Linear Algebra Software for High-Performance Computing

[Part 2: Software for hardware accelerators and coprocessors]

Stan Tomov

Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

ISC High Performance '15
Frankfurt, Germany
July 12, 2015

ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE Department of Electrical Engineering and Computer Science

About ICL

Founded by Prof. Jack Dongarra, ICL celebrated its 25th anniversary

Staff of more than **40 researchers, students, and administrators**

Knoxville, TN

- Mission – provide leading edge tools, enable technologies and software for scientific computing, develop standards for scientific computing in general
- This includes standards and efforts such as
  PVM, MPI, LAPACK, ScaLAPACK, BLAS, ATLAS, Netlib, Top 500, PAPI, NetSolve, and the Linpack Benchmark
- ICL continues these efforts with PLASMA, MAGMA, HPC Challenge, BlackJack, OpenMPI, and MuMI, as well as other innovative computing projects

Outline

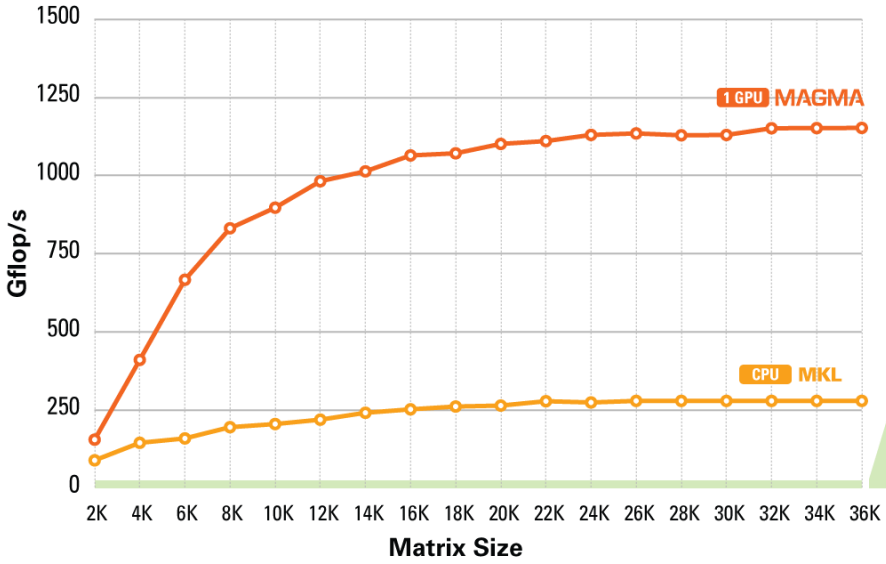
- Motivation
- Overview of the MAGMA library
  - Hybrid GPU+CPU dense linear algebra
  - Methodology and performance
  - Sparse linear algebra
  - Batched linear algebra
  - Power considerations and GPU-only algorithms
- Future Directions

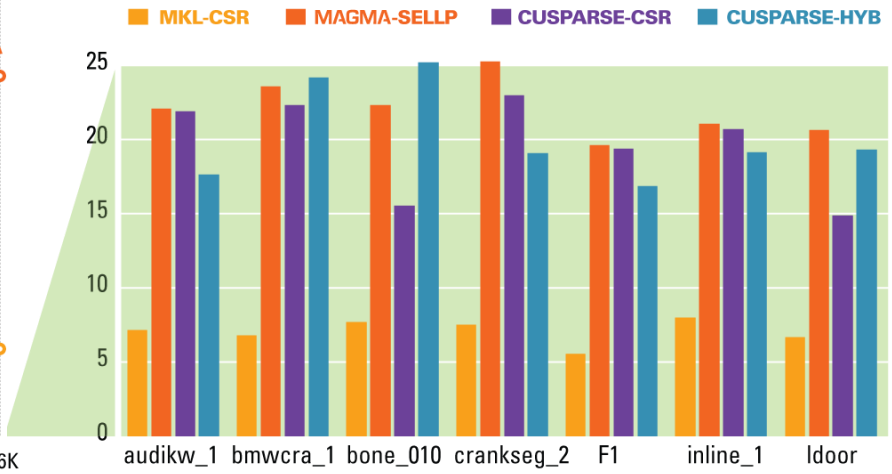
Motivation

- Important scientific applications rely on linear algebra routines

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

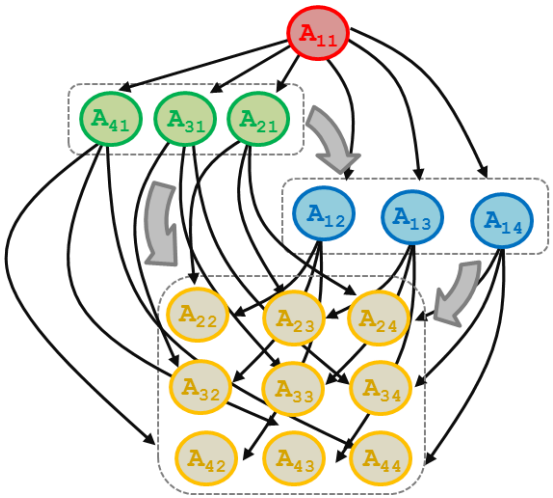
Motivation

- Many dense and sparse direct solvers need HP, energy-efficient LA on many small independent dense matrices
  - Tiled linear algebra algorithms
  - Multifrontal methods
  - Tensor contractions (in high-order FEM, etc)
  - Preconditioners (using DLA) in sparse iterative solvers, many applications, …

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

And many other BLAS/LAPACK, e.g., for application specific solvers, preconditioners, and matrices

Major change to Software

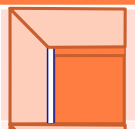
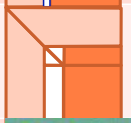
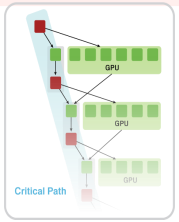
- Must rethink the design of our software for heterogeneous architectures
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
- Numerical libraries for example will change
  - For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this

MAGMA: LAPACK for GPUs

- MAGMA
  - **M**atrix **A**lgebra for **G**PU and **M**ulticore **A**rchitecture
  - LAPACK/ScaLAPACK on hybrid architectures
  - http://icl.cs.utk.edu/magma/

- Next Generation of DLA Software:

- Developers & collaborators:
  - UTK, UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  - Community effort, similarly to LAPACK/ScaLAPACK

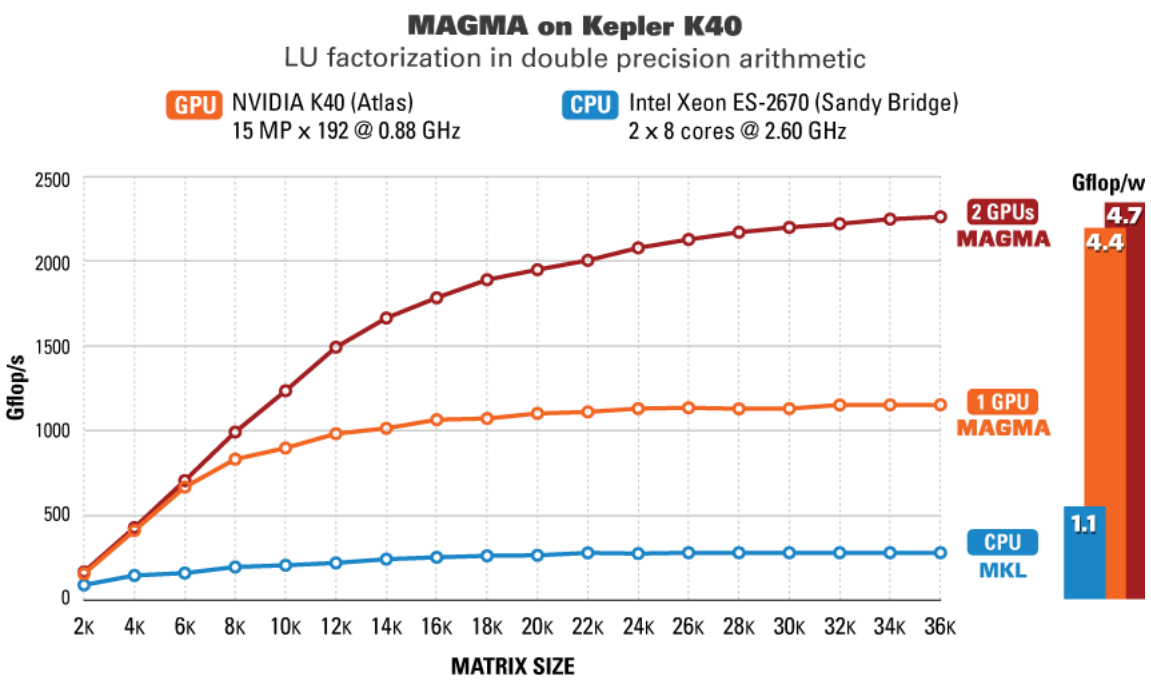
Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (00's) New Algorithms (many-core friendly)		Rely on - a DAG/scheduler - block data layout - some extra kernels
MAGMA Hybrid Algorithms (heterogeneity friendly)		Rely on - hybrid scheduler - hybrid kernels (for nested parallelism)

