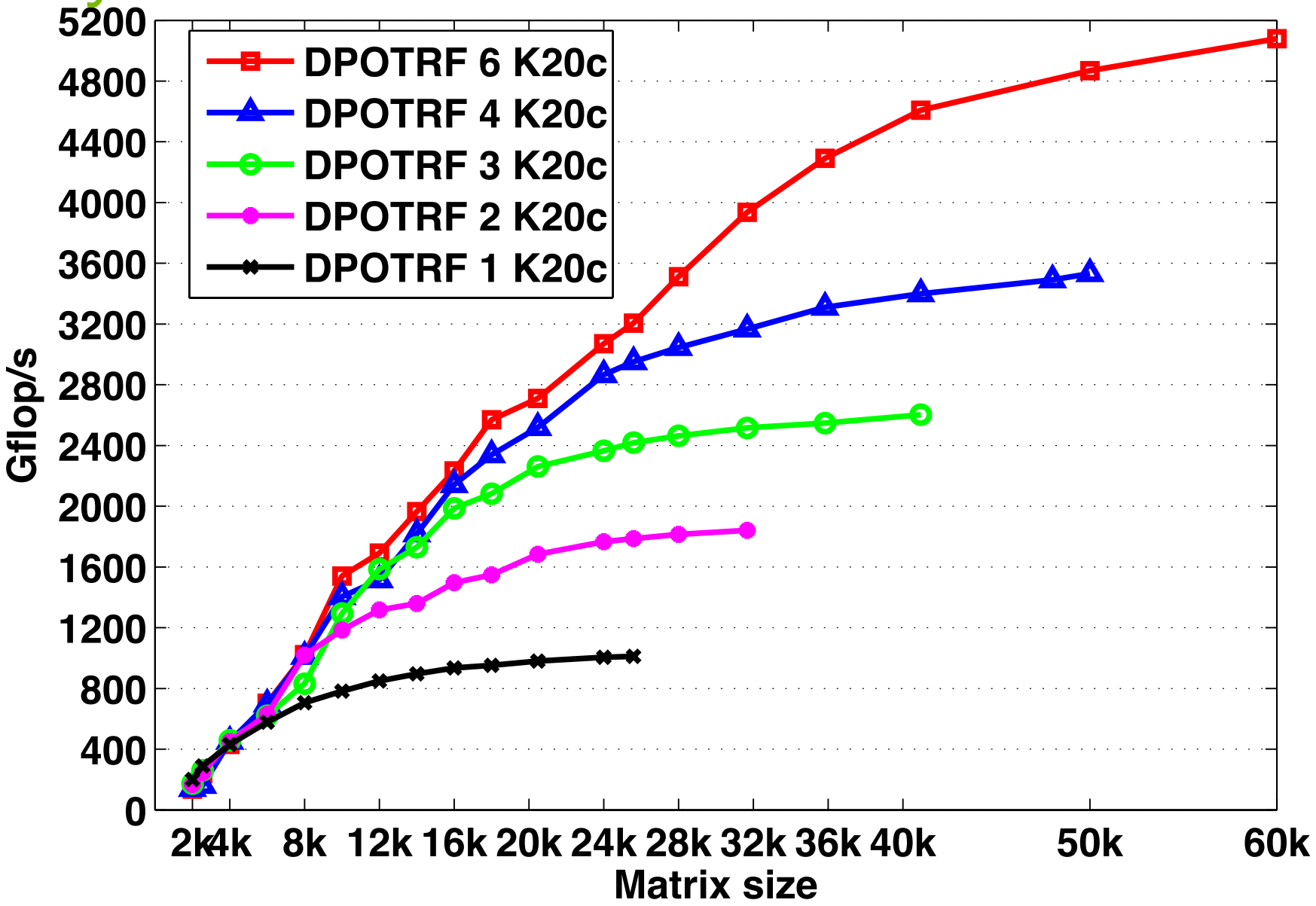
Dynamic MAGMA with QUARK



Dynamic MAGMA with QUARK



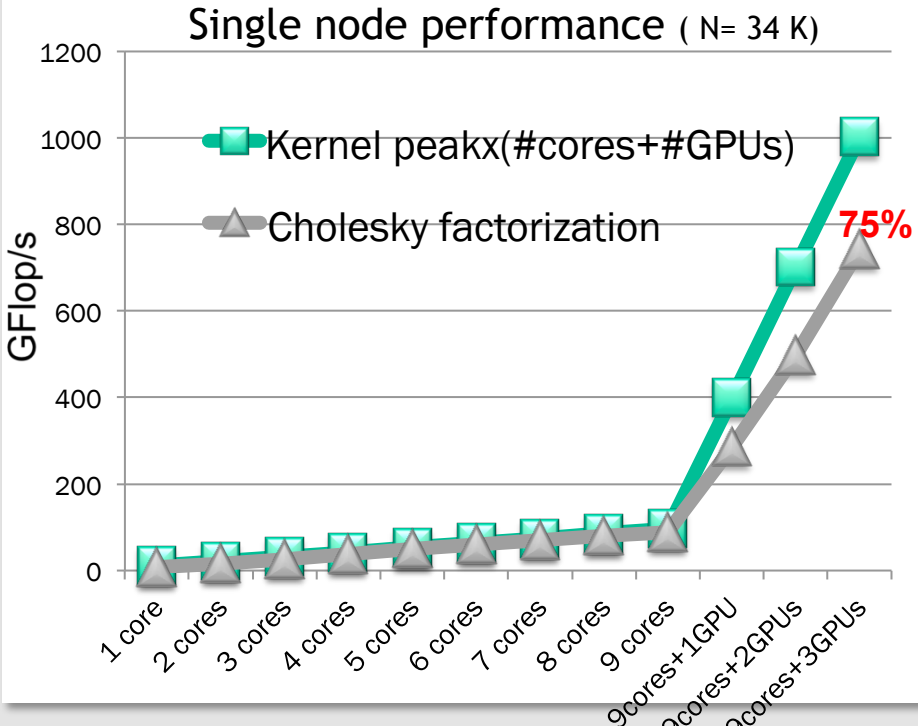
Dynamic MAGMA with QUARK

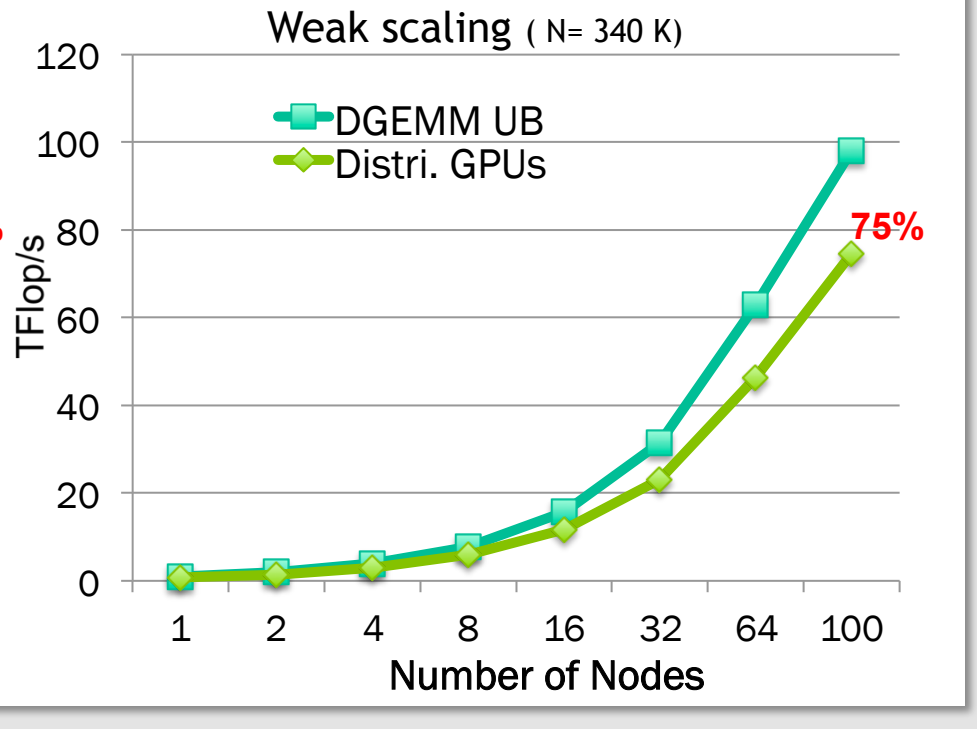


Distributed MAGMA

- Preliminary work on distributed memory systems
- Extensions of the Dynamic MAGMA
  - ScaLAPACK 2D block-cyclic data distribution
  - Lightweight "local" (node) scheduling with QUARK + MPI communications
  - Match in performance previous results using "tile" algorithms

    [F. Song, S. Tomov, and J. Dongarra, "Enabling and Scaling Matrix Computations on Heterogeneous Multi-Core and Multi-GPU Systems", ACM International Conference on Supercomputing (ICS 2012), San Servolo Island, Venice, Italy, June 2012.]

