MAGMA: Development of High-Performance Linear Algebra for GPUs

J. Dongarra

Stan Tomov, I. Yamazaki, A. Haidar, M. Gates,
P. Luszczek, H. Anzt, and T. Dong
University of Tennessee, Knoxville

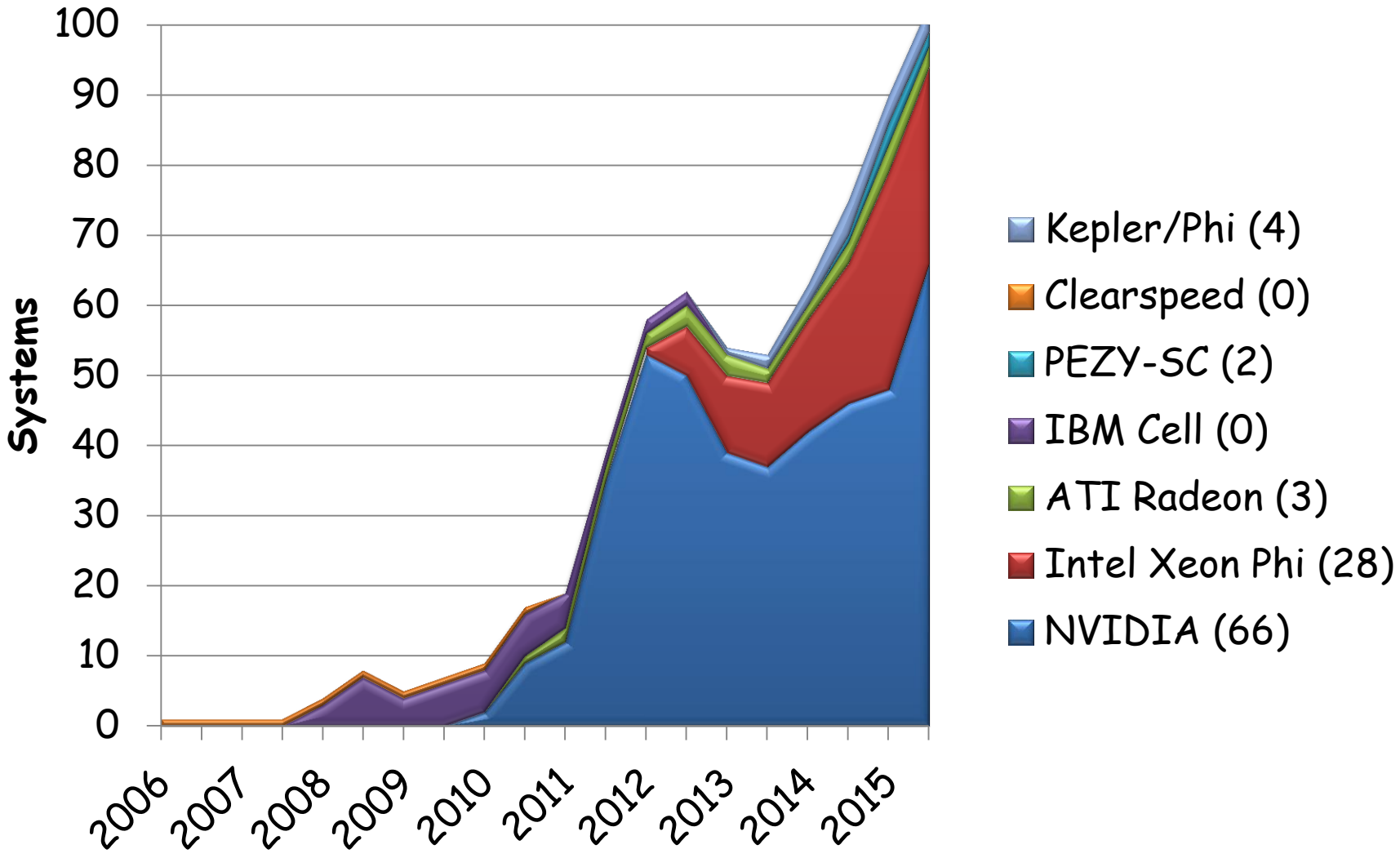
Outline

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

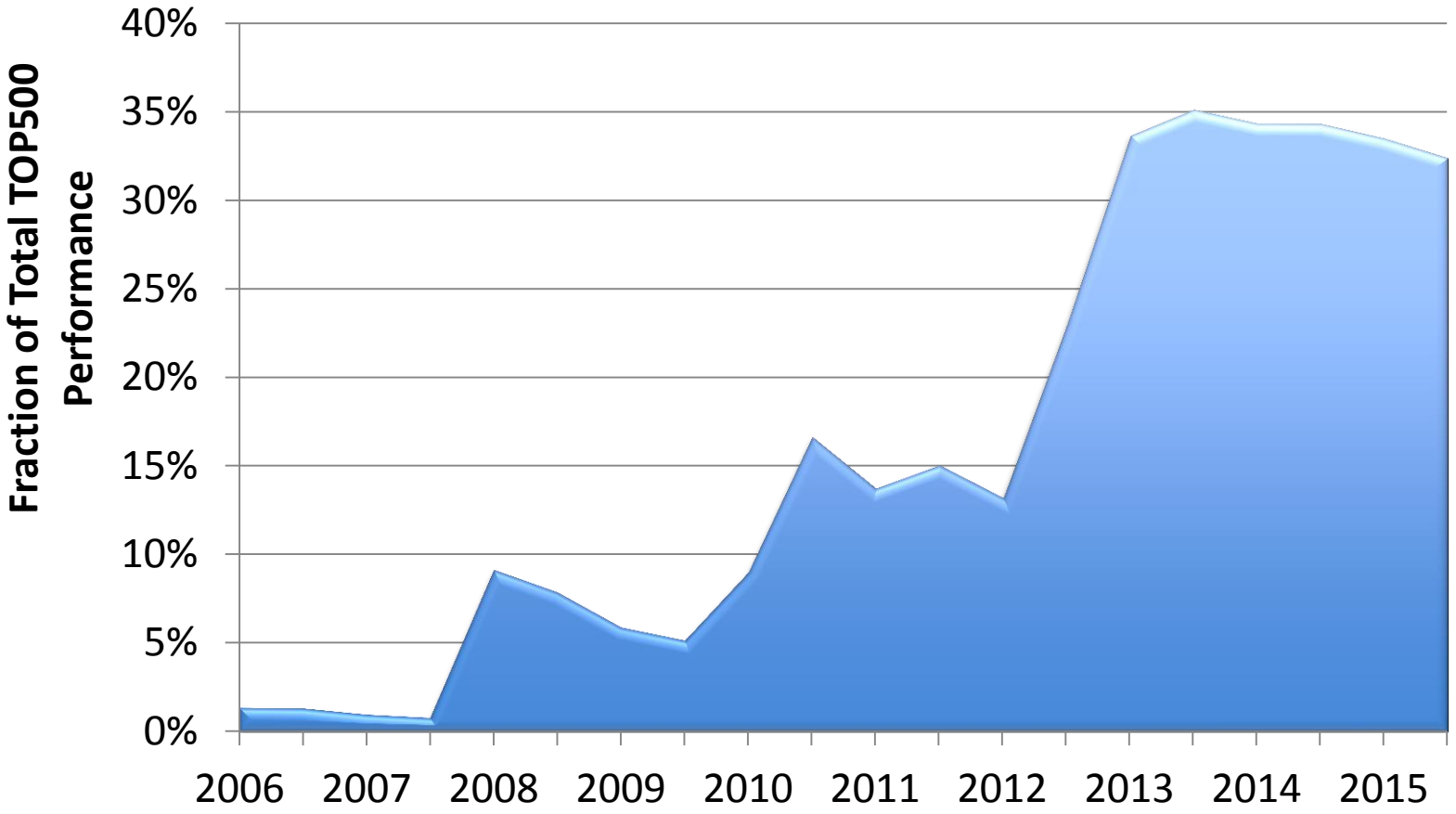
November 2015: The TOP 10 Systems

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	MFlops /Watt
1	National Super Computer Center in Guangzhou	Tianhe-2 NUDT, Xeon 12C + IntelXeon Phi (57c) + Custom	China	3,120,000	33.9	62	17.8	1905
2	DOE / OS Oak Ridge Nat Lab	Titan, Cray XK7, AMD (16C) + Nvidia Kepler GPU (14c) + Custom	USA	560,640	17.6	65	8.3	2120
3	DOE / NNSA L Livermore Nat Lab	Sequoia, BlueGene/Q (16c) + custom	USA	1,572,864	17.2	85	7.9	2063
4	RIKEN Advanced Inst for Comp Sci	K computer Fujitsu SPARC64 VIIIfx (8c) + Custom	Japan	705,024	10.5	93	12.7	827
5	DOE / OS Argonne Nat Lab	Mira, BlueGene/Q (16c) + Custom	USA	786,432	8.16	85	3.95	2066
6	DOE / NNSA / Los Alamos & Sandia	Trinity, Cray XC40,Xeon 16C + Custom	USA	301,056	8.10	80		
7	Swiss CSCS	Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler (14c) + Custom	Swiss	115,984	6.27	81	2.3	2726
8	HLRS Stuttgart	Hazel Hen, Cray XC40, Xeon 12C+ Custom	Germany	185,088	5.64	76		
9	KAUST	Shaheen II, Cray XC40, Xeon 16C + Custom	Saudi Arabia	196,608	5.54	77	2.8	1954
10	Texas Advanced Computing Center	Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB	USA	204,900	5.17	61	4.5	1489
500 (368) Karlsruher		MEGAWARE Intel	Germany	10,800	.206	95		

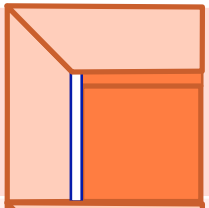
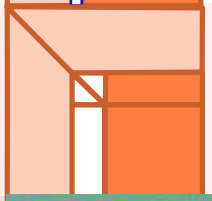
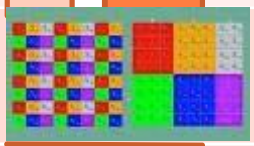
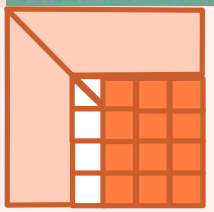
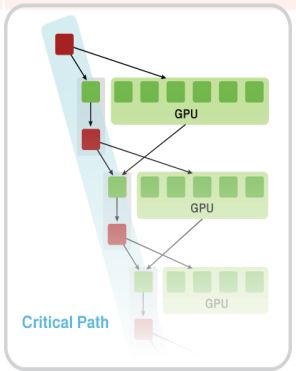
Accelerators



PERFORMANCE SHARE OF ACCELERATORS



Next Generation of DLA Software

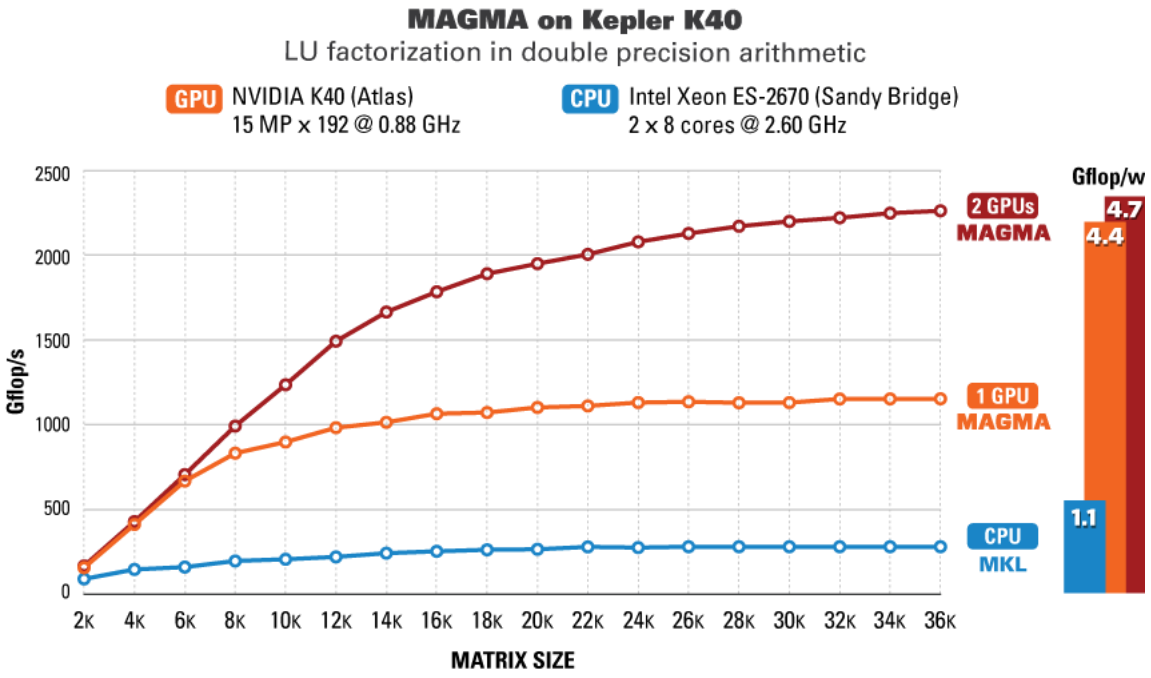
Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (00's) New Algorithms (many-core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

Key Features of MAGMA 1.7

HYBRID ALGORITHMS

MAGMA uses hybrid algorithms where the computation is split into tasks of varying granularity and their execution scheduled over the 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 MICs.