Key Features of MAGMA 1.6

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

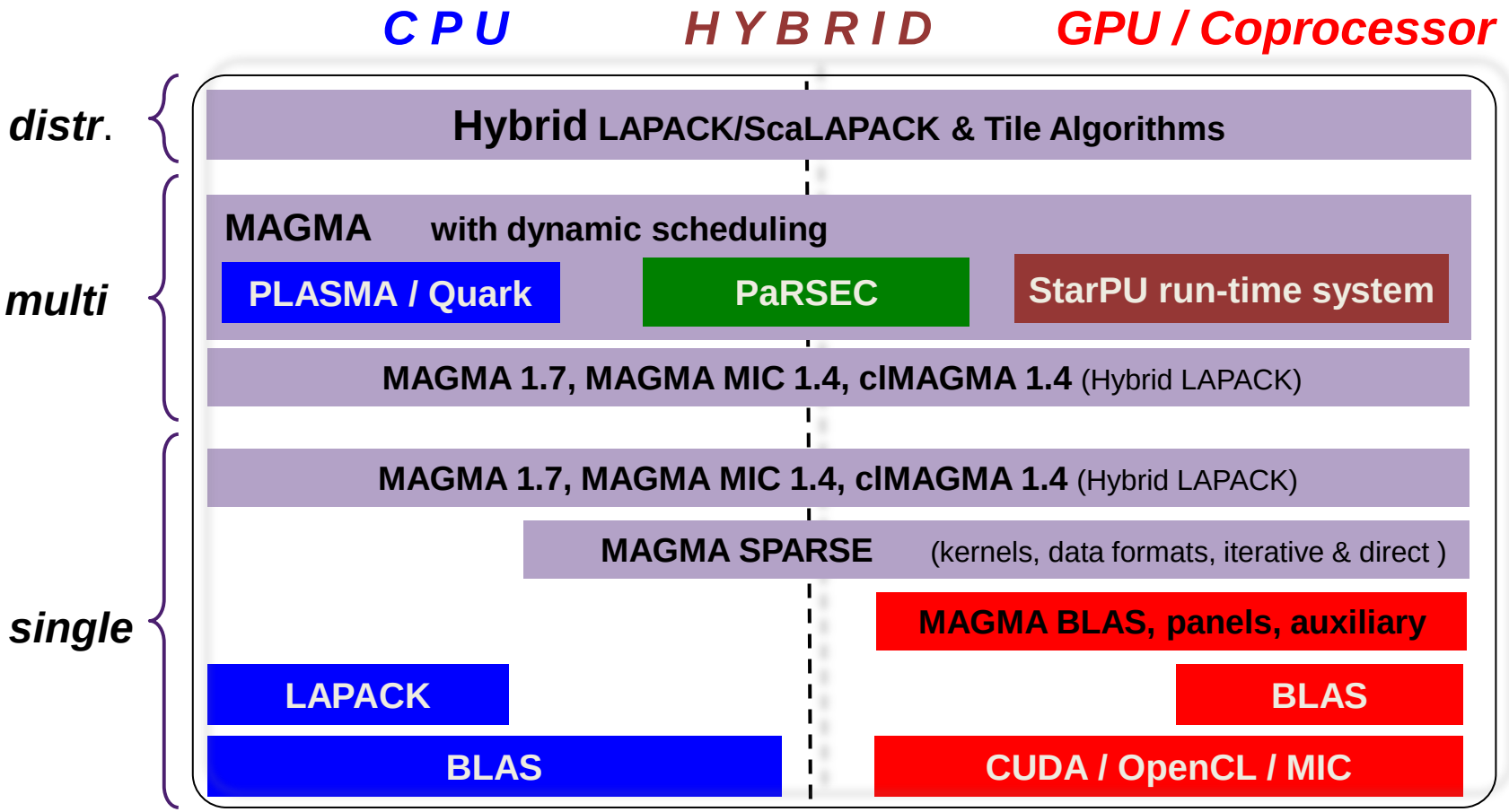


FEATURES AND SUPPORT

- **MAGMA 1.6** FOR **CUDA**
- **clMAGMA 1.3** FOR **OpenCL**
- **MAGMA MIC 1.3** 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***

To install MAGMA

- Get latest MAGMA, MAGMA MIC, or clMAGMA, e.g., MAGMA 1.7:
  > wget http://icl.cs.utk.edu/projectsfiles/magma/downloads/magma-1.7.0.tar.gz
- Unpack the library
  > tar zxvf magma-1.7.0.tar.gz
- Prerequisites: LAPACK, BLAS, and CUDA toolkit
  > module load cudatoolkit intel
- Modify 'make.inc' (where are CUDA & LAPACK, and for what GPU)
  See examples in make.inc.{mkl-gcc | mkl-icc | mkl-ilp64 | mkl-shared | acml | atlas | goto | shared},
  > cp make.inc.mkl-gcc make.inc
  and modify make.inc by setting GPU_TARGET, MKLROOT, and CUDADIR, e.g.,

```
GPU_TARGET ?= Kepler
MKLROOT    = $(INTEL_PATH)/mkl
CUDADIR    = $(CRAY_CUDATOOLKIT_DIR)
```

- Create libmagma.a in 'lib' and testing drivers in 'testing'
  > make
- For more information on installation, read file 'README'

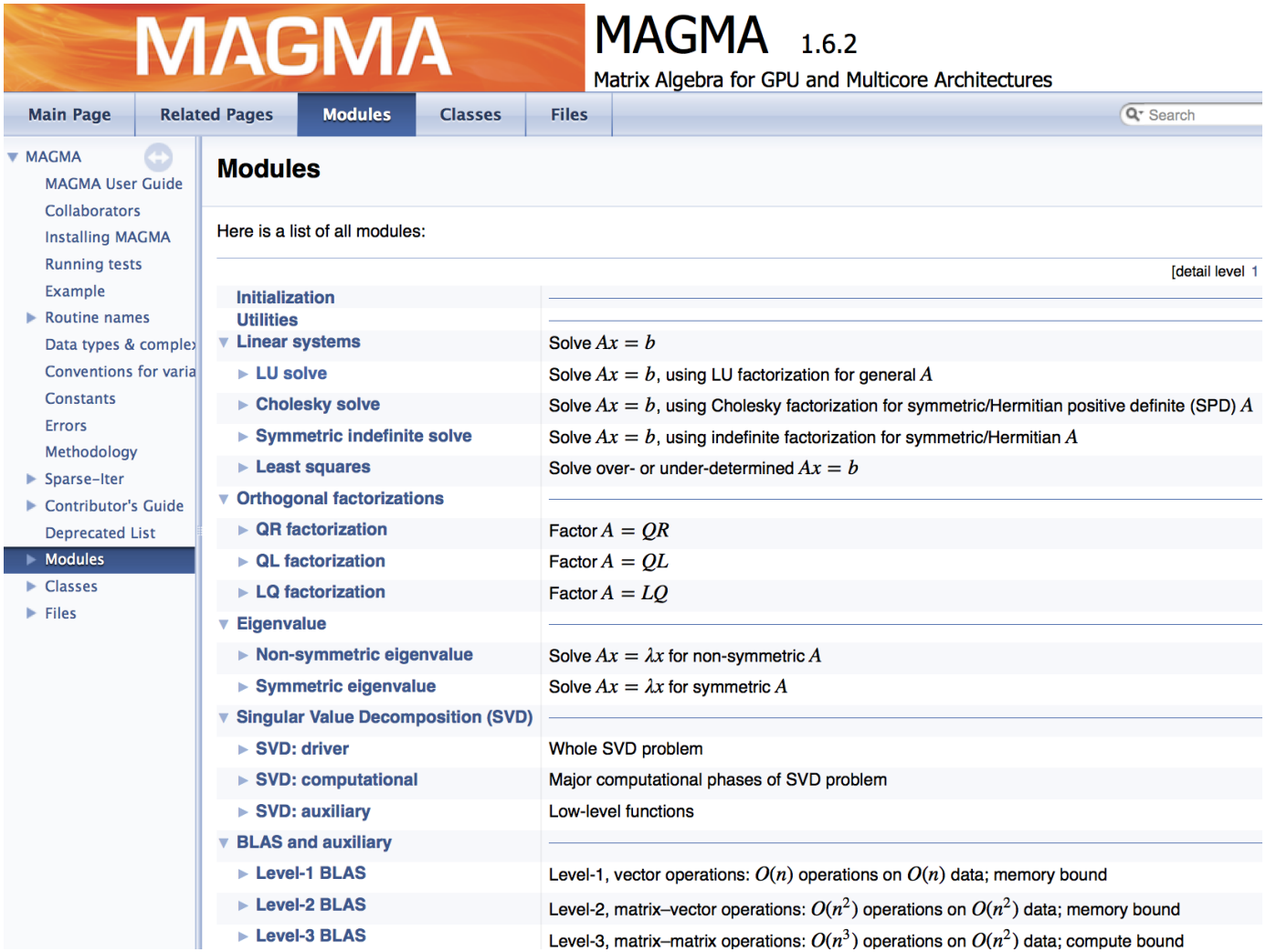
Using MAGMA

- 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/



Using MAGMA

- Naming conventions (e.g., magma_dgesv_gpu)
  - Prefix magma_ or magmablas_
  - Followed by precision
    s single, d double, c single complex, or z double complex
    ds mixed double-single, or zc double complex-single complex
  - Matrix type
    general symmetric hermetian positive definite
    orthogonal unitary triangular
  - Operation
    sv solve ev eigenvalue problem
    trf triangular factorization gv generalized eigenvalue
    qrf QR factorization
  - Suffix _gpu if input and output are on the GPU memory
    (no suffix indicates CPU interface – use GPUs to accelerate where input and output are on the CPU)