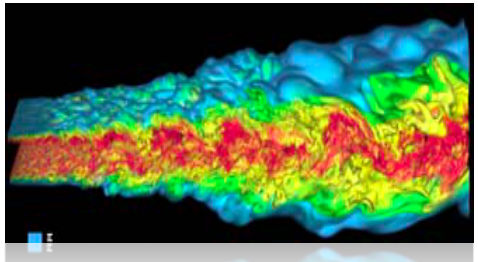
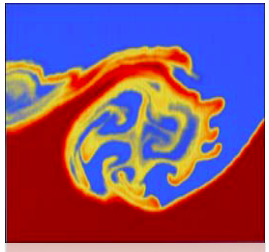
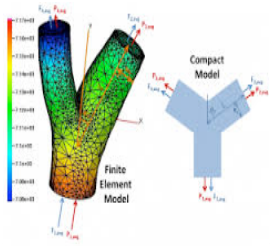


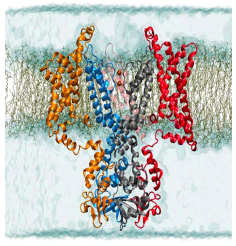


Sparse Solvers for GPUs

Sparse HPC on modern architectures

- Important scientific applications rely on sparse linear algebra



- HPCG – a new benchmark to complement TOP500 (HPL)
  - To solve A x = b, where A is large and sparse
  - To show essential communication & computation patterns in solving PDEs
  - To encourage the focus on architecture features and application needs
  - In collaboration with Sandia National Laboratory and University of Tennessee
- MAGMA Sparse
  - Develop GPU-aware Sparse Solvers
  - Support from DOE, DOD, and NVIDIA

MAGMA Sparse

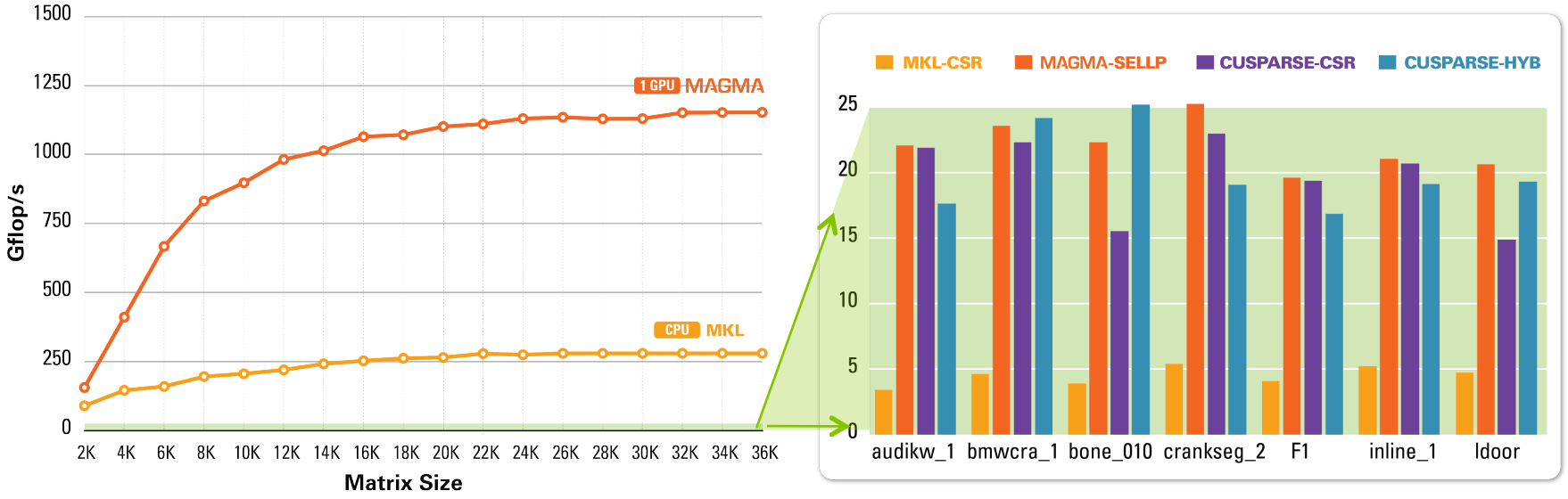
- Recently added MAGMA component for sparse linear algebra
- Under evaluation for release
  - Friendly users
  - Collaborators
  - Industry partners
- Current MAGMA Sparse functionality:
  - Krylov subspace iterative linear system and eigen-problem solvers
  - Support for various matrix formats
  - Sparse BLAS GPU kernels
  - Dense LA building blocks for preconditioners