PERFORMANCE & ENERGY EFFICIENCY

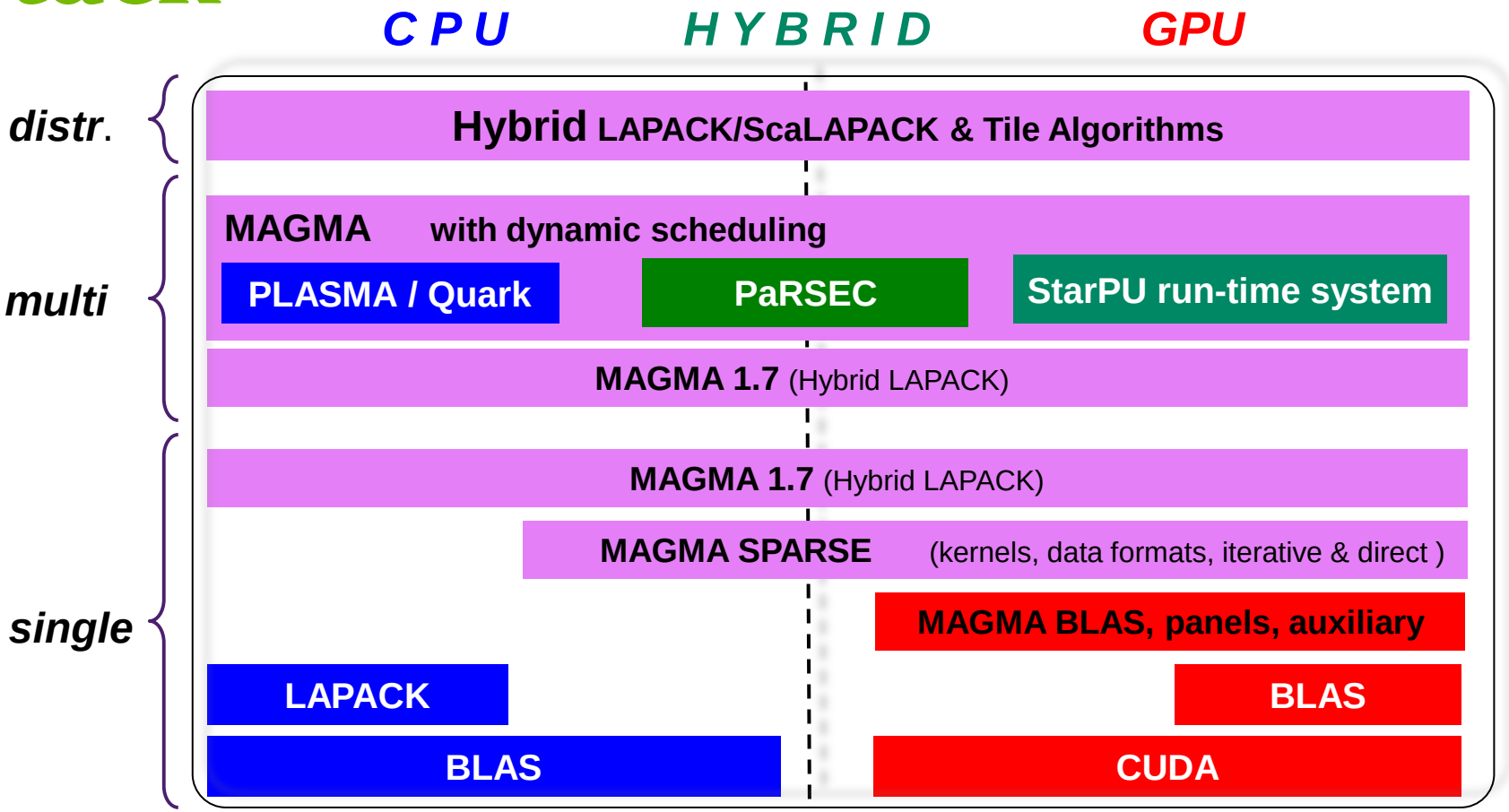


FEATURES AND SUPPORT

- **MAGMA 1.7** FOR **CUDA**
- **clMAGMA 1.3** FOR **OpenCL**
- **MAGMA MIC 1.4** FOR **Intel Xeon Phi**

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

MAGMA Libraries & Software Stack



Support: *Linux, Windows, Mac OS X*; *C/C++, Fortran*; *Matlab, Python*

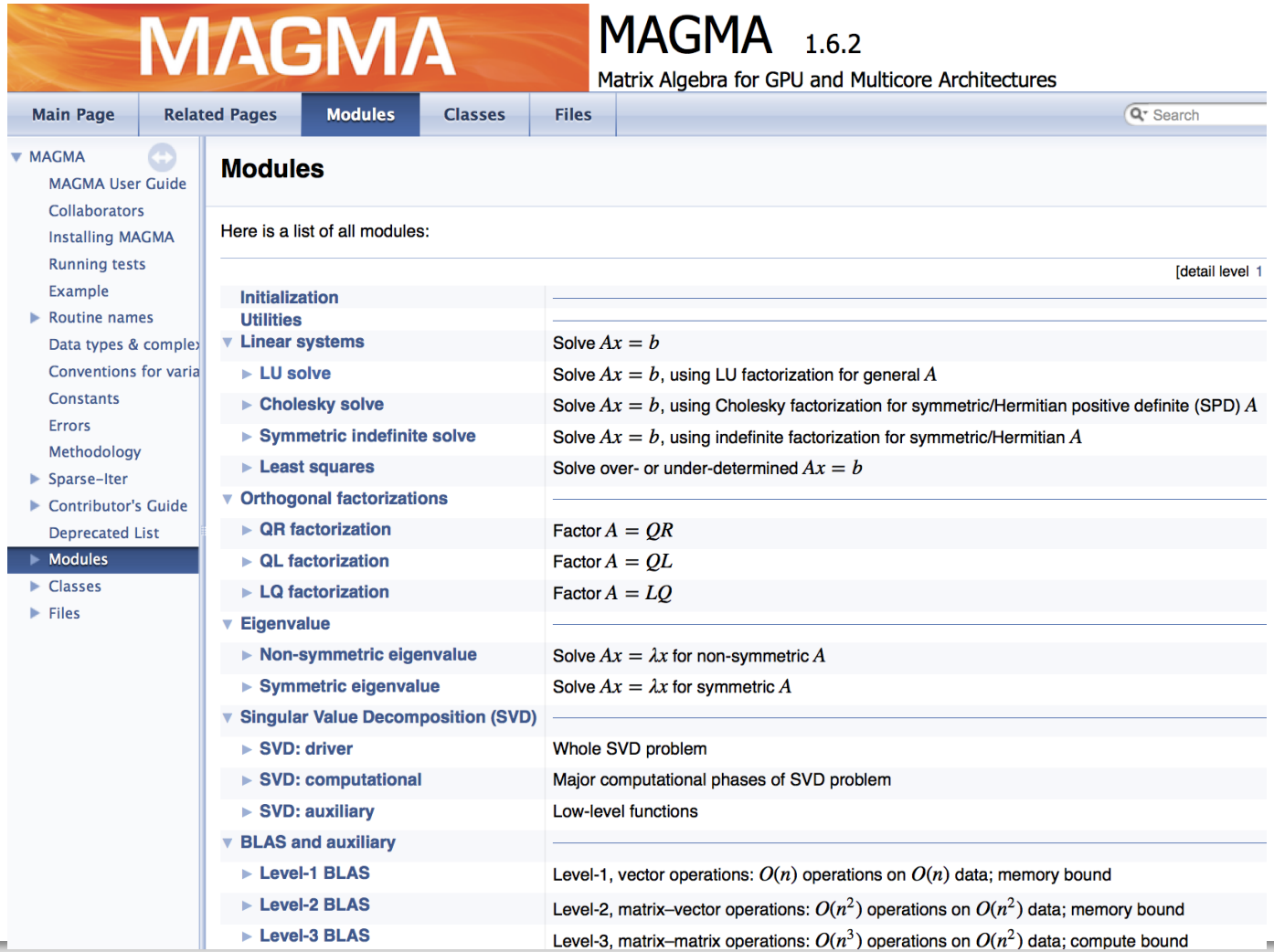
Using MAGMA

- Support is provided through the MAGMA user forum
  http://icl.cs.utk.edu/magma/forum/viewforum.php?f=2

NaN errors with dpotrf and dpotrf_gpu by fletchjp » Tue Dec 28, 2010 7:08 pm	9	3095	by fletchjp Thu Sep 12, 2013 12:03 pm
GMRES on magma by nitinb60 » Sun Aug 12, 2012 8:15 pm	4	1064	by fletchjp Thu Sep 12, 2013 11:02 am
Help please with choice of forums by fletchjp » Sat Apr 13, 2013 2:44 pm	4	1628	by fletchjp Thu Sep 12, 2013 10:42 am
the error when I compile magma1.4.0 in vs2010 by Eva Joo » Mon Sep 09, 2013 4:57 am	2	62	by Eva Joo Thu Sep 12, 2013 4:54 am
Undefined reference to cuda functions within libmagma by Matt Phillips » Mon Sep 09, 2013 10:40 pm	1	63	by Matt Phillips Mon Sep 09, 2013 11:17 pm
crash testing strsv, using OpenBLAS by Matt Phillips » Thu Sep 05, 2013 9:15 pm	0	101	by Matt Phillips Thu Sep 05, 2013 9:15 pm
MAGMA Installation: CLAPACK reference BLAS problem by psrivas2 » Tue Sep 03, 2013 4:07 am	0	84	by psrivas2 Tue Sep 03, 2013 4:07 am
Running MAGMA across several GPUs on several nodes by hsahasra » Tue Aug 13, 2013 1:32 pm	1	310	by mgates3 Fri Aug 30, 2013 3:48 pm
magma_init/finalize missing in fortran interface by stachon » Wed Aug 28, 2013 4:02 am	1	116	by mgates3 Fri Aug 30, 2013 2:13 pm
Error: BLAS/LAPACK routine 'magma_' gave error code -7 by christianHEL » Thu Aug 22, 2013 2:37 pm	2	197	by christianHEL Fri Aug 23, 2013 3:50 pm
Problems testing dsyevd with magma-1.4.0 by dougrabson » Thu Aug 15, 2013 5:44 am	1	173	by mgates3 Wed Aug 21, 2013 2:35 pm

Using MAGMA

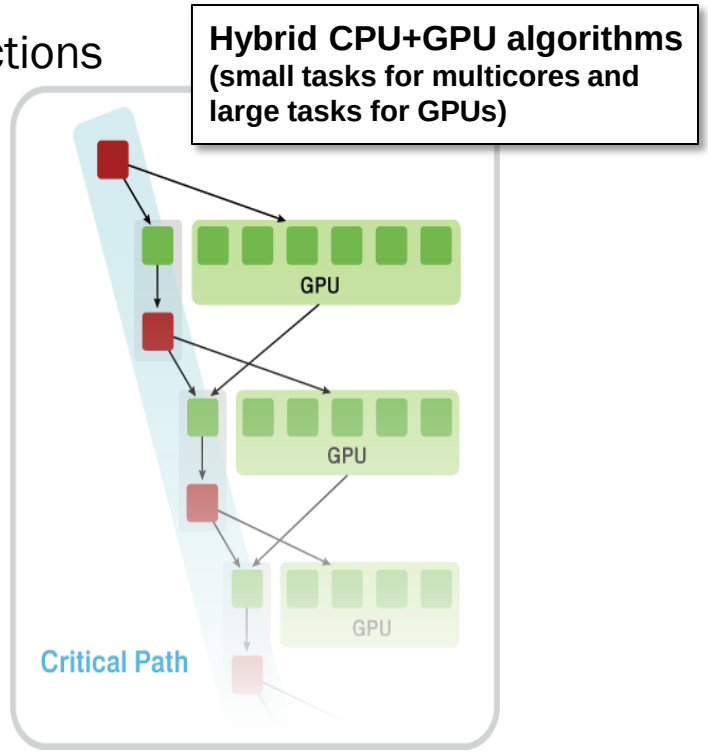
- Doxygen documentation
  http://icl.cs.utk.edu/projectfiles/magma/doxygen/