Examples / Exercises

- Solving a linear system of equations

$$Ax = b$$

```
#include "magma.h”
#include "magma_lapack.h”
…

magma_init();
…
double *hA, *hB;                                                    // A is a typical array on the CPU
magma_malloc_pinned((void **)&hA,  lda * N * sizeof(double));       // A can be allocated in pinned memory
magma_malloc_cpu(     (void **)&hB, nrhs* M *sizeof(double));
…
init_matrix(M, N, hA, lda);
…
magma_dgetrf( M, N, hA, lda, ipiv, &info);                          // LU factorization (using CPU+GPU)
lapackf77_dgetrs( MagmaNoTransStr, &N, &nrhs,                       // Solve on CPU with LAPACK
                  hA, &lda, ipiv, hB, &ldb, &info );
```

Alternatively, there is a direct MAGMA function to solve:
see testing_dgesv.cpp and testing_dgetrf.cpp

Examples / Exercises

- Solving an eigenvalue problem

$$Ax = \lambda x$$

```
#include "magma.h”
#include "magma_lapack.h”
…

magma_init();
…
double *hA;                                                          // A is a typical array on the CPU
magma_malloc_pinned((void **)&hA,  lda * N * sizeof(double));     // A can be allocated in pinned memory
…
init_matrix(M, N, hA, lda);
…
magma_dsyevd( opts.jobz, opts.uplo,
                        N, hA, lda, w1,
                        h_work, lwork,
                        iwork, liwork, &info );
```

see testing_dsyevd.cpp

Examples / Exercises

- Many benchmarks in testing, e.g.,
  run DGEMM benchmark on my laptop

```
Stans-MacBook-Pro:testing tomov$ ./testing_dgemm -l
% MAGMA 1.6.2 svn compiled for CUDA capability >= 2.0, 32-bit magma_int_t, 64-bit pointer.
% CUDA runtime 7000, driver 7000. MAGMA not compiled with OpenMP.
% device 0: GeForce GT 750M, 925.5 MHz clock, 2047.6 MB memory, capability 3.0
% Sat Jul 11 07:29:32 2015
% Usage: ./testing_dgemm [options] [-h|--help]

% If running lapack (option --lapack), MAGMA and cuBLAS error are both computed
% relative to CPU BLAS result. Else, MAGMA error is computed relative to cuBLAS result.

% transA = No transpose, transB = No transpose
%   M     N     K   MAGMA Gflop/s (ms)  cuBLAS Gflop/s (ms)   CPU Gflop/s (ms)  MAGMA error  cuBLAS  error
%=================================================================================================
  1088  1088  1088     23.06 ( 111.69)      25.25 ( 102.01)     132.46 (  19.45)     8.96e-16     8.96e-16    ok
  2112  2112  2112     24.72 ( 762.16)      28.40 ( 663.42)     151.04 ( 124.74)     1.19e-15     1.19e-15    ok
  3136  3136  3136     26.06 (2367.26)      28.44 (2169.11)     148.88 ( 414.31)     1.56e-15     1.56e-15    ok
  …
```

Examples / Exercises

- Writing a hybrid algorithm

  Develop a hybrid CPU-GPU algorithm for this algorithm (Cholesky QR for a “tall-and-skinny” matrix A)

Algorithm	MATLAB	Hybrid CPU-GPU algorithm

$G = A^T A$	G = A’*A;	cublasDgemm(...);
		magma_dgetvector(...); // Send G from GPU to CPU memory,
		// e.g, hG

$G = L L^T$	U = chol(G);	dpotrf_(…) // LAPACK on CPU to do Cholesky on hG
		magma_dsetvector(...); // send result back to the GPU)

$Q = A (L^T)^{-1}$	Q = A * inv(U)	magma_dtrsm(...); // Triangular matrix solve on the GPU

see testing_dgegqr_gpu.cpp and dgegqr_gpu.cpp

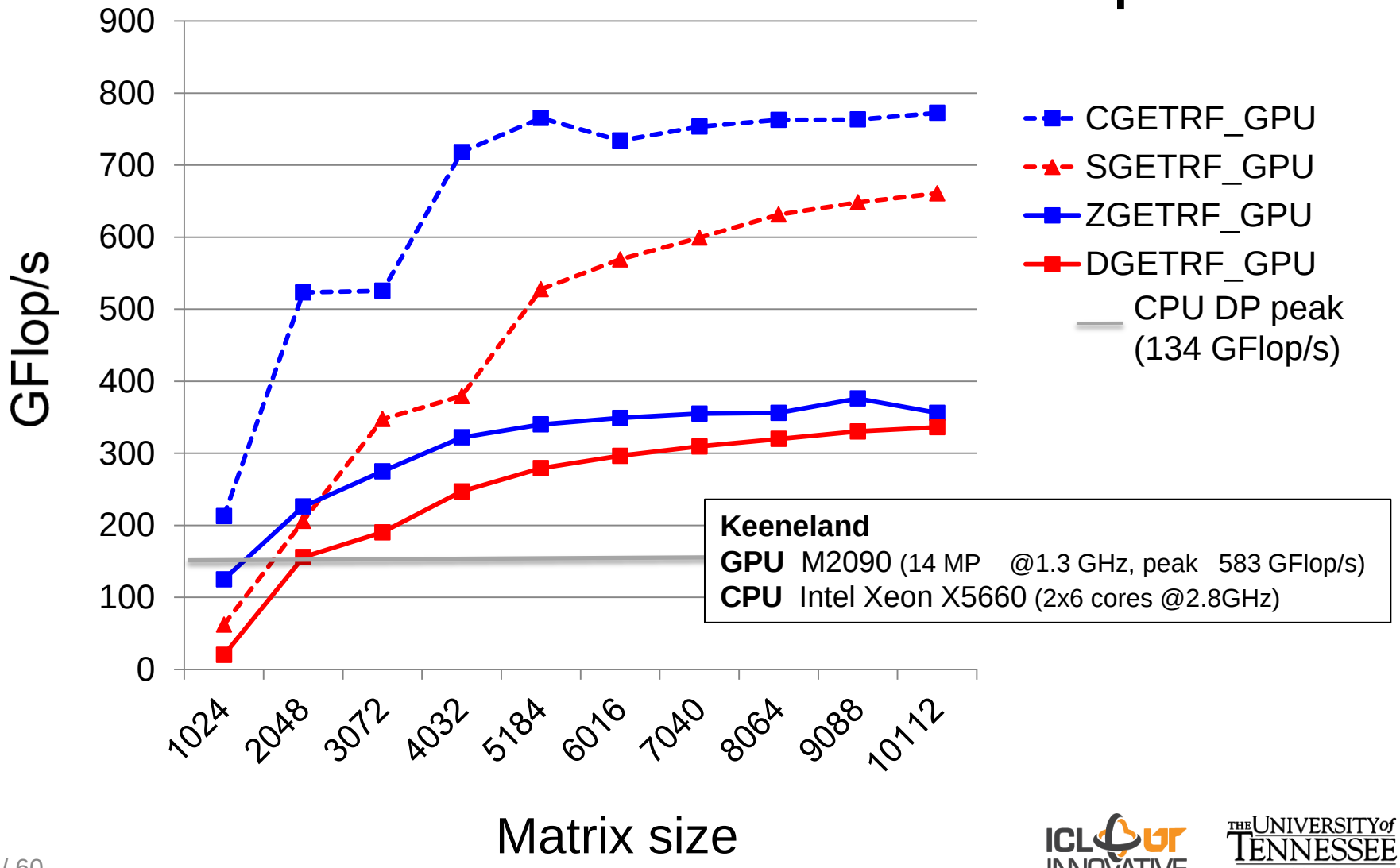
Contributions

The MAGMA program style follows the general guidelines for Sca/LAPACK in terms of interfaces, copyrights and licensing, citing the authors of the software, and documentation:
http://icl.cs.utk.edu/projectsfiles/magma/doxygen/contributors-guide.html

- Routine Naming and Design
  - Use BLAS
  - Develop in double complex; other precisions are generated
  - Machine parameters are determined at magma_init (as needed for tuning)
  - Block sizes are provided by extern "C" int magma_get_ROUTINE_nb(int m), i.e., not by ilaenv
- Language, source formatting, timing, and testing
  - Written in C/C++; wrappers for Fortran are supported

Multiple precision support

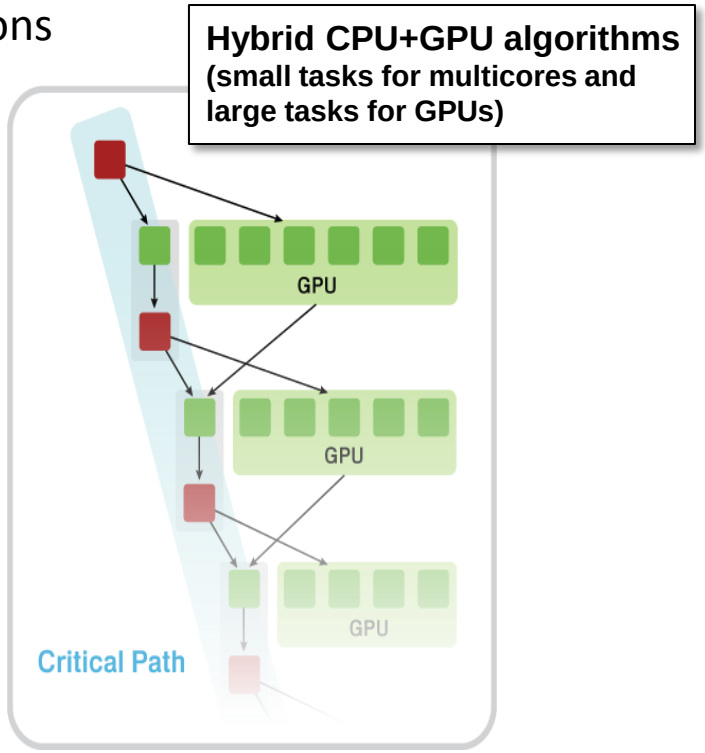
Performance of the LU factorization in various precisions