Challenges on Modern Architectures

- The explosion of parallelism
  - single K40 has 2,880 CUDA cores
  - algorithms must account for these levels of parallelism
- The growing gap of compute *vs.* data-movement capabilities

K40 GPU computing efficiency on

Compute intensive (dense LU) *vs.* Memory-bound computation (SpMV)



GPU NVIDIA K40 (Atlas) 15 MP x 192 @ 0.88 GHz

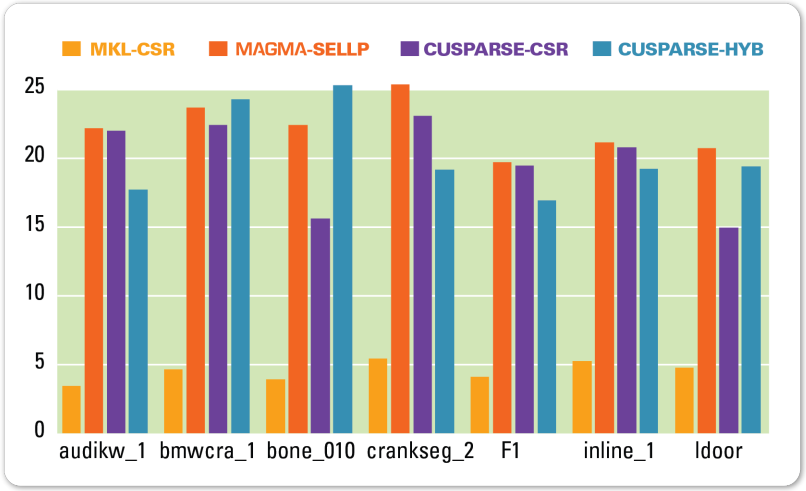
CPU Intel Xeon ES-2670 (Sandy Bridge) 2 x 8 cores @ 2.60 GHz

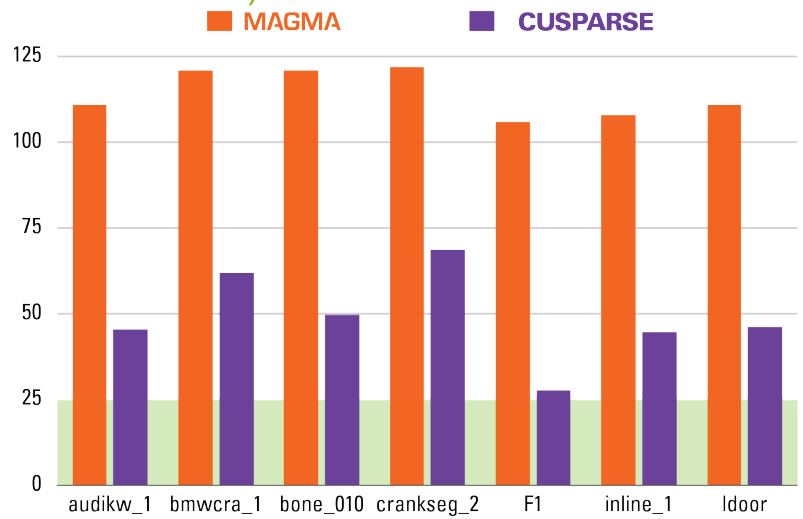
GPU-Aware Sparse Solvers

Breaking the memory-bound performance limitation

- Reference and optimized implementations
  - kernels & reduced communication
  - included are SpMV / SpMM in various formats, e.g., DENSE, CSR, Block-CSR, ELLPACK, ELLPACKT,ELLPACKRT, HYB, COO, CSC, SELL-C/SELLC-σ; and other kernels/building blocks …

Performance of SpMM for various matrices and a block of 32 vectors on a K40 GPU (GPU MAGMA & CUBLAS)





[1] H. Anzt, S. Tomov, and J. Dongarra, *"Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs"*, UTK Technical Report UT-EECS-14-727.