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



- 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

A Hybrid Algorithm Example

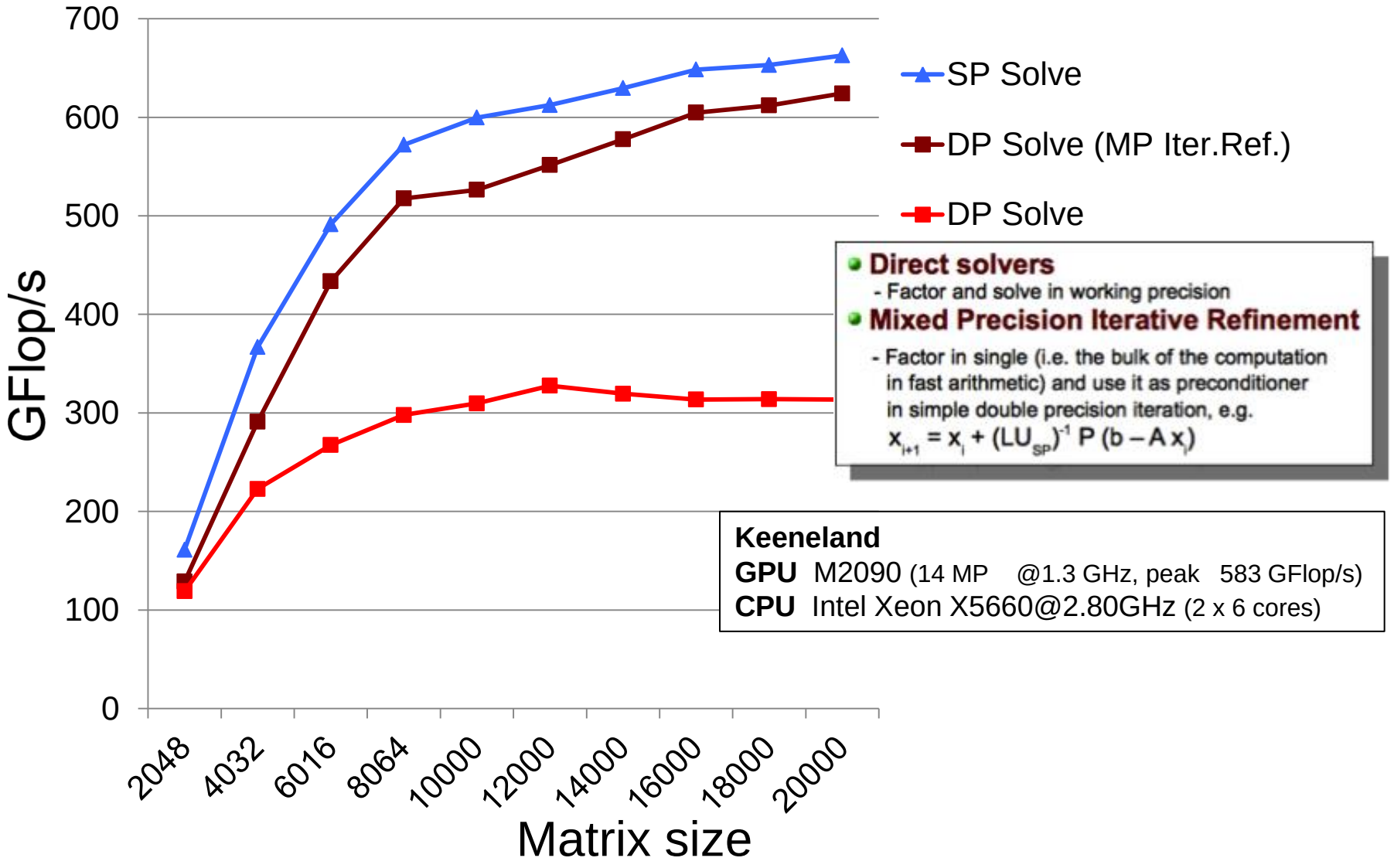
- Left-looking hybrid Cholesky factorization in MAGMA

```
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 work on the GPU (from line 6)

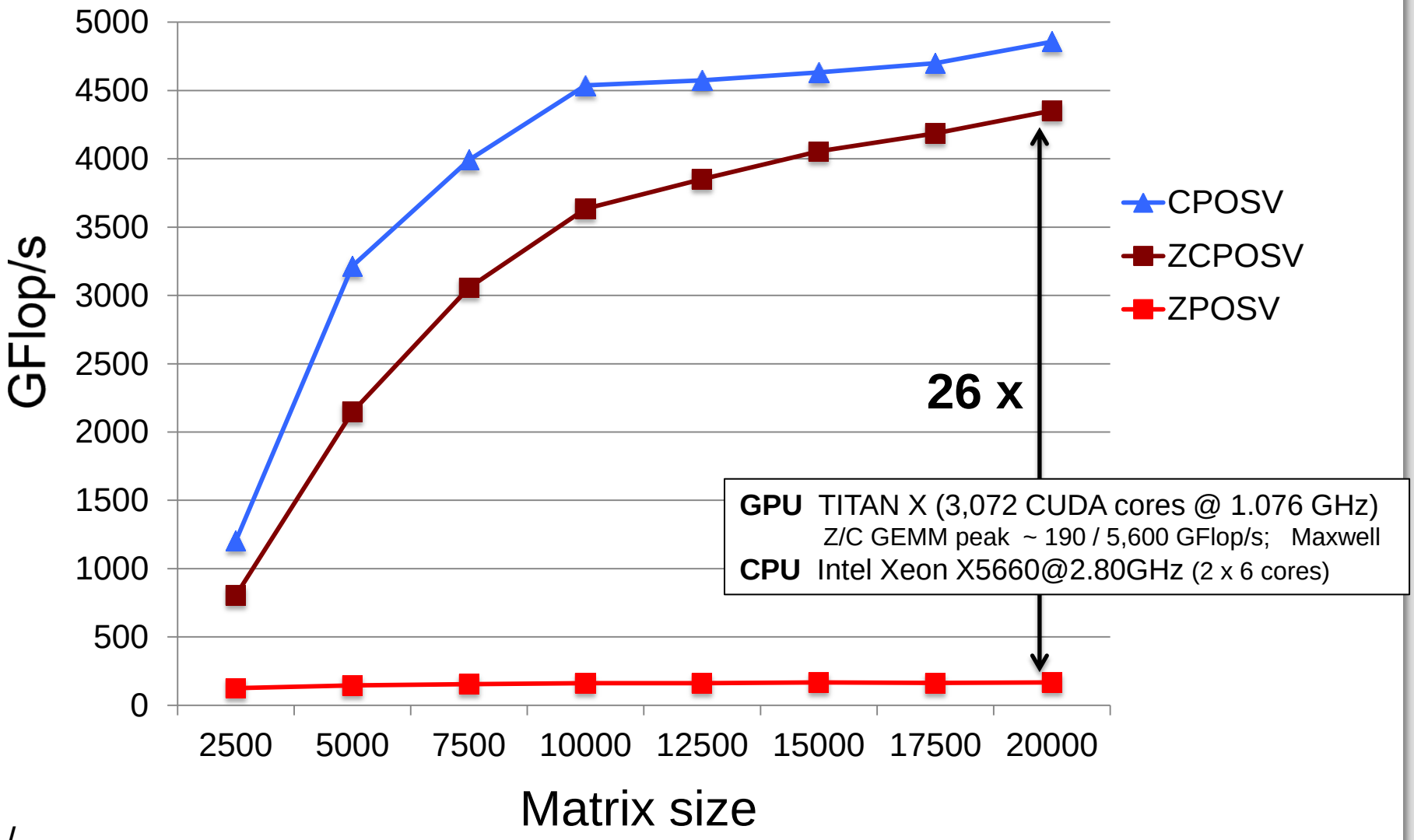
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



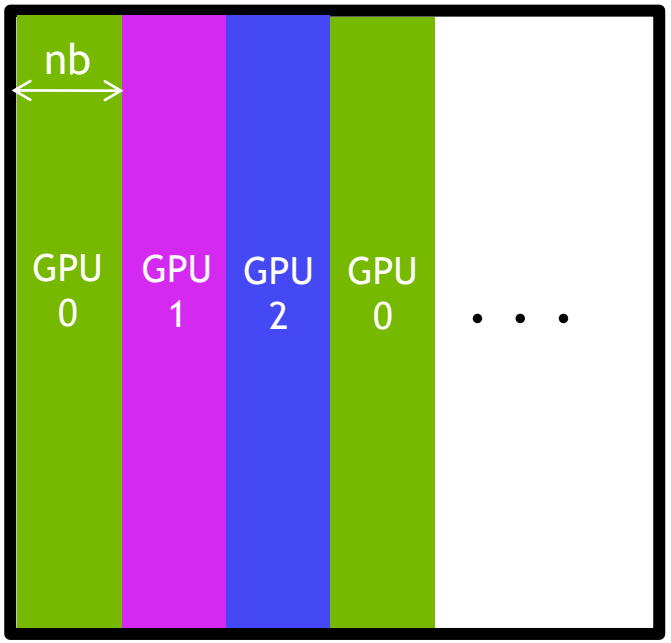
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

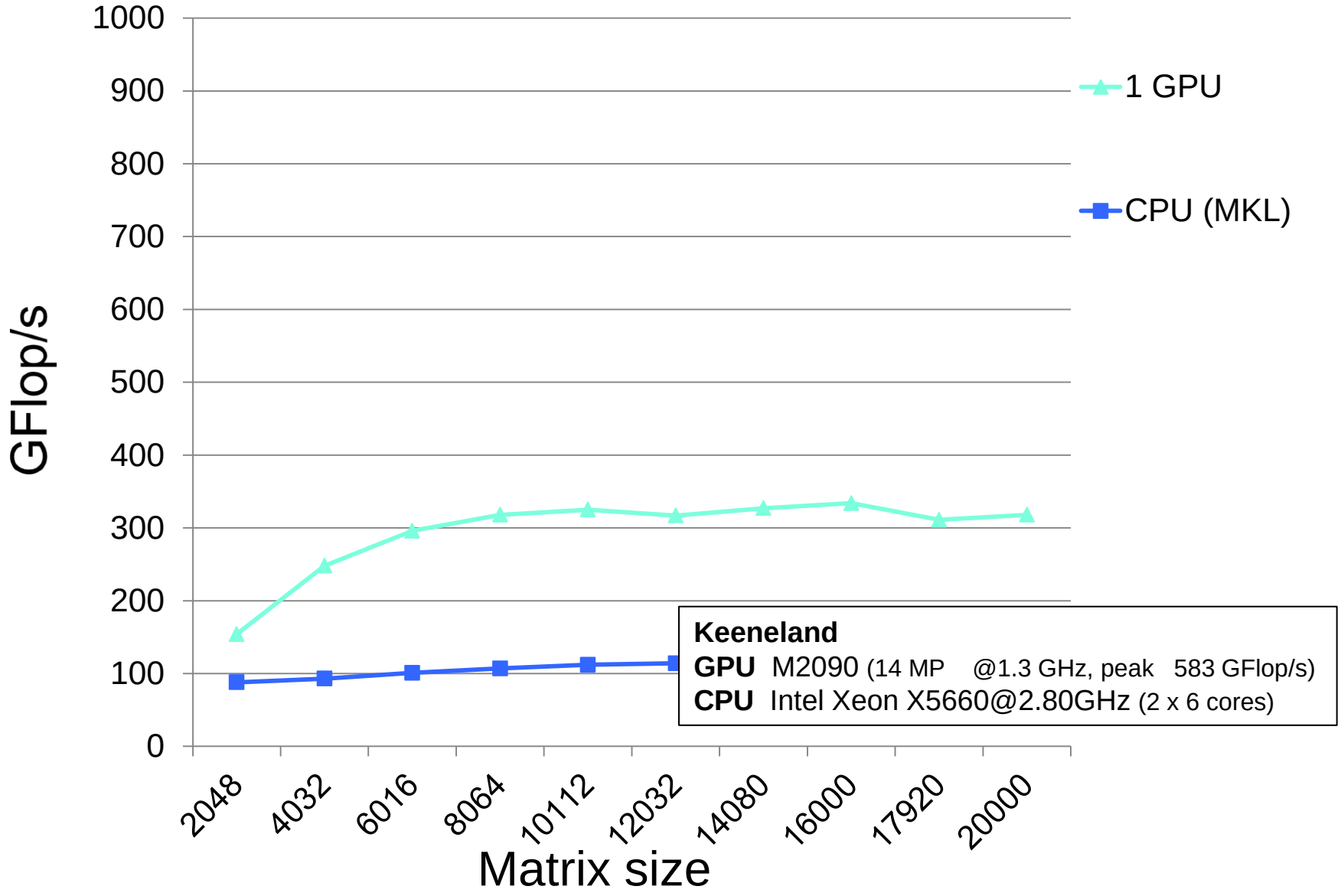


MultiGPU Support

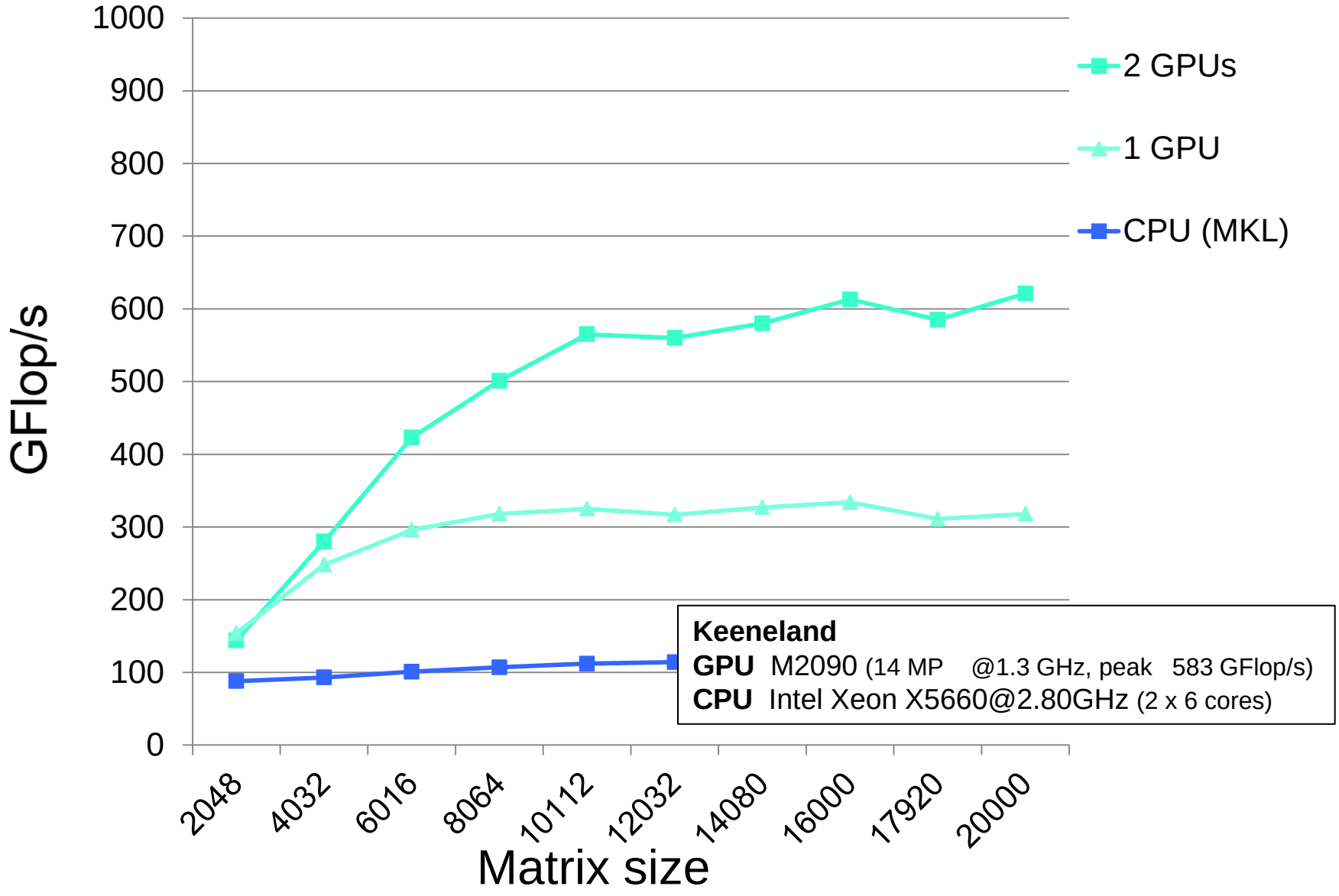
- Data distribution
  - 1-D block-cyclic distribution
- Algorithm
  - GPU holding current panel is sending it to CPU
  - All updates are done in parallel on the GPUs
  - Look-ahead is done with GPU holding the next panel



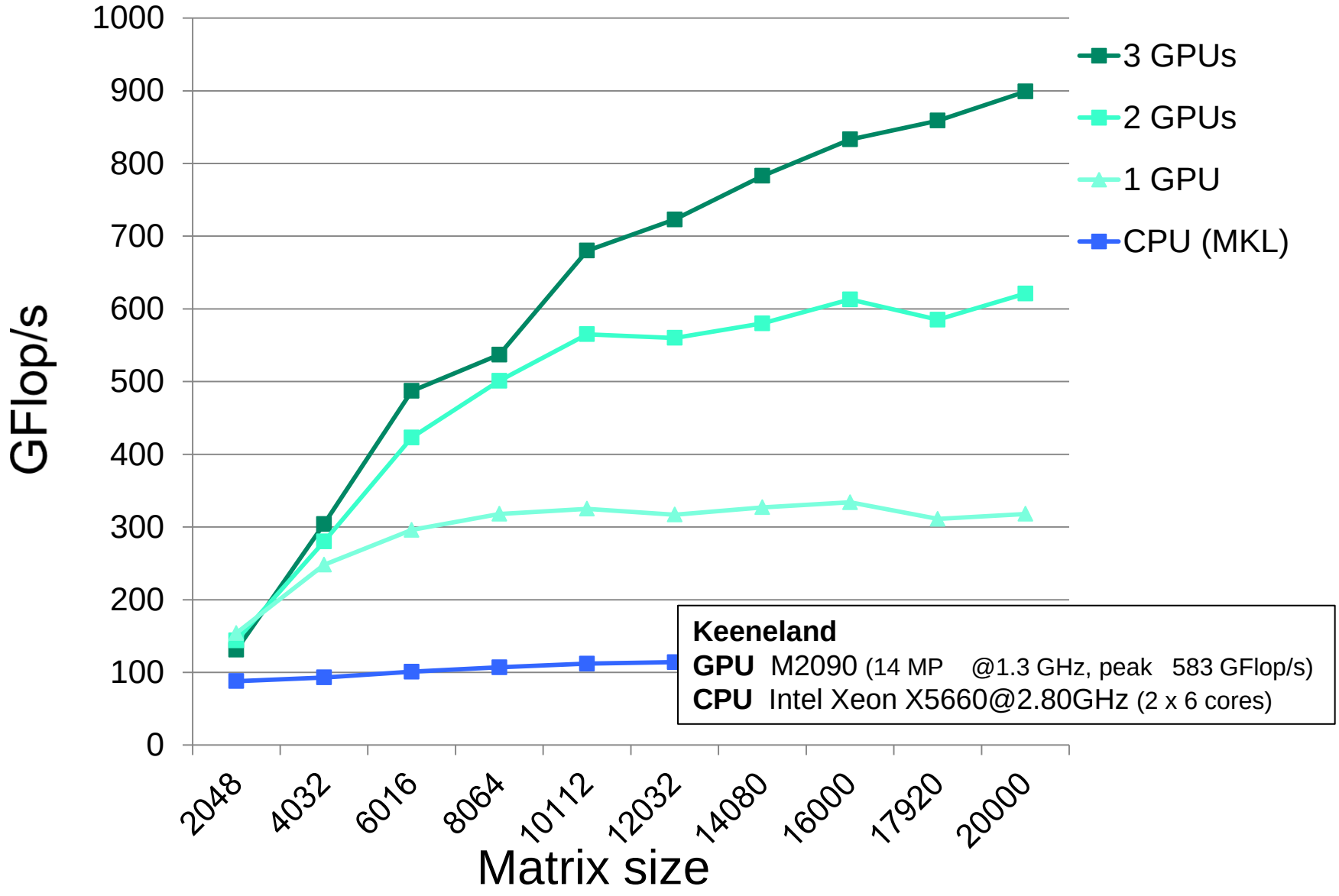
LU on multiGPUs in DP



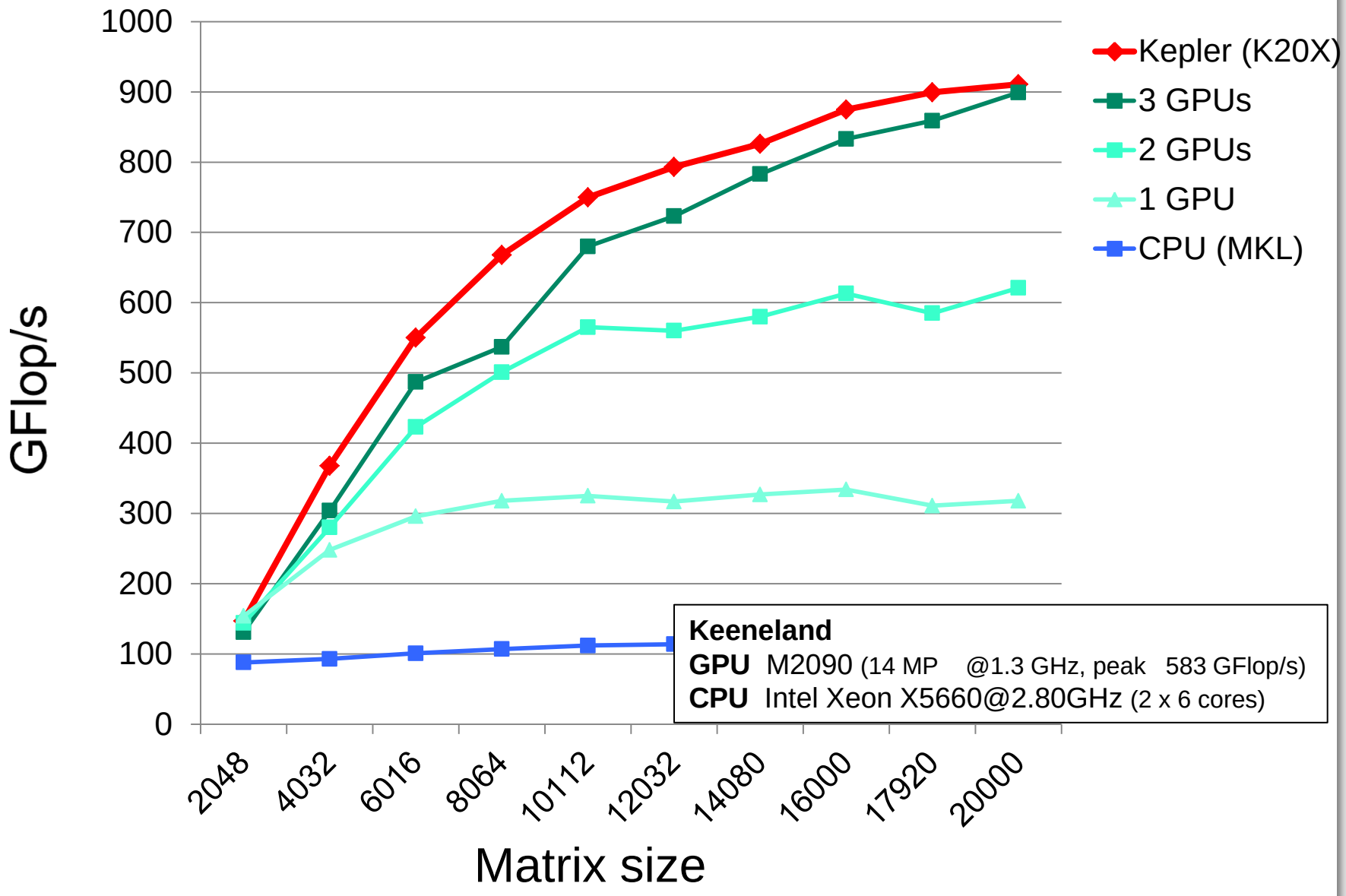
LU on multiGPUs in DP



LU on multiGPUs in DP



LU on Kepler in DP



Density Functional Theory

Many-body Schrödinger equation (exact but exponential scaling)

$$\{-\sum_i \frac{1}{2}\nabla_i^2 + \sum_{i,j} \frac{1}{|r_i - r_j|} + \sum_{i,I} \frac{Z}{|r_i - R_I|}\}\Psi(r_1,..r_N) = E\Psi(r_1,..r_N)$$

In modern electronic structure methods (**density functional theory**) this is reduced to the **Kohn-Sham equation** (single particle method with polynomial complexity)

$$\{-\frac{1}{2}\nabla^2 + V(r,\rho)\}\psi_i(r) = E_i\psi_i(r)$$

The potential *V* depends on Ψ_i, so the equation must be solved many times until convergence:

- **Introduce a basis ϕ_i for the orbitals Ψ**
- **The Hamiltonian is a Hermitian matrix**

$$H_{i,j} = \int f_i(r)\left(-\frac{1}{2}\nabla^2 + V(r,r)\right)f_j(r)\,dr$$

- **The basis may not be orthonormal**

$$S_{i,j} = \int f_i(r) f_j(r)\,dr$$

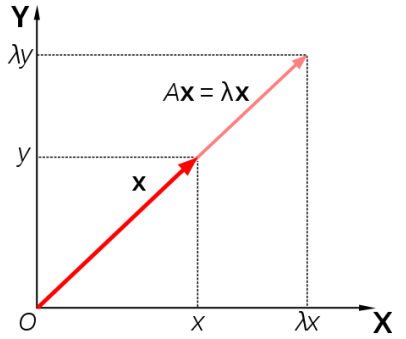
- **Solve the generalized eigenvalue problem** $H\,x = \varepsilon\,S\,x$

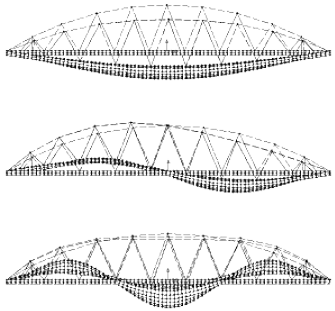
Eigenproblem Solvers in MAGMA

$A x = \lambda x$

- 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-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.

Distributed memory systems
Generalized Hermitian Definite eigensolvers

with Thomas Schulthess et al. [1], CSCS

- Generalized Hermitian definite eigenproblem of the form

$$H\,x = \varepsilon\, S\,x$$

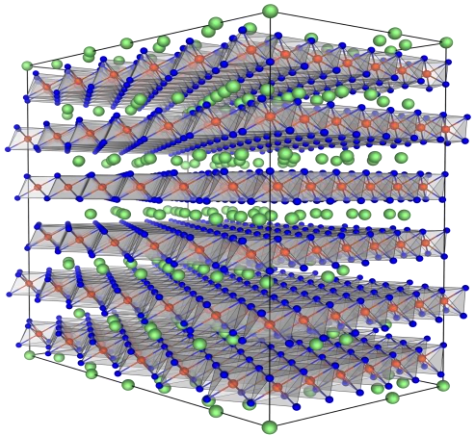
- Solution follows the following steps

1) Compute the Cholesky factorization of
$$S = L\,L^H$$
2) Transform the generalized eigenproblem to standard form
$$\hat{H}\,z = \lambda\,z, \quad \hat{H} = L^{-1}\,H\,L^{-H}$$
3) Solve the standard eigenproblem
$$\hat{H} = Z\,\Lambda\,Z^H$$
4) Back-solve the eigenvectors with the Cholesky factor
$$X = L^{-H}\,Z$$