Methodology overview

A methodology to use all available resources:

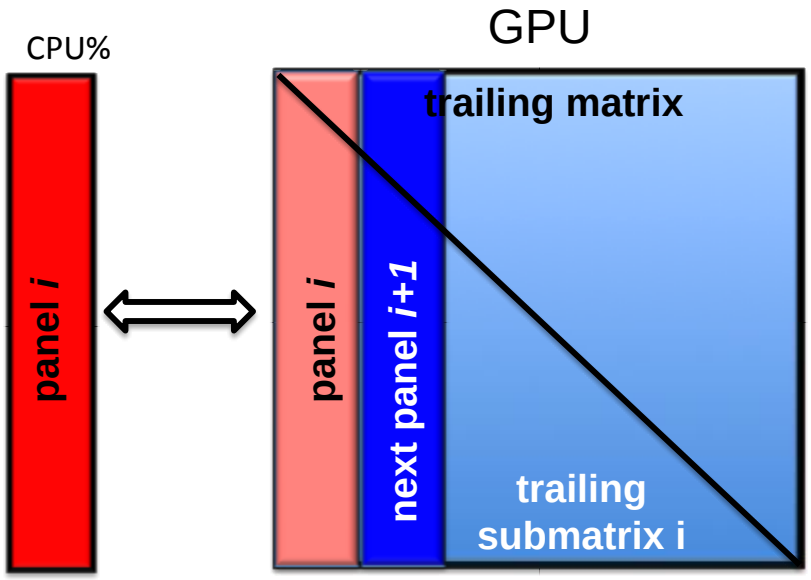
- MAGMA uses hybrid algorithms based on
  - Representing linear algebra algorithms as collections of tasks and data dependencies among them
  - Properly scheduling tasks' execution over multicore and GPU hardware components
- Successfully applied to fundamental linear algebra algorithms
  - One- and two-sided factorizations and solvers
  - Iterative linear and eigensolvers
- Productivity
  - 1) High level;
    2) Leveraging prior developments;
    3) Exceeding in performance homogeneous solutions



Hybrid Algorithms

One-Sided Factorizations (LU, QR, and Cholesky)

- Panels (Level 2 BLAS) are factored on CPU using LAPACK
- Trailing matrix updates (Level 3 BLAS) are done on the MIC using "look-ahead"



A Hybrid Algorithm Example

Left-looking hybrid Cholesky

From sequential LAPACK

```
for( j=0, j<n; j+=nb) {
    jb = min(nb, n-j);
    zherk( MagmaUpper
           jb, j, one, dA(0
    if (j+jb < n)
      zgemm( MagmaCo
             dA(0,j), ldd
    zpotrf( MagmaUpper
    if (info != 0)
       *info += j;
    If (j+jb) < n) {
      ztrsm( MagmaLeft,
             jb, n
    }
}
```

to parallel hybrid **MAGMA**

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

Note:

- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences are further hidden in a dynamic scheduler making the top level representation of MAGMA algorithms almost identical to LAPACK

A Hybrid Algorithm Example

Left-looking hybrid Cholesky

From sequential LAPACK

```
for( j=0, j<n; j+=nb) {
    jb = min(nb, n-j);
    zherk( MagmaUpper
           jb, j, one, dA(0
    if (j+jb < n)
      zgemm( MagmaCo
             dA(0,j), ldd
    zpotrf( MagmaUpper
    if (info != 0)
       *info += j;
    If (j+jb) < n) {
      ztrsm( MagmaLeft,
                jb, n
    }
}
```

to parallel hybrid MAGMA

```
1  for( j=0, j<n; j+=nb) {
2      jb = min(nb, n-j);
3      magma_zherk( MagmaUpper, MagmaConjTra
                    jb, j, one, dA(0,j), ldda, one, dA
4      magma_zgetmatrix_async( jb, jb, dA(j,j), ldd
5      if (j+jb < n)
6        magma_zgemm( MagmaConjTrans, Magma
                      dA(0,j), ldda, dA(0,j+jb), ldd
7      magma_event_sync( event );
8      zpotrf( MagmaUpperStr, &jb, work, &jb, info);
9      if (info != 0)
10       *info += j;
11     magma_zsetmatrix_async(jb, jb, work, jb, dA
12     If (j+jb) < n) {
13       magma_event_sync( event );
14       magma_ztrsm( MagmaLeft, MagmaUpper,
                      jb, n-j-jb, one, dA(j,j), ldda, d
       }
   }
```

MAGMA runtime environment

- Scheduling can be static or dynamic
- Dynamic is based on QUARK
- Uses CUDA streams to offload computation to the GPU

Note:
- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences are further hidden in a dynamic scheduler making the top level representation of MAGMA algorithms almost identical to LAPACK

A Hybrid Algorithm Example

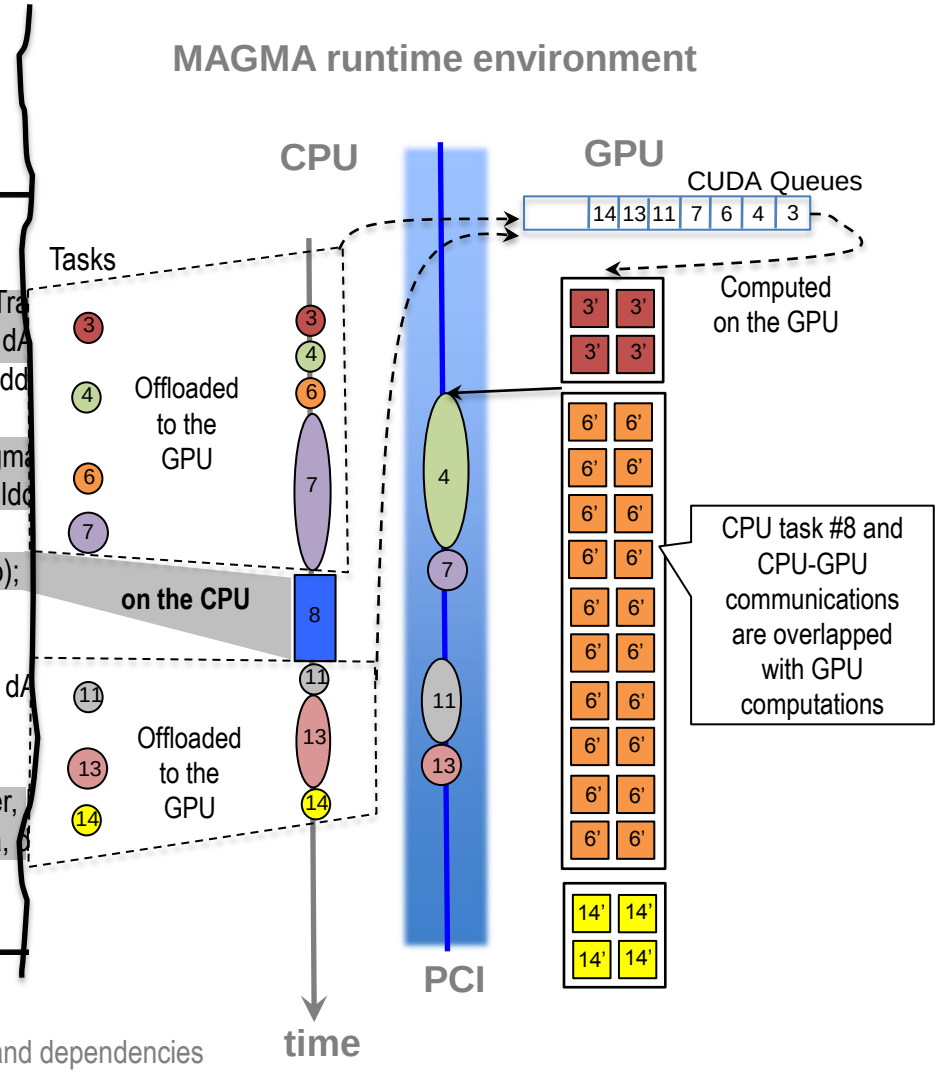
Left-looking hybrid Cholesky

From sequential LAPACK

```
for( j=0, j<n; j+=nb) {
    jb = min(nb, n-j);
    zherk( MagmaUpper
           jb, j, one, dA(0
    if (j+jb < n)
      zgemm( MagmaCo
             dA(0,j), ldd
    zpotrf( MagmaUpper
    if (info != 0)
       *info += j;
    If (j+jb) < n) {
      ztrsm( MagmaLeft,
             jb, n
    }
}
```

to parallel hybrid MAGMA

```
1  for( j=0, j<n; j+=nb) {
2      jb = min(nb, n-j);
3      magma_zherk( MagmaUpper, MagmaConjTra
                    jb, j, one, dA(0,j), ldda, one, dA
4      magma_zgetmatrix_async( jb, jb, dA(j,j), ldd
5      if (j+jb < n)
6        magma_zgemm( MagmaConjTrans, Magma
                      dA(0,j), ldda, dA(0,j+jb), ldd
7      magma_event_sync( event );
8      zpotrf( MagmaUpperStr, &jb, work, &jb, info);
9      if (info != 0)
10       *info += j;
11     magma_zsetmatrix_async(jb, jb, work, jb, dA
12     If (j+jb) < n) {
13       magma_event_sync( event );
14       magma_ztrsm( MagmaLeft, MagmaUpper,
                      jb, n-j-jb, one, dA(j,j), ldda, d
       }
   }
```