GPU-Aware Sparse Solvers

- Reference and optimized implementations
  - kernels & reduced communication
  - included are CG, BiCGSTAB, GMRES, preconditioned versions, CA-GMRES, and LOBPCG]

BiCGSTAB Method implementation

while $(k < maxiter)$ && $(res > \tau)$

$k := k + 1$	
$\rho_k := \hat{r}_0^T r_{k-1}$	(dot)
$\beta_{k+1} := \frac{\rho_k}{\rho_{k-1}} \frac{\alpha_{k-1}}{\omega_{k-1}}$	
$p_k := r_{k-1} + \beta (p_{k-1} - \omega_{k-1} v_{k-1})$	(scal + 2 axpy)
$v_k := A p_k$	(SpMV)
$\alpha_k := \frac{\rho_k}{\hat{r}_0^T v}$	(dot)
$s_k := r_{k-1} - \alpha v_k$	(copy + axpy)
$t_k := A s_k$	(SpMV)
$\omega_k := \frac{s_k^T t_k}{t_k^T t_k}$	(2 dot)
$x_{k+1} := x_k + \alpha_k p_k + \omega_k s_k$	(2 axpy)
$r_k := s_k - \omega t_k$	(copy + axpy)
$res = r_k^T r_k$	(dot)

end

Optimized

```
while( ( k < maxiter ) && ( res_host > epsilon ) ){
    magma_dbicgmerge_p_update<<<Gs, Bs, 0>>>
                    ( n, skp, v, r, p );

    magma_dbicgmerge_spmv1<<<Gs, Bs, Ms1>>>
                ( n, valA, rowA, colA, p, r, v, d1 );
    magma_zbicgmerge_reduce1( n, Gs, Bs, d1, d2, skp );

    magma_dbicgmerge_s_update<<<Gs, Bs, 0>>>
                    ( n, skp, r, v, s );

    magma_dbicgmerge_spmv2<<<Gs, Bs, Ms2>>>
                ( n, valA, rowA, colA, s, t, d1 );
    magma_dbicgmerge_reduce2( n, Gs, Bs, d1, d2, skp);

    magma_dbicgmerge_xr_update<<<Gs, Bs, 0>>>
                    ( n, skp, r_hat, r, p, s, t, x, d1);
    magma_dbicgmerge_reduce3( n, Gs, Bs, d1, d2, skp);
    magma_memcopy( 1, skp+5, res_host );
    k++;
}
```

[2] H. Anzt, S. Tomov, P. Luszczek, I. Yamazaki, J. Dongarra, and W. Sawyer, "*Optimizing Krylov Subspace Solvers on Graphics Processing Units*", in IEEE IPDPS ASHES'14 Proceedings, Phoenix, Arizona, May 19—23, 2014.