[1] **R. Solca, A. Kozhevnikov, A. Haidar, S. Tomov, T. Schulthess, and J. Dongarra,** *Efficient implementation of quantum material simulations on distributed CPU-GPU systems.* SC'15, Best Paper Award finalist, Austin, TX, November 15-20, 2015.

Distributed memory systems

Generalized Hermitian Definite eigensolvers: performance results



	Setup H20	Solve HC=40C	The rest	total
392 CPU nodes ScaLAPACK	463.7	3839.7	61.6	4365.0
392 CPU nodes ELPA	471.7	1199.3	60.7	1731.7
192 CPU+GPU nodes MAGMA	166.7	911.9	79.9	1158.5

MAGMA is **1.5 times faster** than CPU

MAGMA is **2 times more energy efficient**

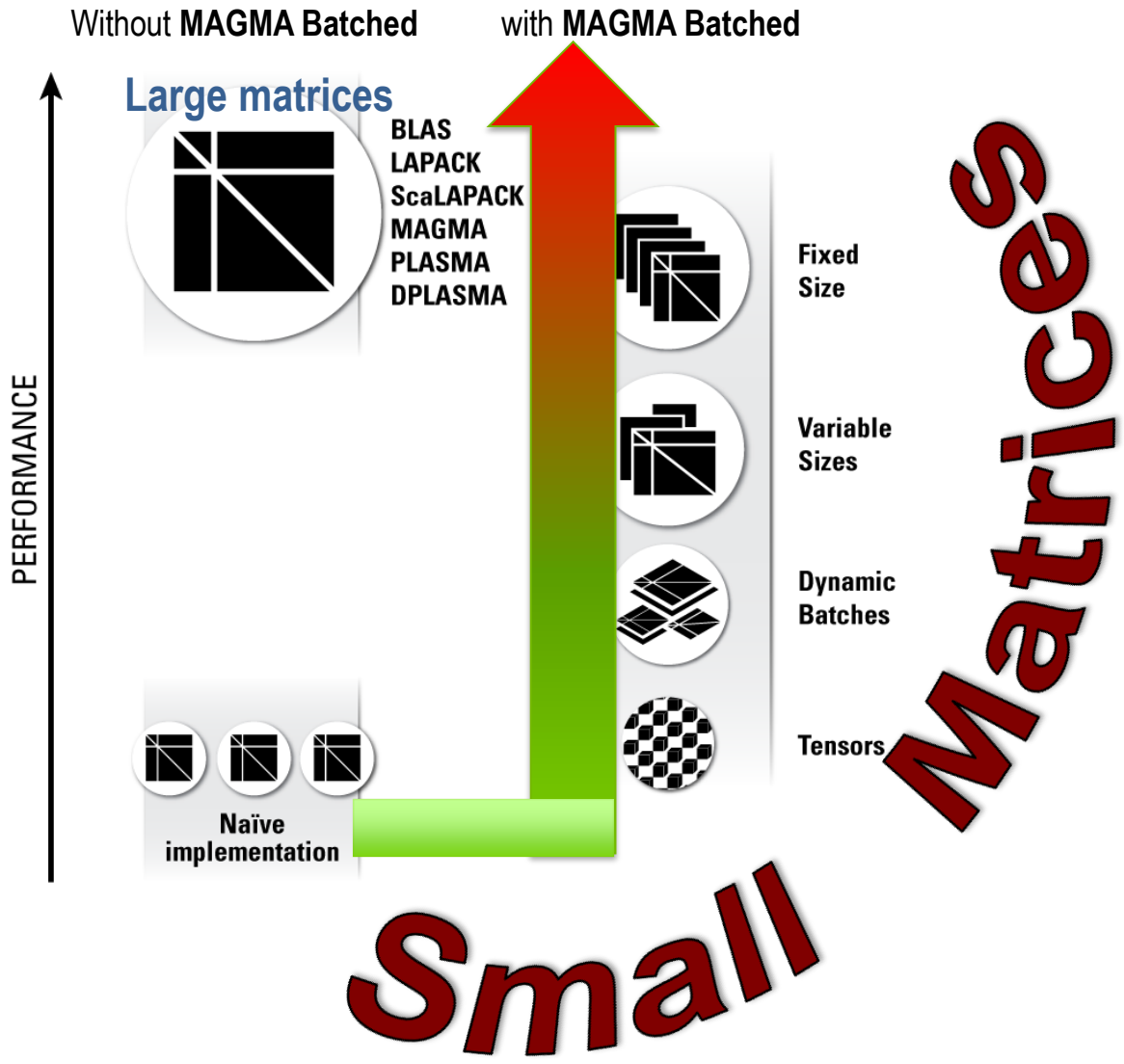
Piz Daint
Cray XC30:
- 8-core Intel Xeon E5-2670 Sandy Bridge socket
- Nvidia K20X

- **R. Solca, A. Kozhevnikov, A. Haidar, S. Tomov, T. Schulthess, and J. Dongarra,** *Efficient implementation of quantum material simulations on distributed CPU-GPU systems.* SC'15, Best Paper Award finalist, Austin, TX, November 15-20, 2015.

- **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,
International Journal of High Performance Computing Applications , vol. 28, no 2, pp. 196—209, May 2014.

- **A. Haidar, R. Solca, S. Tomov, T. Schulthess and J. Dongarra.** *Leading edge multi-GPU algorithms for generalized eigenproblems for electronic structure calculations.* International Supercomputing Conference IEEE-ISC 2013.

Motivation

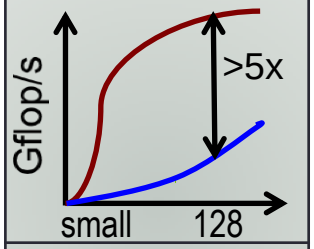
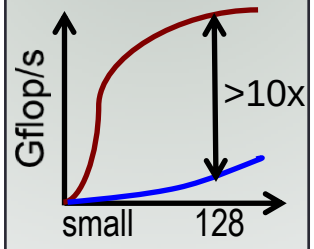


Linear Algebra on small problems are needed in many applications:

- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, and more

Motivation ...

Batched *vs.* standard LA techniques

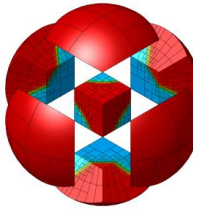
LA problems \ Techniques	Batched (for small problems)	Standard (for large problems)	Expected acceleration ranges
Basic Linear Algebra Subprograms (BLAS)	**Batched BLAS** (no scheduling overheads)	Vendor optimized BLAS (e.g., CUBLAS, Intel MKL)	
Advanced routines: • Linear system solvers • Eigensolvers & SVD	• **Built on Batched BLAS** • **GPU-only** (no comm.) • **Batch-aware algorithms** • **Batch-scheduled**	• Built on BLAS • Hybrid CPU + GPU • High-level algorithms • DAG scheduling	

Examples

Need of **Tensor contractions** for **FEM simulations**

[collaboration with LLNL on BLAST package and Inria, France]

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

$$\textbf{Momentum Conservation:} \quad \frac{d\mathbf{v}}{dt} = -\mathbf{M_v^{-1}F} \cdot \mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{d\mathbf{e}}{dt} = \mathbf{M_e^{-1}F^T} \cdot \mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{d\mathbf{x}}{dt} = \mathbf{v}$$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M_v}$ and $\mathbf{M_e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ that needs to be evaluated at every time step.

[1] V. Dobrev, T.Kolev, R.Rieben. *High order curvilinear finite element methods for Lagrangian hydrodynamics*. SIAM J.Sci.Comp.34(5), B606–B641. (36 pages)