Note:

- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences are further hidden in a dynamic scheduler making the top level representation of MAGMA algorithms almost identical to LAPACK

A Hybrid Algorithm Example

Left-looking hybrid Cholesky

From sequential LAPACK

```
for( j=0, j<n; j+=nb) {
    jb = min(nb, n-j);
    zherk( MagmaUpper
           jb, j, one, dA(0
    if (j+jb < n)
      zgemm( MagmaCo
             dA(0,j), ldd
    zpotrf( MagmaUpper
    if (info != 0)
       *info += j;
    If (j+jb) < n) {
      ztrsm( MagmaLeft,
                   jb, n
    }
}
```

to parallel hybrid MAGMA

```
1  for( j=0, j<n; j+=nb) {
2      jb = min(nb, n-j);
3      magma_zherk( MagmaUpper, MagmaConjTra
                    jb, j, one, dA(0,j), ldda, one, dA
4      magma_zgetmatrix_async( jb, jb, dA(j,j), ldd
5      if (j+jb < n)
6        magma_zgemm( MagmaConjTrans, Magma
                      dA(0,j), ldda, dA(0,j+jb), ldd
7      magma_event_sync( event );
8      zpotrf( MagmaUpperStr, &jb, work, &jb, info);
9      if (info != 0)
10       *info += j;
11     magma_zsetmatrix_async(jb, jb, work, jb, dA
12     If (j+jb) < n) {
13       magma_event_sync( event );
14       magma_ztrsm( MagmaLeft, MagmaUpper,
                      jb, n-j-jb, one, dA(j,j), ldda, d
       }
   }
```

MAGMA wrappers to support:

- GPUs with CUDA
- Xeon Phi
- OpenCL

Example: a zgemm interface in MAGMA MIC

```
// ======================================
// BLAS functions
magma_err_t
magma_zgemm(
    magma_trans_t transA, magma_trans_t transB,
    magma_int_t m, magma_int_t n, magma_int_t k,
    magmaDoubleComplex alpha, magmaDoubleComplex_const_ptr dA, size_t dA_offset, magma_i
                              magmaDoubleComplex_const_ptr dB, size_t dB_offset, magma_i
    magmaDoubleComplex beta,  magmaDoubleComplex_ptr       dC, size_t dC_offset, magma_i
    magma_queue_t handle )
{
    int err;
    magma_mic_zgemm_param gemm_param;

    gemm_param.transa = transA;
    gemm_param.transb = transB;
    gemm_param.m      = m;
    gemm_param.n      = n;
    gemm_param.k      = k;
    gemm_param.alpha  = alpha;
    gemm_param.a      = dA + dA_offset;
    gemm_param.lda    = lda;
    gemm_param.b      = dB + dB_offset;
    gemm_param.ldb    = ldb;
    gemm_param.beta   = beta;
    gemm_param.c      = dC + dC_offset;
    gemm_param.ldc    = ldc;

    int control_msg = magma_mic_ZGEMM;

    if ((err = scif_send(gEpd, &control_msg, sizeof(control_msg), 1)) <= 0) {
        err = errno;
        printf("scif_se  failed with err %d\n", errno);
        fflush(stdout);
    }

    if ((err = scif_send(g                ram, sizeof(gemm_param), 1)) <= 0) {
                rno;
                scif_send
```

Note:

- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences are further hidden in a dynamic scheduler making th
  representation of MAGMA algorithms almost identical to LAPAC

Send asynchronous requests to the MIC;
Queued & Executed on the MIC

A Hybrid Algorithm Example

Left-looking hybrid Cholesky

From sequential LAPACK

```
for( j=0, j<n; j+=nb) {
    jb = min(nb, n-j);
    zherk( MagmaUpper
           jb, j, one, dA(0
    if (j+jb < n)
      zgemm( MagmaCo
             dA(0,j), ldd
    zpotrf( MagmaUpper
    if (info != 0)
       *info += j;
    If (j+jb) < n) {
      ztrsm( MagmaLeft,
                jb, n
    }
}
```

to parallel hybrid MAGMA

```
1  for( j=0, j<n; j+=nb) {
2      jb = min(nb, n-j);
3      magma_zherk( MagmaUpper, MagmaConjTra
                    jb, j, one, dA(0,j), ldda, one, dA
4      magma_zgetmatrix_async( jb, jb, dA(j,j), ldd
5      if (j+jb < n)
6        magma_zgemm( MagmaConjTrans, Magma
                      dA(0,j), ldda, dA(0,j+jb), ldd
7      magma_event_sync( event );
8      zpotrf( MagmaUpperStr, &jb, work, &jb, info);
9      if (info != 0)
10       *info += j;
11     magma_zsetmatrix_async(jb, jb, work, jb, dA
12     If (j+jb) < n) {
13       magma_event_sync( event );
14       magma_ztrsm( MagmaLeft, MagmaUpper,
                      jb, n-j-jb, one, dA(j,j), ldda, d
       }
   }
```

MAGMA wrappers to support:

- GPUs with CUDA
- Xeon Phi
- OpenCL

Example: a zsetmatrix_async in clMAGMA

```
// -------------------
magma_err_t
magma_zgetmatrix_async(
    magma_int_t m, magma_int_t n,
    magmaDoubleComplex_const_ptr dA_src, size_t dA_offset, magma
    magmaDoubleComplex*          hA_dst, size_t hA_offset, magma
    magma_queue_t queue, magma_event_t *event )
{
    size_t buffer_origin[3] = { dA_offset*sizeof(magmaDoubleComp
    size_t host_orig[3]     = { 0, 0, 0 };
    size_t region[3]        = { m*sizeof(magmaDoubleComplex), n,
    cl_int err = clEnqueueReadBufferRect(
        queue, dA_src, CL_FALSE,  // non-blocking
        buffer_origin, host_orig, region,
        ldda*sizeof(magmaDoubleComplex), 0,
        ldha*sizeof(magmaDoubleComplex), 0,
        hA_dst, 0, NULL, event );
    return err;
}
```

Note:

- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences are further hidden in a dynamic scheduler making the top level representation of MAGMA algorithms almost identical to LAPACK

Mixed precision iterative refinement

- Iterative refinement for dense systems, $Ax = b$, can work this way.

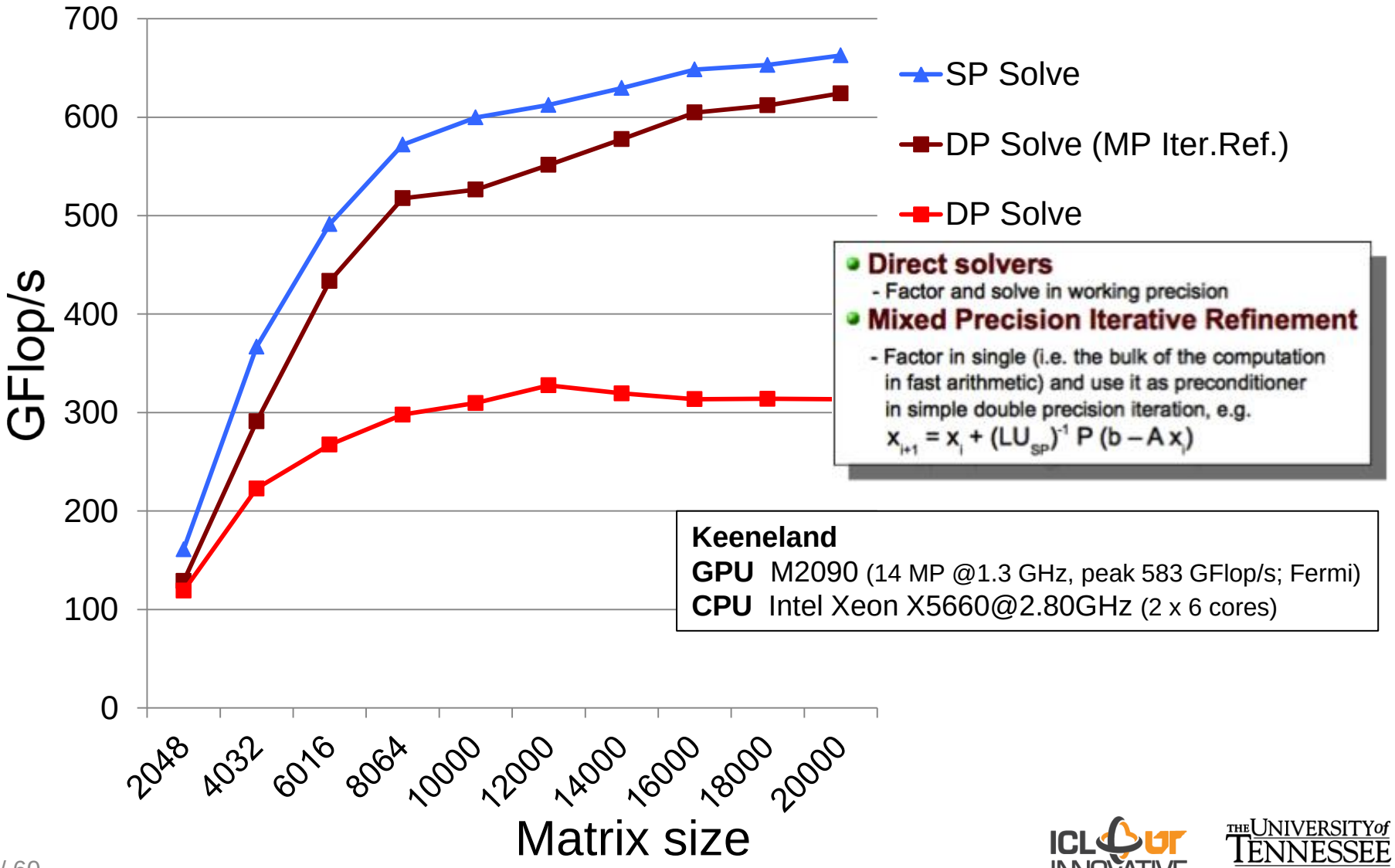
L U = lu(A)	SINGLE	$O(n^3)$
x = L\(U\b)	SINGLE	$O(n^2)$
r = b – Ax	DOUBLE	$O(n^2)$
WHILE || r || not small enough		
z = L\(U\r)	SINGLE	$O(n^2)$
x = x + z	DOUBLE	$O(n^1)$
r = b – Ax	DOUBLE	$O(n^2)$
END		