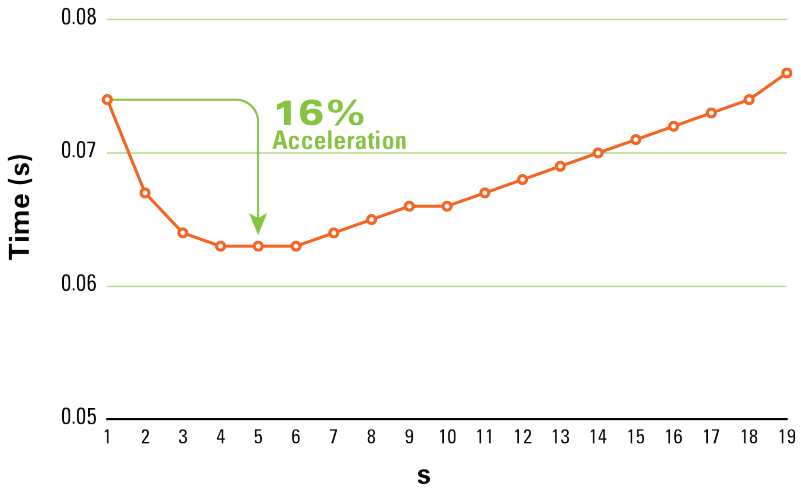
GPU-Aware Sparse Solvers

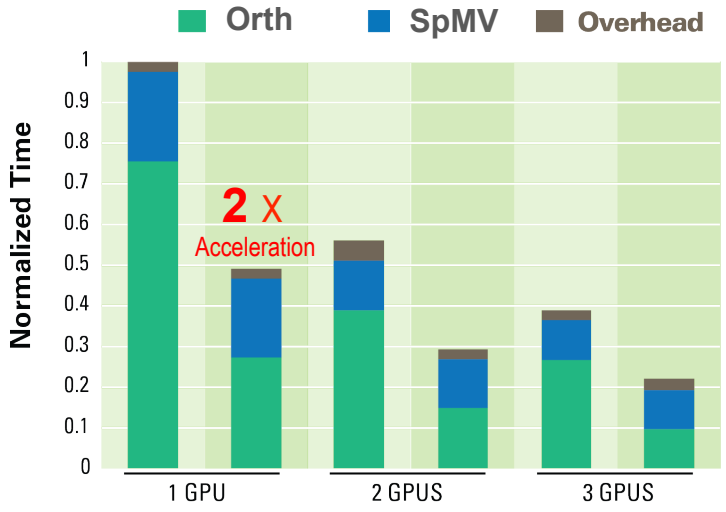
- Communication avoiding GMRES (CA-GMRES)
  - Replacing GMRES' SpMV with Matrix Powers Kernel (MPK):

$$v_{k+1} = A\, v_k \quad \text{for } k = j, \ldots, j+s$$

Level 2 BLAS ➔ Level 3 BLAS-based orthogonalization (coming next …)

MPK to generate 100 vectors for various s	Overall performance improvement on up to 3 GPUs





[3] I. Yamazki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra, "*Improving the Performance of CA-GMRES on Multicores with Multiple GPUs*", Proc. of IPDPS 2014 (accepted), Phoenix, Arizona, May 19—23, 2014.

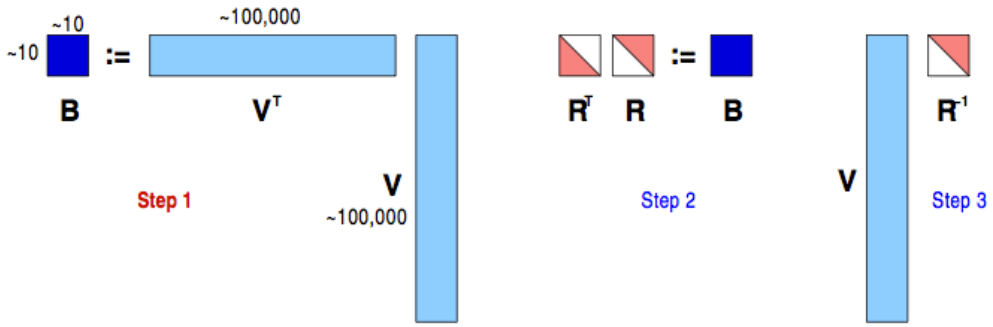
Orthogonalization Procedures

- Mixed-precision Cholesky QR
  - CholQR obtains BLAS-3 performance, but error is bounded by $\varepsilon\ \kappa(V)^2$
  - Remove the "square" by selectively using double-double (doubled) precision

Step 1 Gram-matrix formation $B := V^T V$
on GPUs in doubled-precision.

Step 2 Cholesky factorization $R^T R := B$
on CPUs in doubled-precision.

Step 3 Backward-substitution $Q := VR^{-1}$
on GPUs in standard-precision.