- Contractions can often be implemented as index reordering plus **batched GEMM** (and hence, be highly efficient)

TENSOR KERNELS FOR ASSEMBLY/EVALUATION 3

stored components	FLOPs for assembly	amount of storage	FLOPs for matvec	numerical kernels
full assembly				
M	$O(p^{3d})$	$O(p^{2d})$	$O(p^{2d})$	$B, D \mapsto B^T D B, x \mapsto Mx$
decomposed evaluation				
B, D	$O(p^{2d})$	$O(p^{2d})$	$O(p^{2d})$	$x \mapsto Bx, x \mapsto B^T x, x \mapsto Dx$
near-optimal assembly – equations (1) and (2)				
$M_{i_1,\cdots,j_d}$	$O(p^{2d+1})$	$O(p^{2d})$	$O(p^{2d})$	$A_{i_1,k_2,j_1} = \sum_{k_1} B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1} D_{k_1,k_2}$ (1a)
				$A_{i_1,i_2,j_1,j_2} = \sum_{k_2} B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2} C_{i_1,k_2,j_1}$ (1b)
				$A_{i_1,k_2,k_3,j_1} = \sum_{k_1} B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1} D_{k_1,k_2,k_3}$ (2a)
				$A_{i_1,i_2,k_3,j_1,j_2} = \sum_{k_2} B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2} C_{i_1,k_2,k_3,j_1}$ (2b)
				$A_{i_1,i_2,i_3,j_1,j_2,j_3} = \sum_{k_3} B^{1d}_{k_3,i_3} B^{1d}_{k_3,j_3} C_{i_1,i_2,k_3,j_1,j_2}$ (2c)
near-optimal evaluation (partial assembly) – equations (3) and (4)				
B^{1d}, D	$O(p^d)$	$O(p^d)$	$O(p^{d+1})$	$A_{j_1,k_2} = \sum_{j_2} B^{1d}_{k_2,j_2} V_{j_1,j_2}$ (3a)
				$A_{k_1,k_2} = \sum_{j_1} B^{1d}_{k_1,j_1} C_{j_1,k_2}$ (3b)
				$A_{k_1,i_2} = \sum_{k_2} B^{1d}_{k_2,i_2} C_{k_1,k_2}$ (3c)
				$A_{i_1,i_2} = \sum_{k_1} B^{1d}_{k_1,i_1} C_{k_1,i_2}$ (3d)
				$A_{j_1,j_2,k_3} = \sum_{j_3} B^{1d}_{k_3,j_3} V_{j_1,j_2,j_3}$ (4a)
				$A_{j_1,k_2,k_3} = \sum_{j_2} B^{1d}_{k_2,j_2} C_{j_1,j_2,k_3}$ (4b)
				$A_{k_1,k_2,k_3} = \sum_{j_1} B^{1d}_{k_1,j_1} C_{j_1,k_2,k_3}$ (4c)
				$A_{k_1,k_2,i_3} = \sum_{k_3} B^{1d}_{k_3,i_3} C_{k_1,k_2,k_3}$ (4d)
				$A_{k_1,i_2,i_3} = \sum_{k_2} B^{1d}_{k_2,i_2} C_{k_1,k_2,i_3}$ (4e)
				$A_{i_1,i_2,i_3} = \sum_{k_1} B^{1d}_{k_1,i_1} C_{k_1,i_2,i_3}$ (4f)
matrix-free evaluation				
none	none	none	$O(p^{d+1})$	evaluating entries of B^{1d}, D, (3a)–(4f) sums

Examples

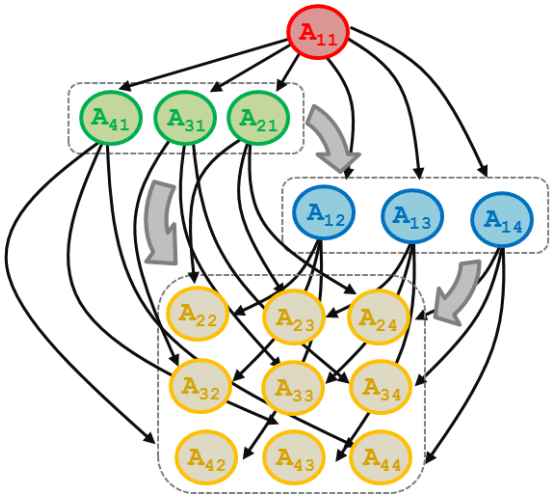
Need of **Batched** routines for **Numerical LA**

[e.g., sparse direct multifrontal methods, preconditioners for sparse iterative methods, tiled algorithms in dense linear algebra, etc.;]

Sparse / Dense Matrix System

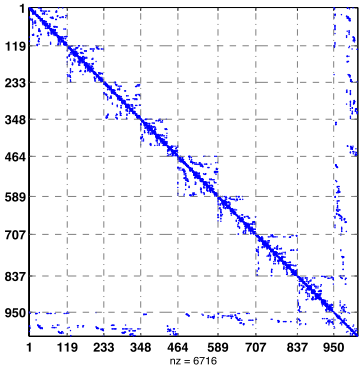
$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

DAG-based factorization



To capture main LA patterns needed in a numerical library for Batched LA

- **LU, QR,** or **Cholesky** on small diagonal matrices
- **TRSM**s, **QR**s, or **LU**s
- **TRSM**s, **TRMM**s
- **Updates (Schur complement) GEMM**s, **SYRK**s, **TRMM**s

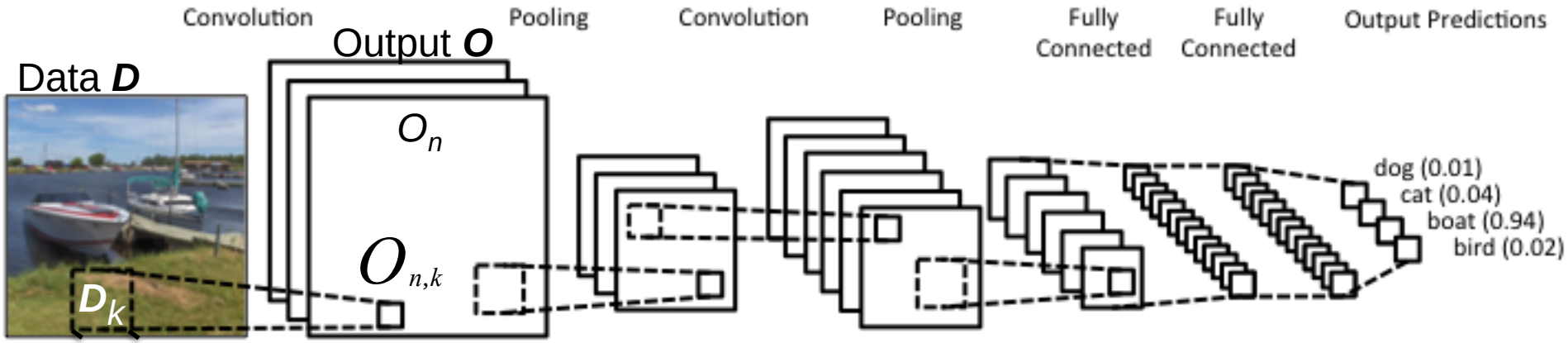


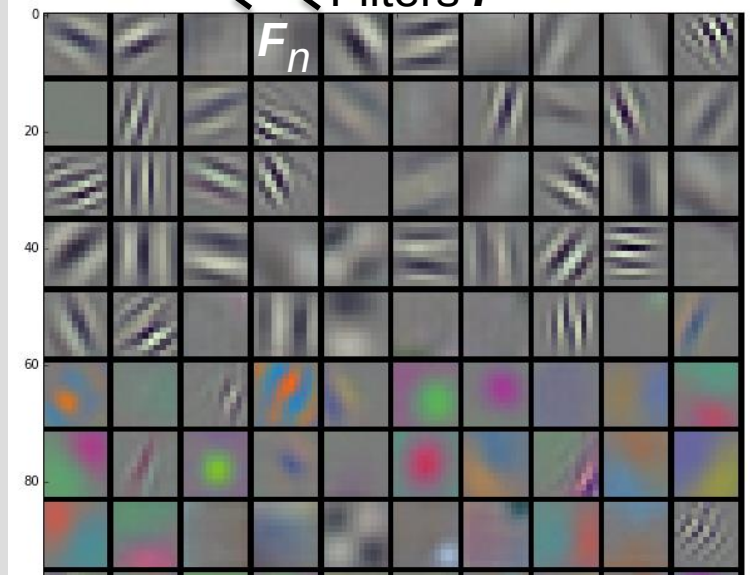