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

- Requires extra storage, total is 1.5 times normal;
- $O(n^3)$ work is done in lower precision
- $O(n^2)$ work is done in high precision
- Problems if the matrix is ill-conditioned in sp; $O(10^8)$

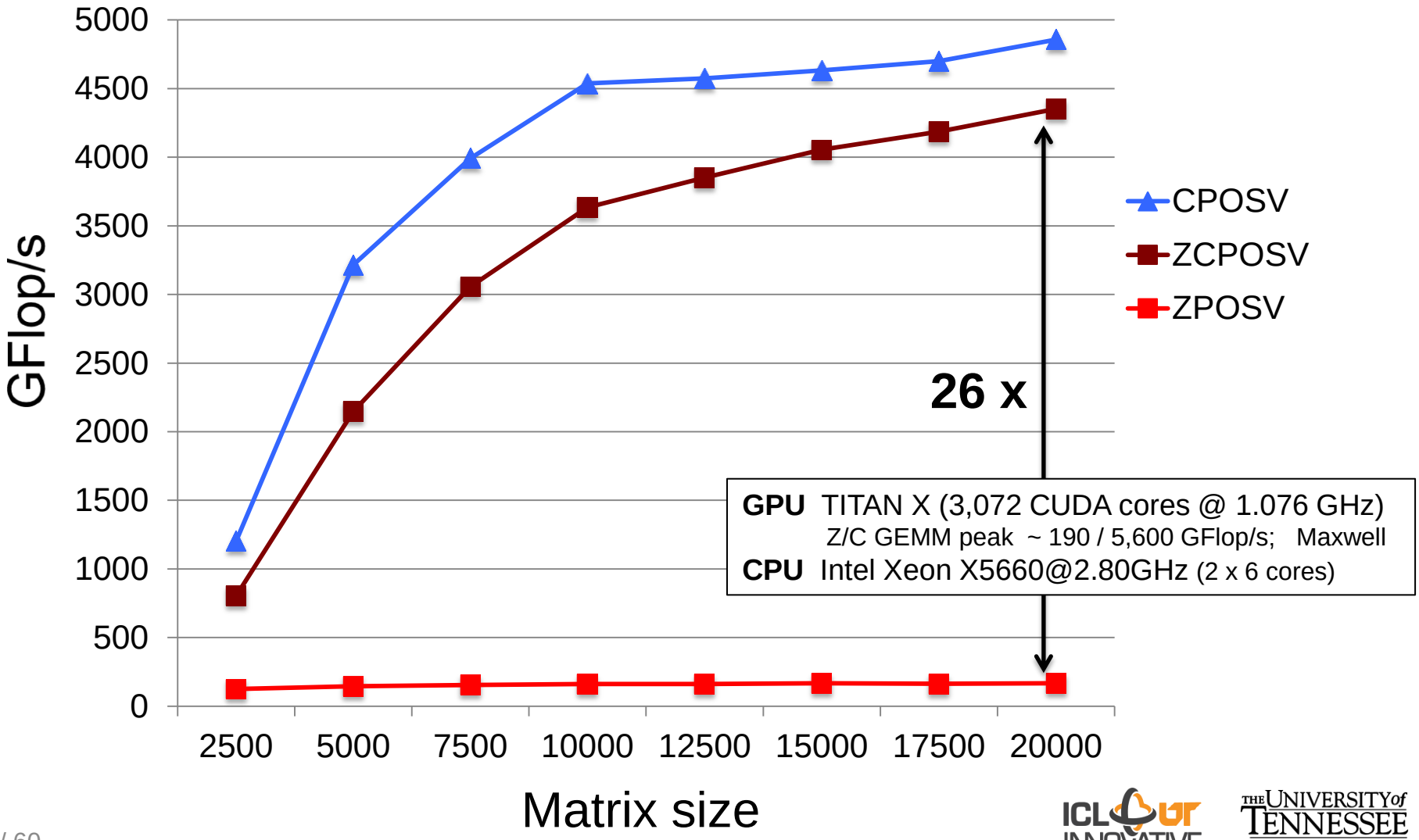
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



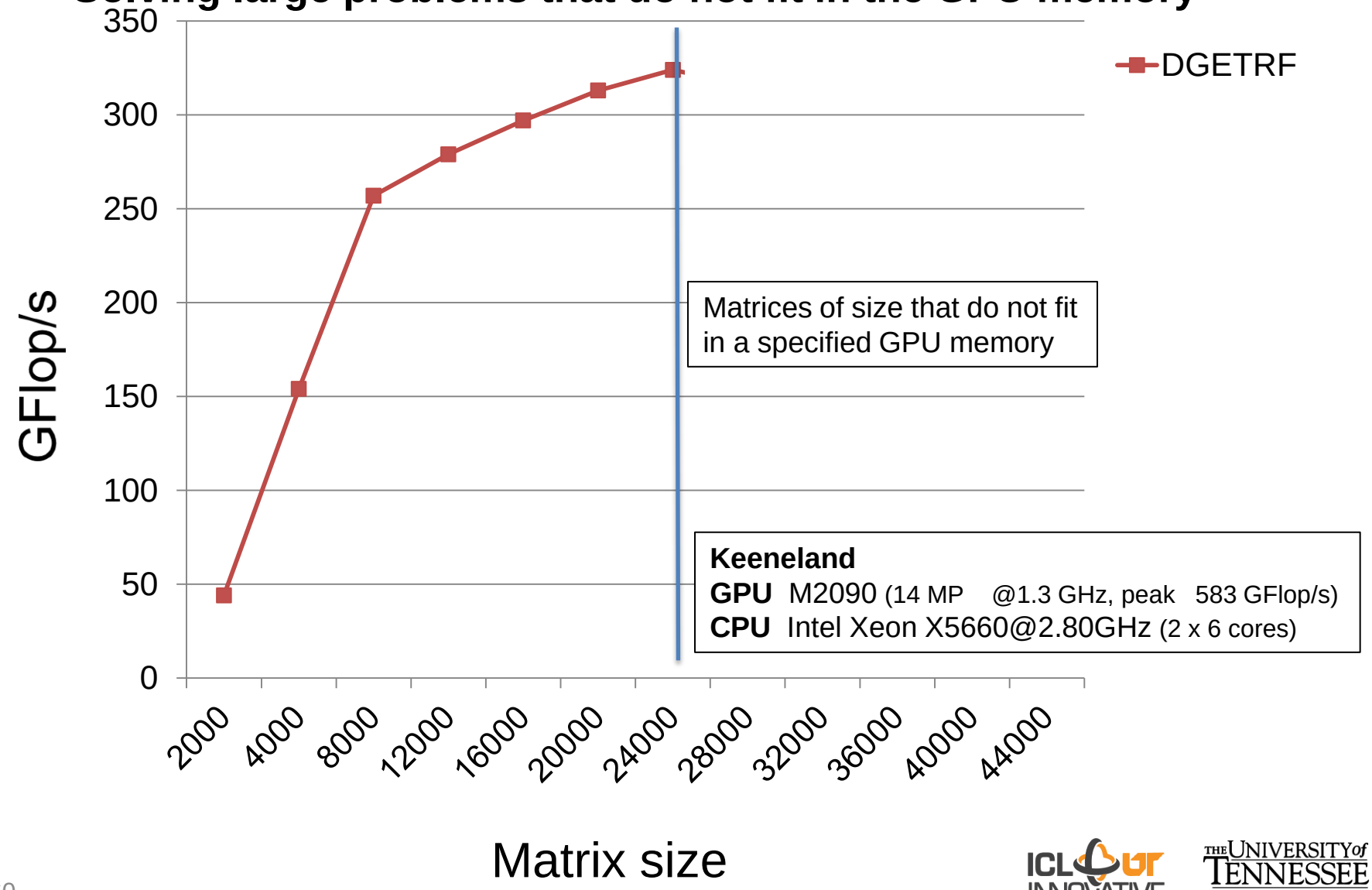
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