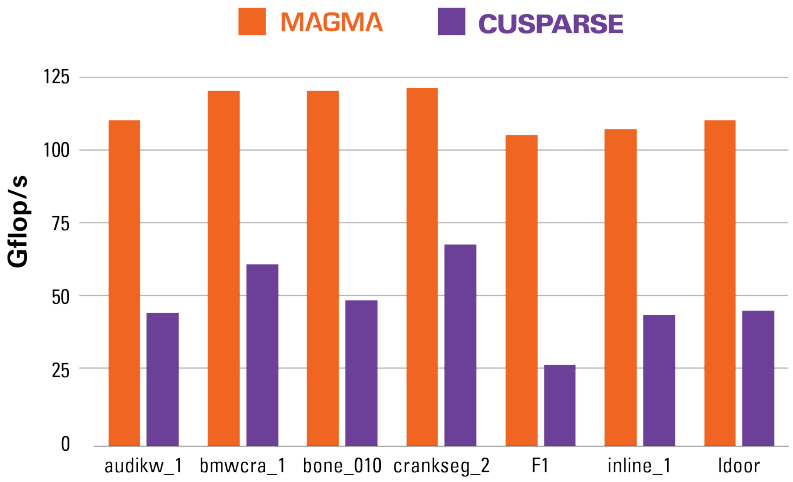


[4] I. Yamazaki, S. Tomov, T. Dong, and J. Dongarra, "*Mixed-precision orthogonalization scheme and adaptive step size for CA-GMRES on GPUs*", VECPAR 2014 (accepted), Eugene, Oregon, USA, June 30—July 3, 2014.

GPU-Aware Sparse Eigen-Solvers

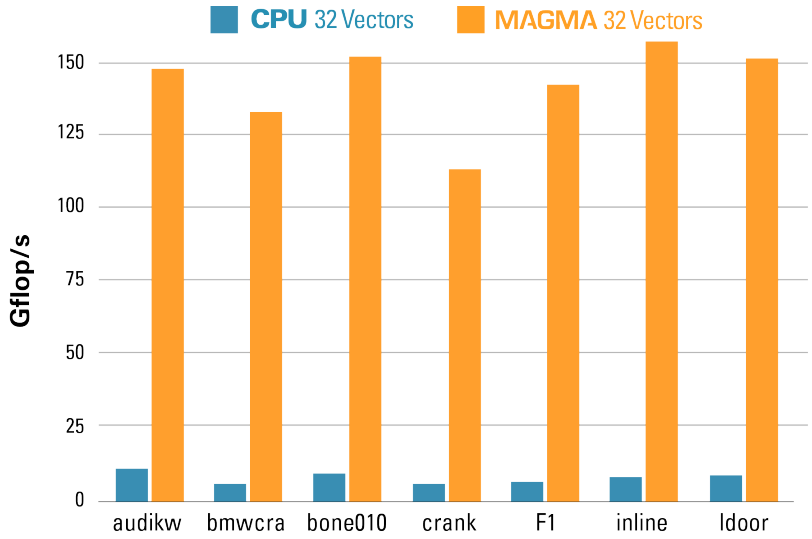
- Locally Optimal Block PCG (LOBPCG)
  - Find a set of smallest eigen-states of a sparse SPD matrix ($A x = \lambda x$)
  - Replace finding the states one-by-one by a block algorithm
    - Simultaneous finding; needs fast SpMM, re-orthogonalization, and GEMM of particular sizes

Performance of SpMM with various matrices (x 32 vec.)



GPU K40 (all 16 cores) **CPU** 2 x 8-core Intel Sandy Bridge + GPU

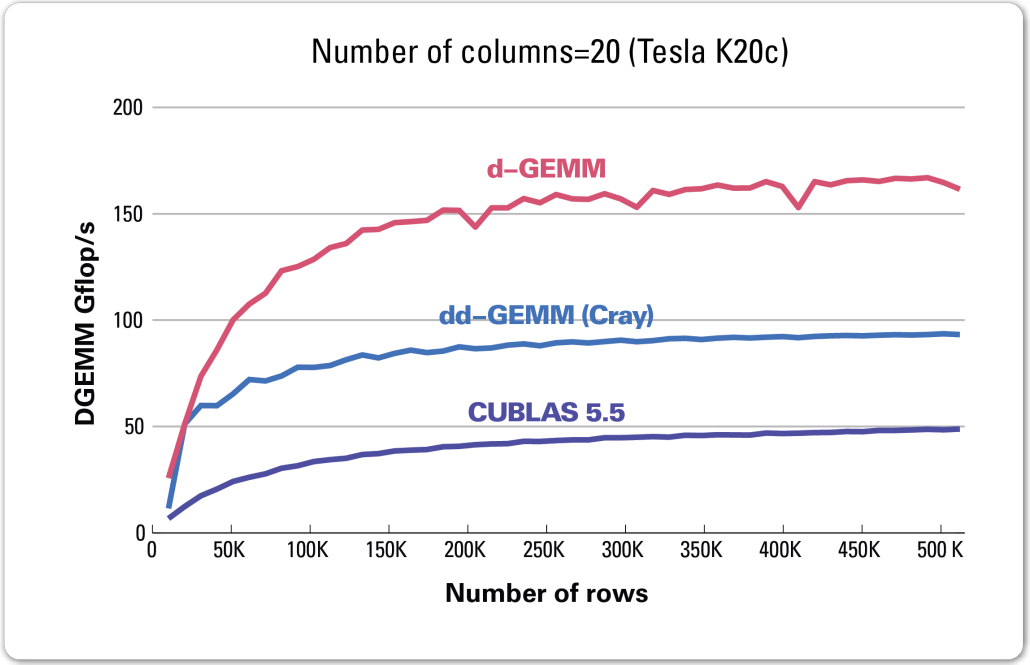
Overall speedup vs. LOBPCG from BLOPEX on CPUs