- Example matrix from Quantum chromodynamics
- Reordered and ready for sparse direct multifrontal solver
- Diagonal blocks can be handled in parallel through batched LU, QR, or Cholesky factorizations

Examples

Need of **Batched and/or Tensor contraction** routines in **machine learning**

e.g., Convolutional Neural Networks (CNNs) used in computer vision
Key computation is convolution of Filter Fi (feature detector) and input image D (data):





Convolution operation:

- For every filter F_n and every channel, the computation for every pixel value $O_{n,k}$ is a **tensor contraction**:

$$O_{n,k} = \sum_i D_{k,i} F_{n,i}$$

- Plenty of parallelism; small operations that must be batched
- With data "reshape" the computation can be transformed into a **batched GEMM** (and hence, efficiently implemented; among other approaches)

Examples

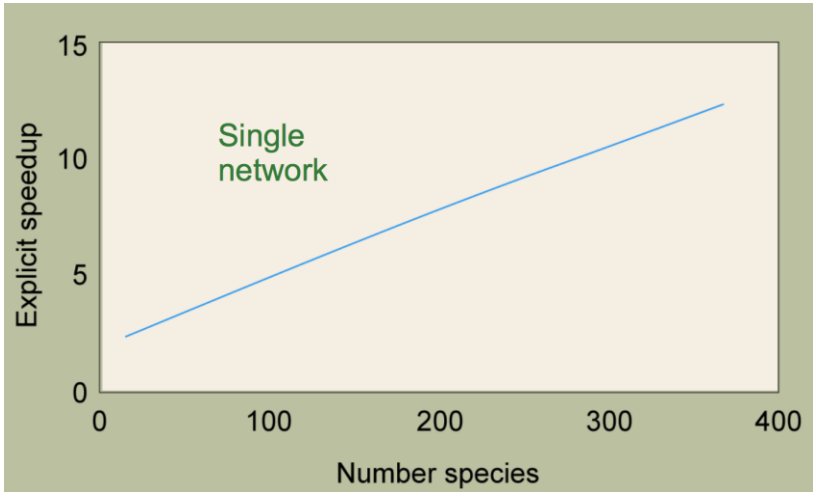
Multi-physics problems need **Batched LA on small problems**

Collaboration with ORNL and UTK physics department (Mike Guidry, Jay Billings, Ben Brock, Daniel Shyles, Andrew Belt)

- Many physical systems can be modeled by a fluid dynamics plus kinetic approximation
  e.g., in astrophysics, stiff equations must be integrated numerically:
  - **Implicitly**; standard approach, leading to need of batched solvers (e.g., as in XNet library)
  - **Explicitly**; a new way to stabilize them with Macro- plus Microscopic equilibration

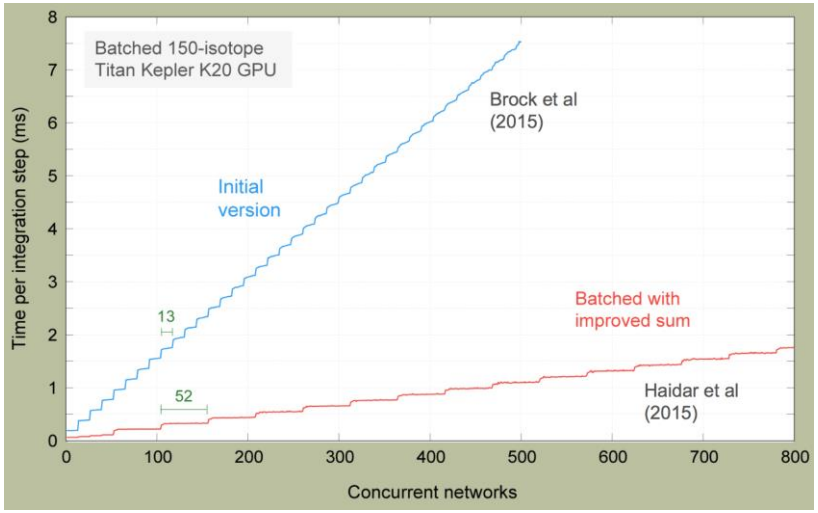
    need batched tensor contractions of variable sizes

Explicit vs. Implicit speedup on single network



10x speedup on few hundred species
(few hundred dof batched solve in implicit methods)

Additional acceleration achieved through MAGMA Batched



An additional **7x speedup** over initially highly optimized explicit method implementation

Collaborators and Support

MAGMA team

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

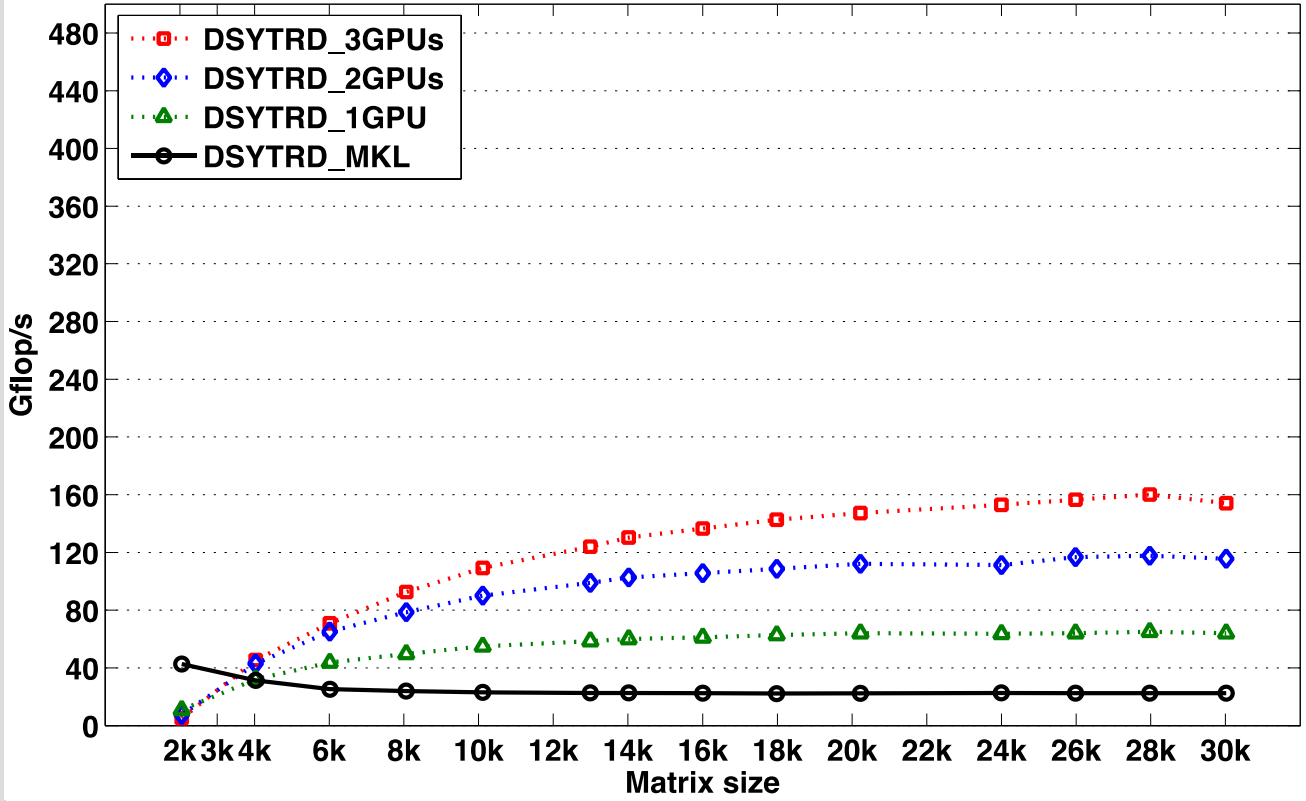
PLASMA team

http://icl.cs.utk.edu/plasma

Collaborating partners

University of Tennessee, Knoxville
University of California, Berkeley
University of Colorado, Denver
INRIA, France (StarPU team)
KAUST, Saudi Arabia

Toward fast Eigensolvers



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

Higher is faster

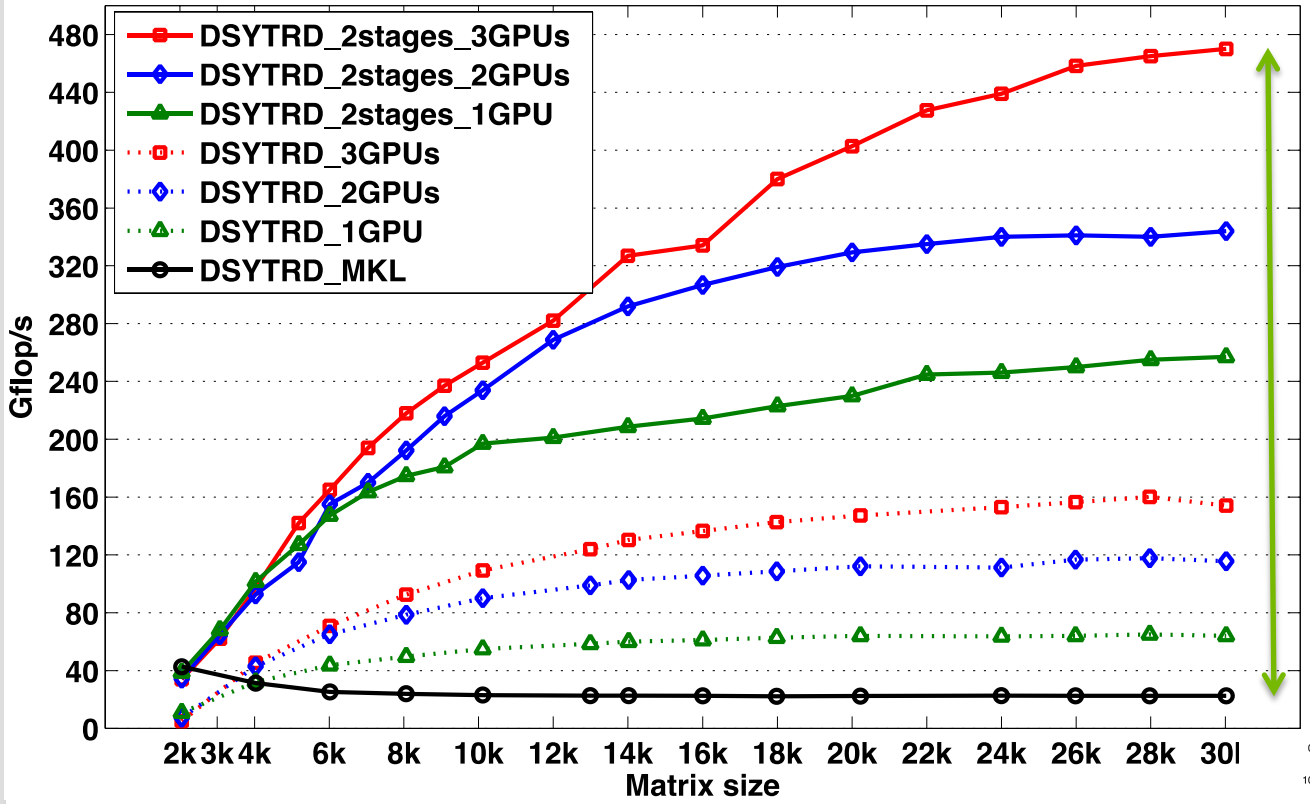
Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

Characteristics

- Blas-2 GEMV moved to the GPU,
- Accelerate the algorithm by doing all BLAS-3 on GPU,
- ➔**Bulk sync 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: $n^3/3*time$

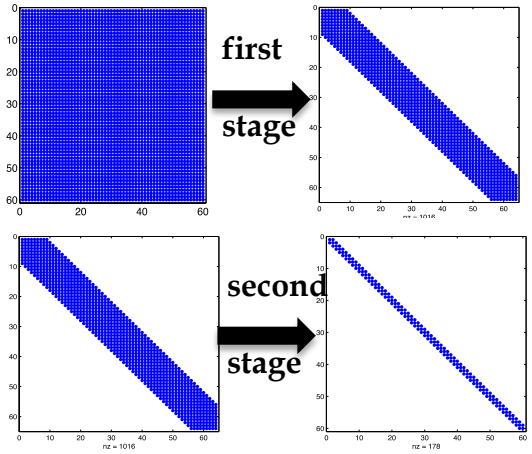
Higher is faster

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:
15 X 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.

Current work

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

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, ...);
    }
}
```