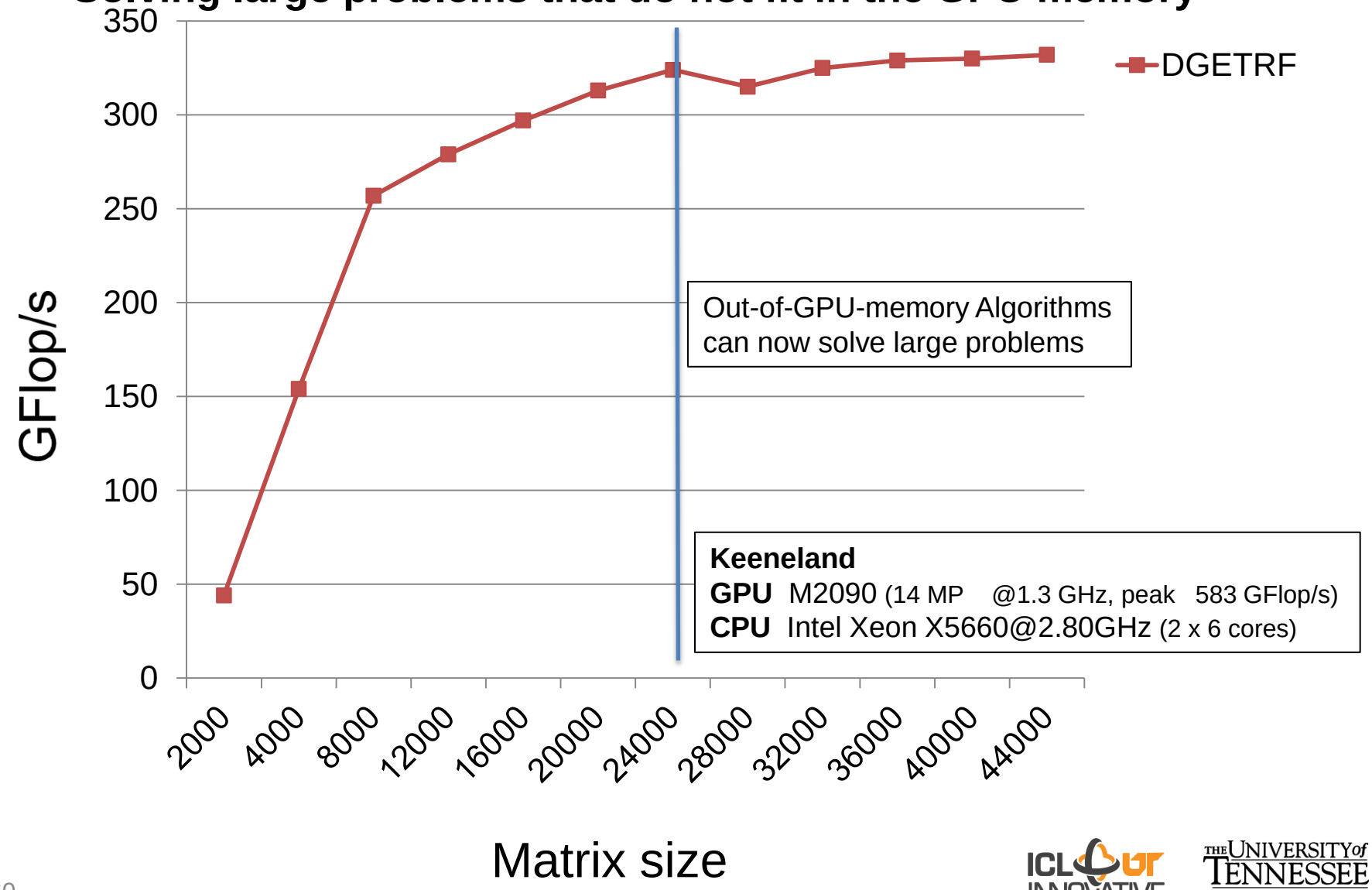
Out of GPU Memory Algorithms

Solving large problems that do not fit in the GPU memory



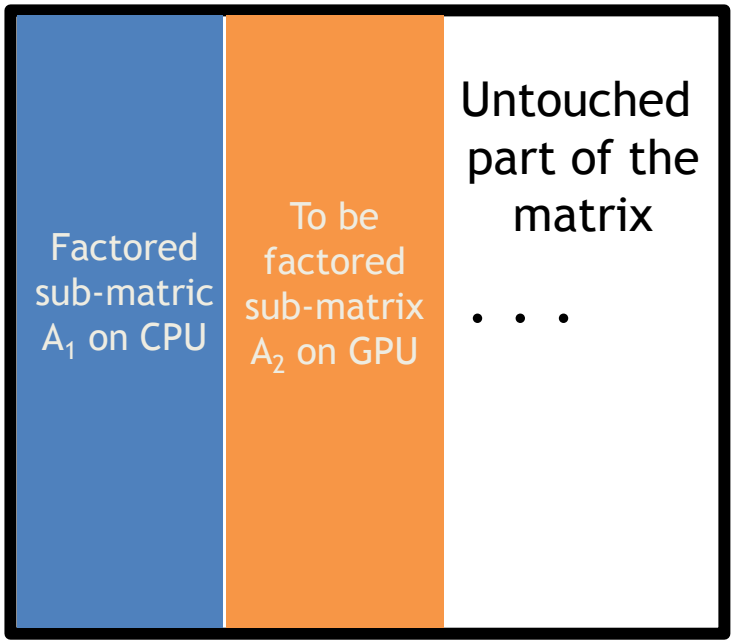
Out of GPU Memory Algorithms

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

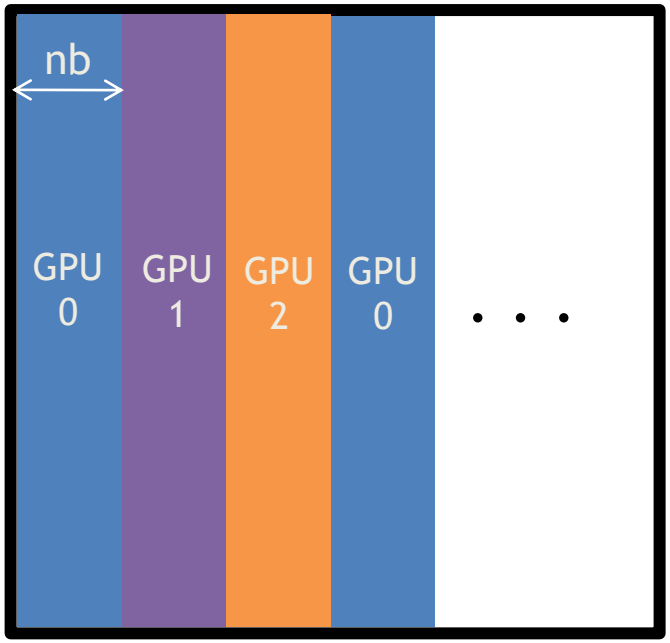


1) Copy A_2 to the GPU
2) Update A_2 using A_1 (a panel of A_1 at a time)
3) Factor the updated A_2 using existing hybrid code
4) Copy factored A_2 to the CPU

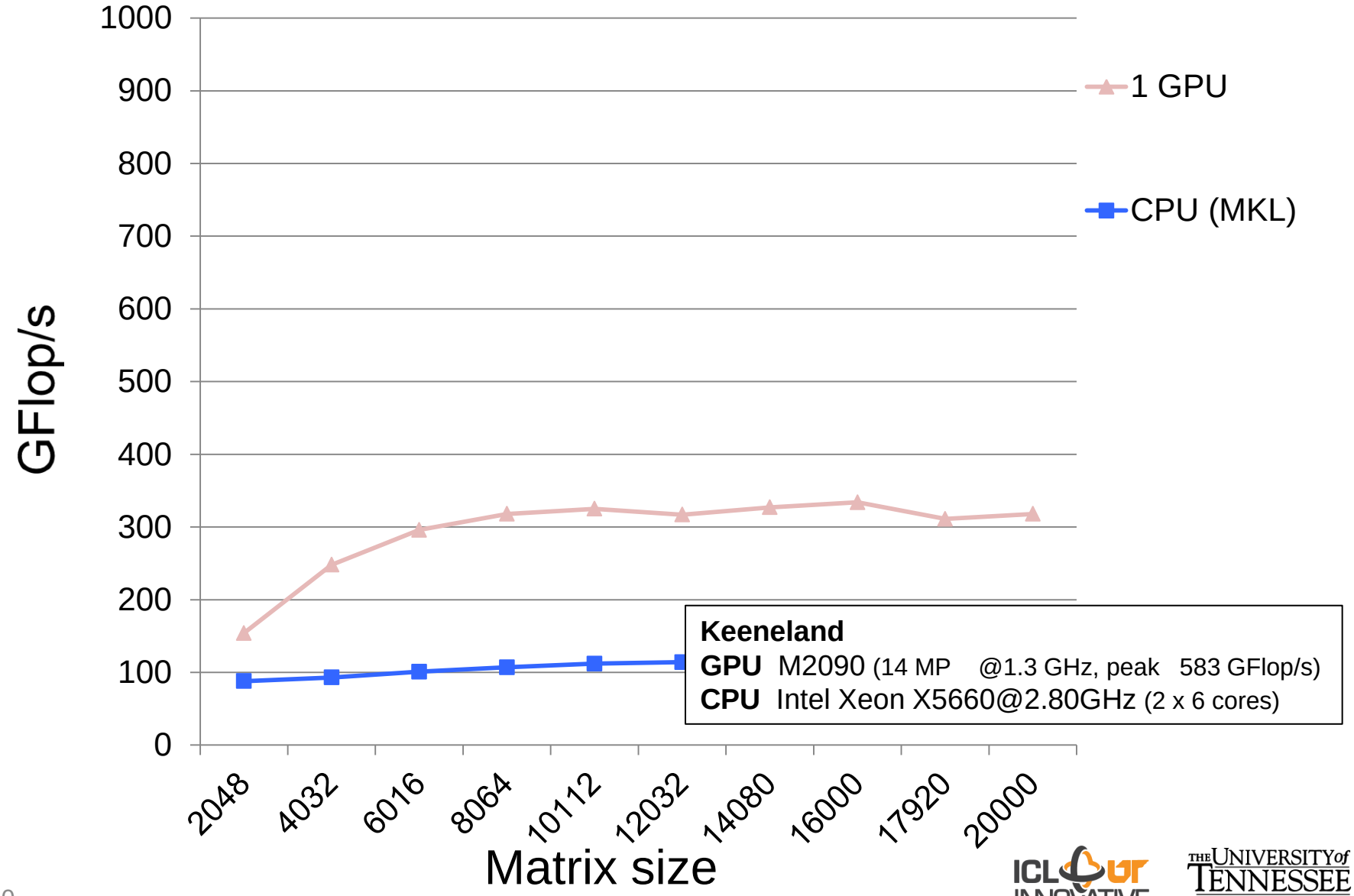
Trivially extended to multiGPUs:
A_2 is "larger" with 1-D block cyclic distribution, again reusing existing algorithms