BLOPEX LOBPCG: uses CPU **MAGMA Sparse:** uses CPU

Batched Dense Linear Algebra and Other Building Blocks

- Many small DLA problems solved in parallel
  - Needed in preconditioners [5,6], orthogonalization routines [4,5], some sparse direct solvers, high-order FEMs [8], and batched higher-level DLA [7]



[5] I. Yamazaki, A. Napov, S. Tomov, and J. Dongarra, "*Computing Low-Rank Approximation of Dense Submatrices in a Heirarchically Semiseparable Matrix and its GPU Acceleration*", UTK Technical Report, August, 2013.

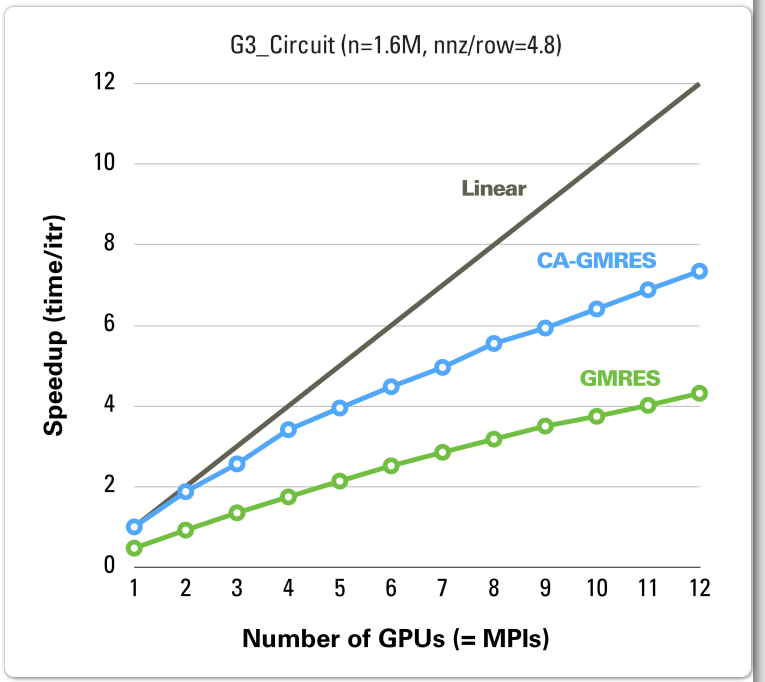
[6] D. Lukarski, H. Anzt, S. Tomov, and J. Dongarra, "*Hybrid Multi-Elimination ILU Preconditioners on GPUs*", 23rd Heterogeneity in Computing Workshop (HCW 2014), in Proc. of IPDPS 2014 (accepted), Phoenix, Arizona, May 19—23.

[7] T. Dong, A. Haidar, S. Tomov, and J. Dongarra, "A Fast *Batched Cholesky Factorization on a GPU*", ICPP 2014 (submitted), Minneapolis, MN, USA, September 9—12, 2014.

[8] T. Dong, V. Dobrev, T. Kolev, R. Rieben, S. Tomov, and J. Dongarra, "*A Step towards Energy Efficient Computing: Redesigning a Hydrodynamic Application on CPU-GPU*", IPDPS 2014 (accepted), Phoenix, AZ, May 19—23, 2014.

Future Directions

- Distributed multi-GPU solvers
  - Scheduling
  - Reproducibility
- Further tuning
  - partitioning
  - Communication-computation overlap
  - autotuning
- Extended functionality
  - Other mixed precision techniques
  - Other communication-avoiding techniques
  - Sparse direct multifrontal solvers & preconditioners
  - Other techniques of bringing Level 3 BLAS performance to sparse solvers



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