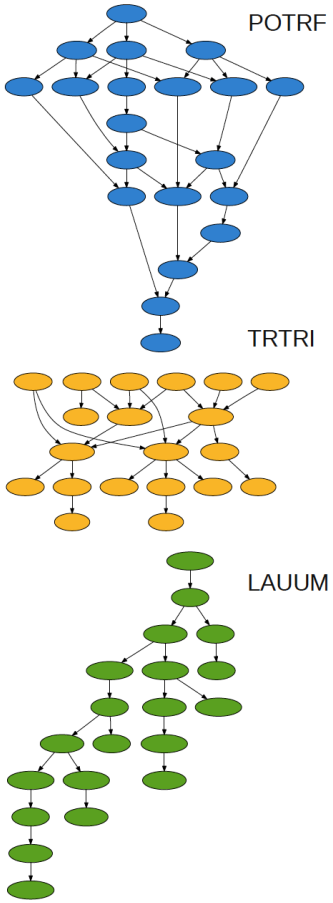
Various runtime systems can be used:

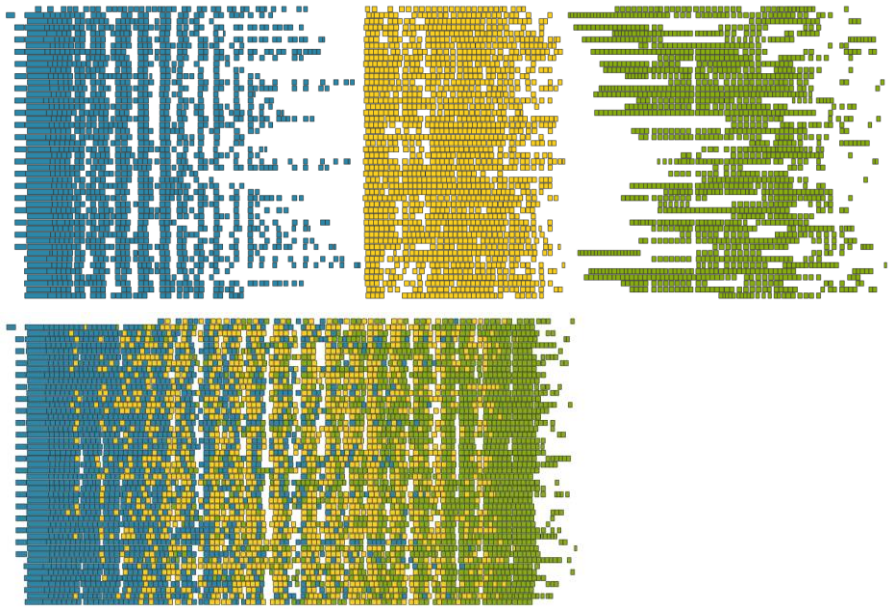
- **StarPU** http://icl.cs.utk.edu/projectsdev/morse
- **PaRSEC** https://icl.cs.utk.edu/parsec/
- **QUARK** http://icl.cs.utk.edu/quark/

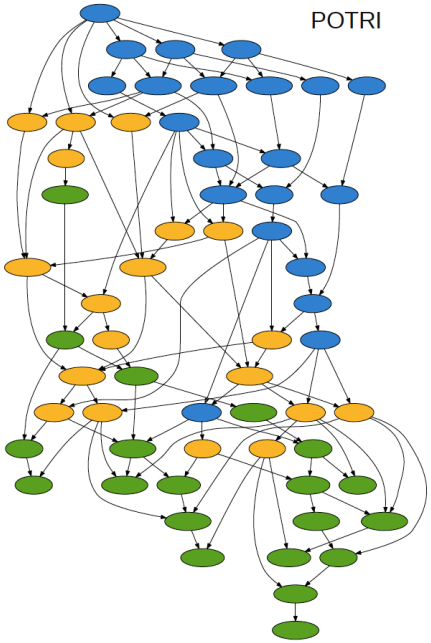
Current work

- Schedule task execution using

Dynamic Runtime Systems

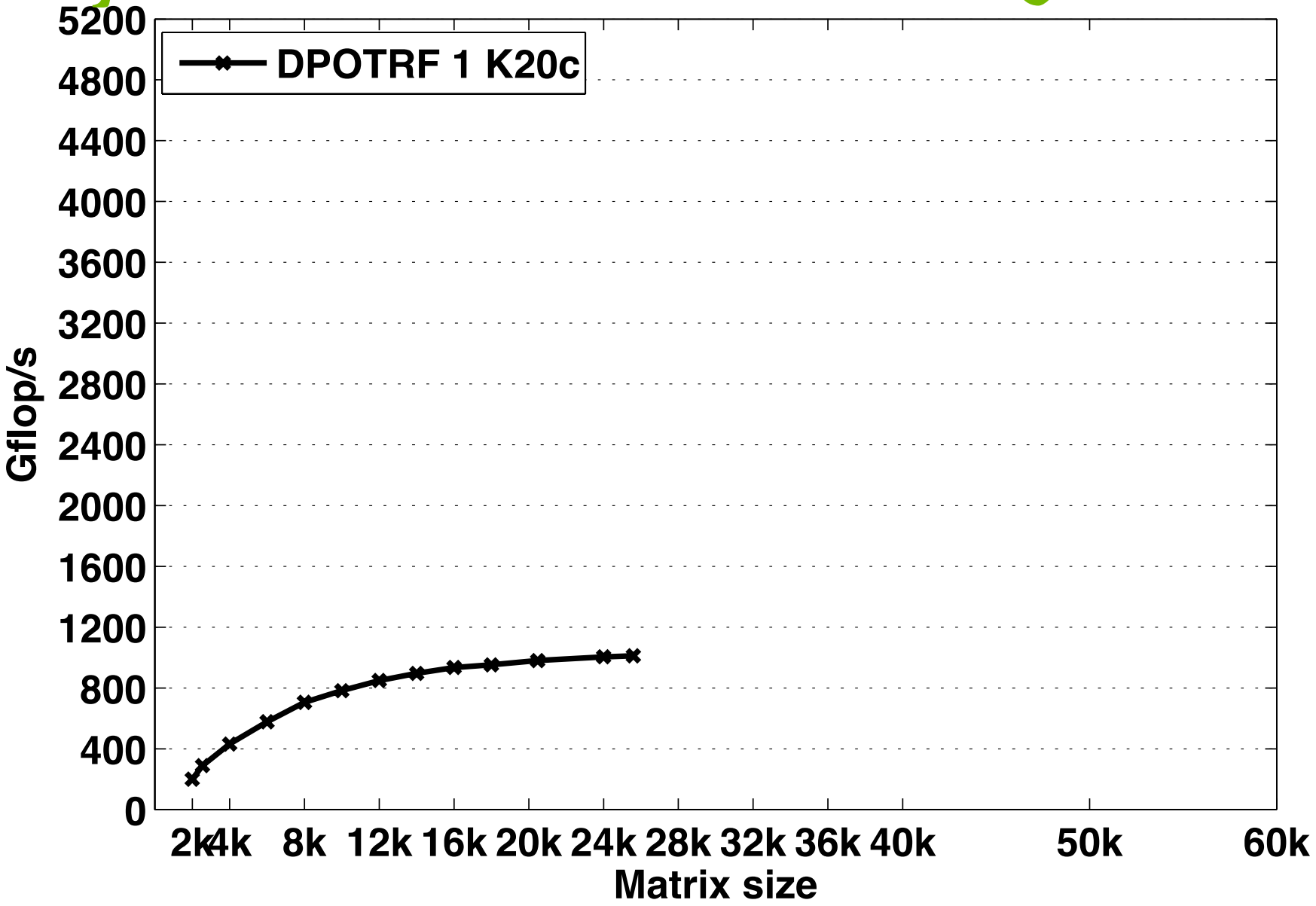




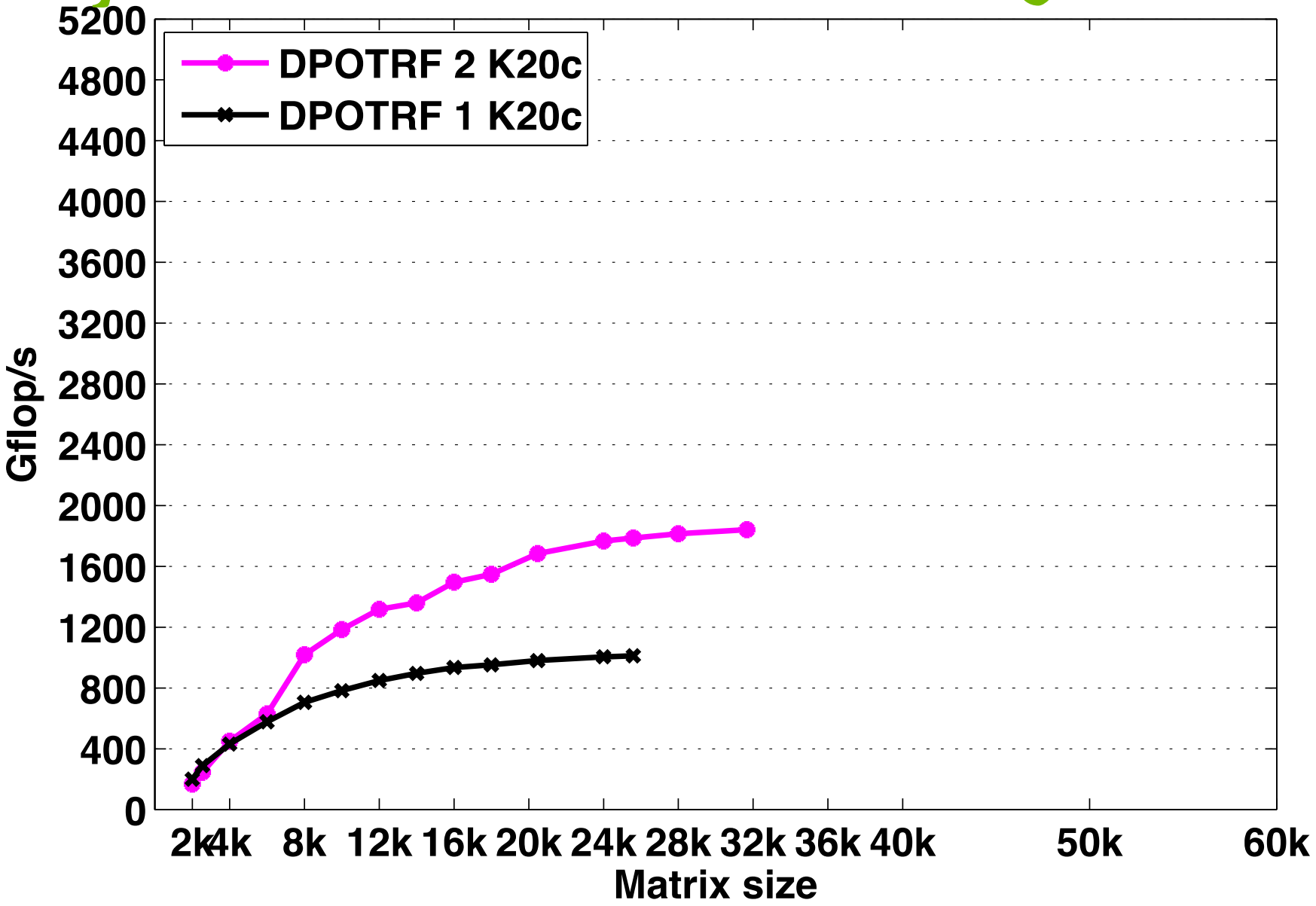


48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200

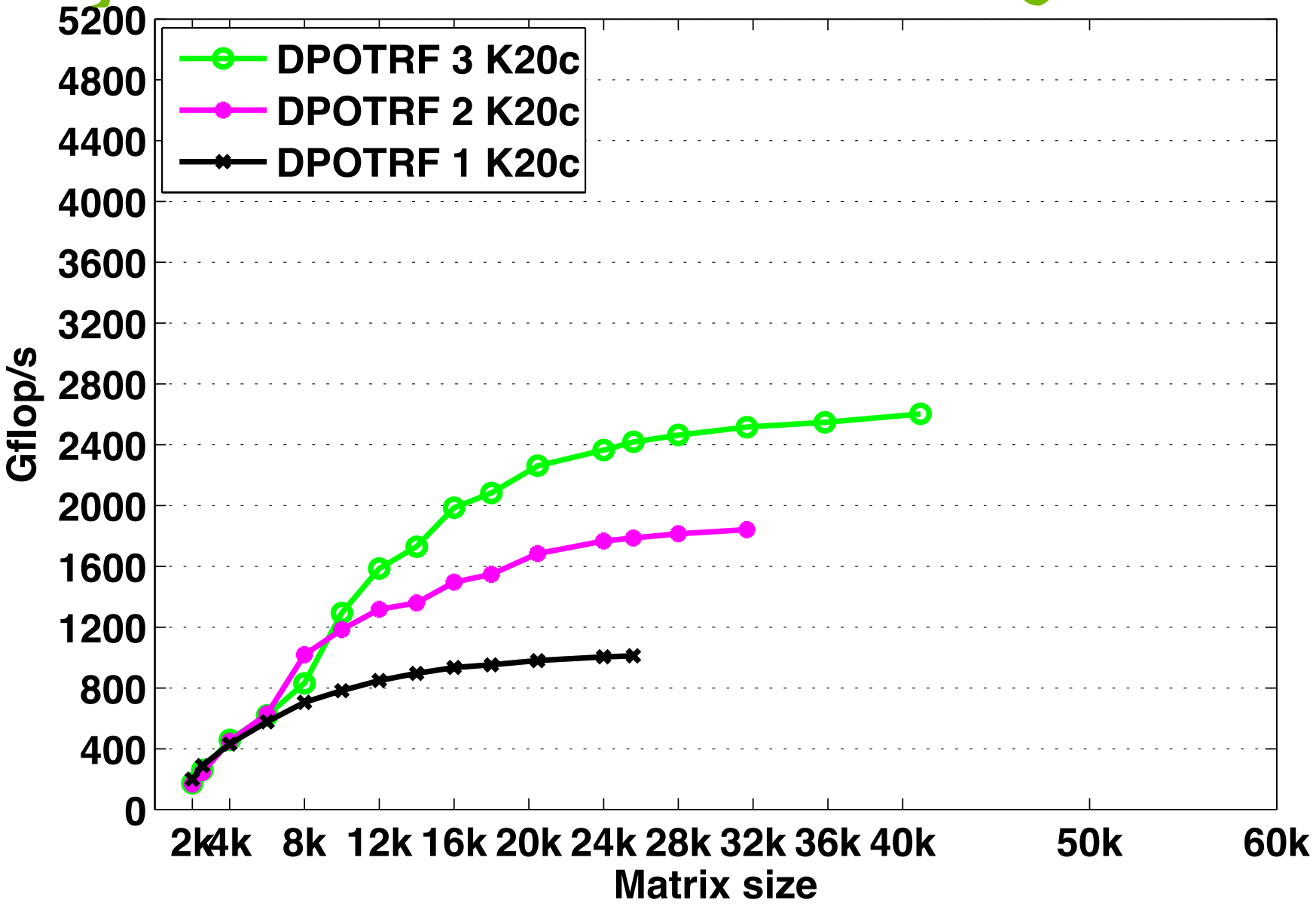
Dynamic MAGMA with QUARK



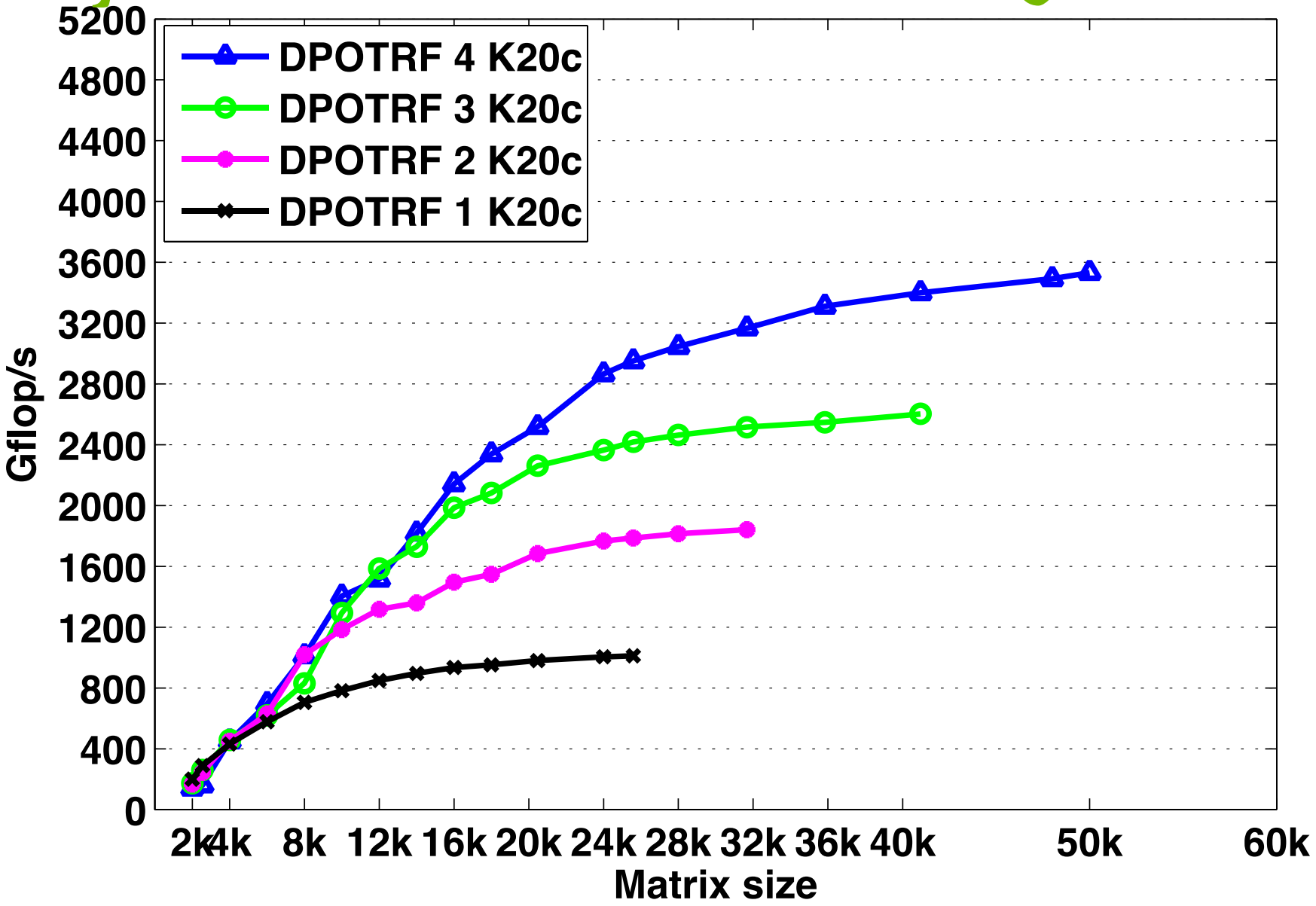
Dynamic MAGMA with QUARK



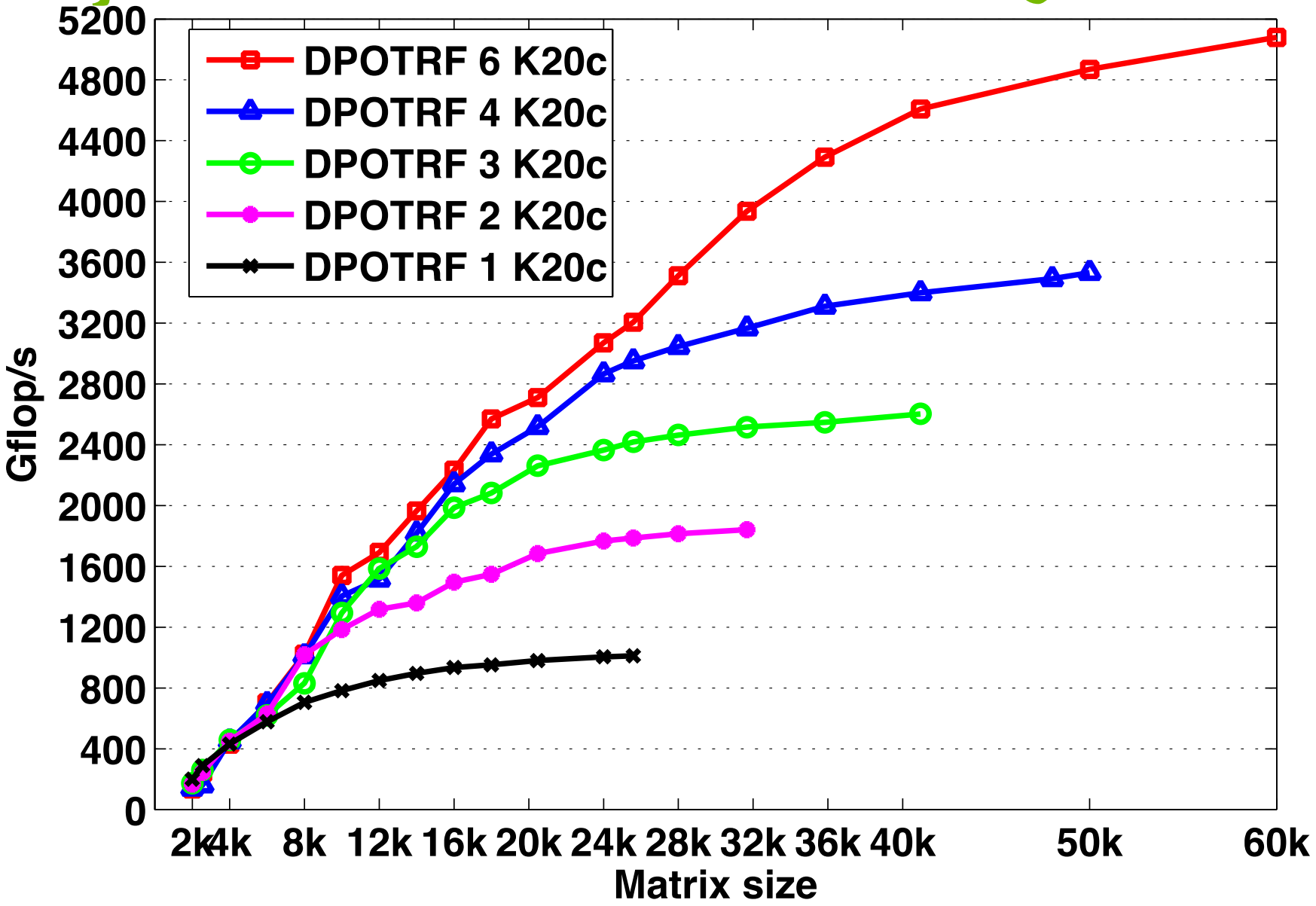
Dynamic MAGMA with QUARK



Dynamic MAGMA with QUARK



Dynamic MAGMA with QUARK



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