MultiGPU Support

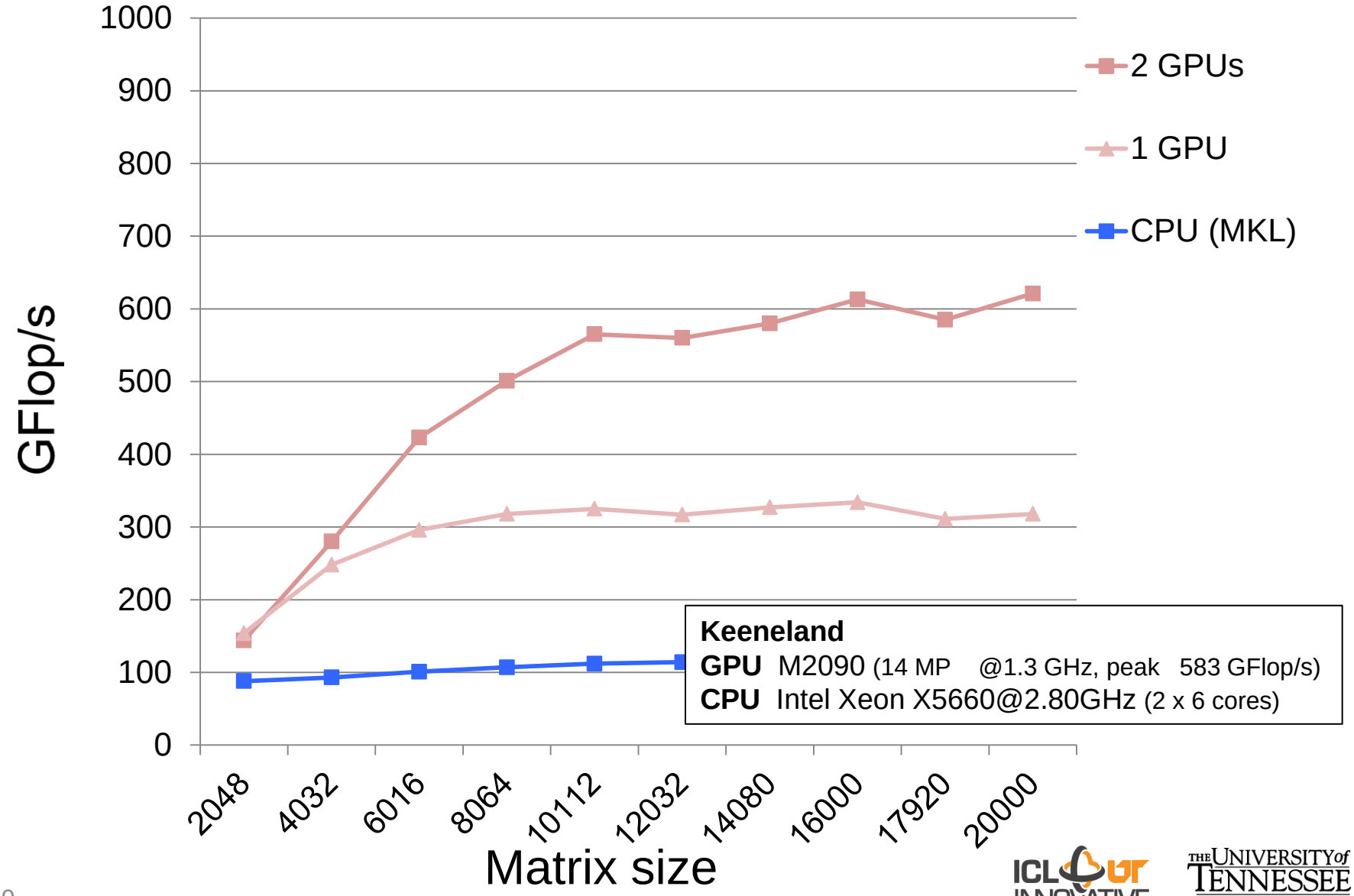
- Data distribution
  - 1-D block-cyclic distribution
- Algorithm
  - GPU holding current panel is sending it to CPU
  - All updates are done in parallel on the GPUs
  - Look-ahead is done with GPU holding the next panel



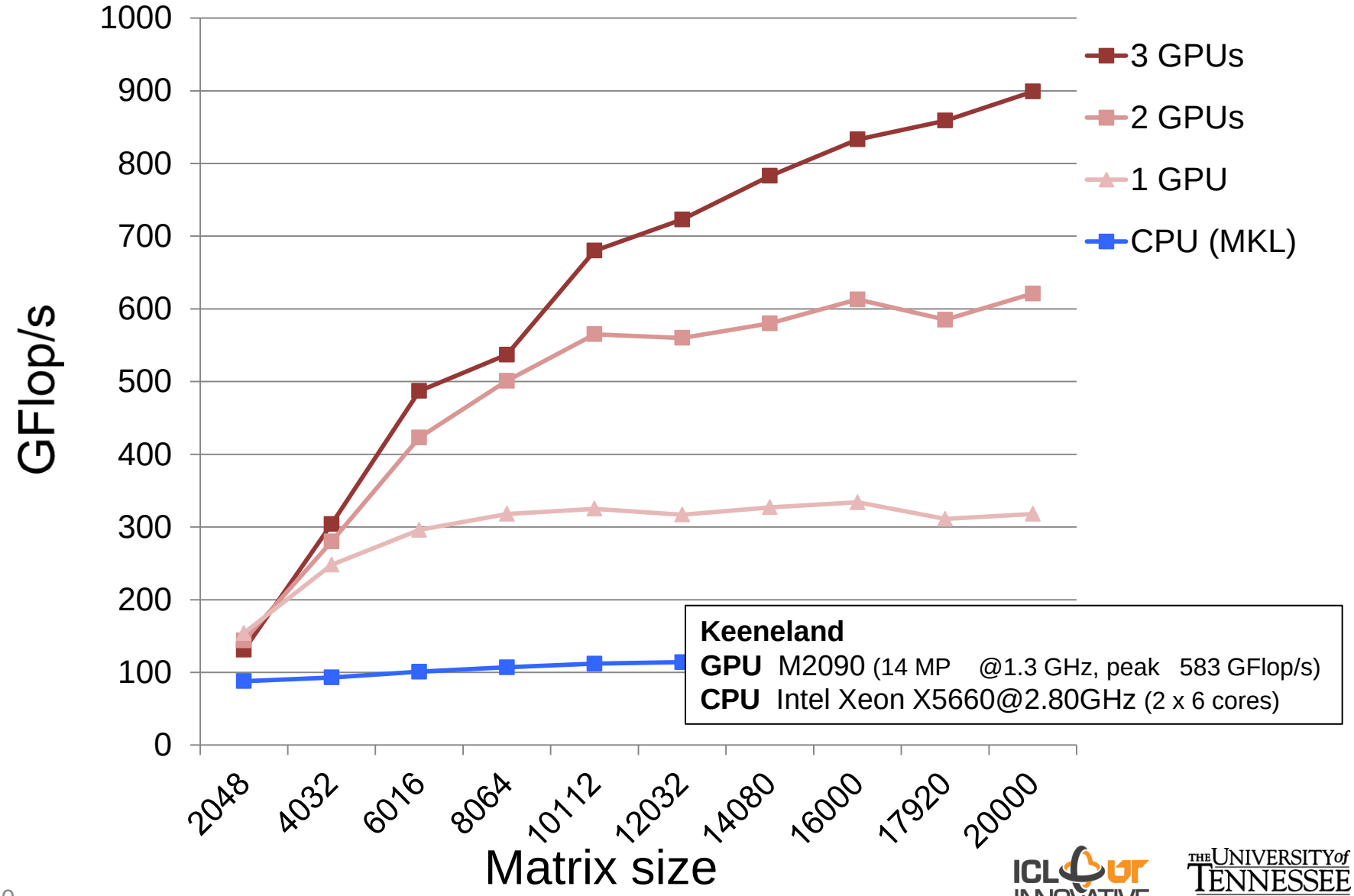
LU on multiGPUs in DP



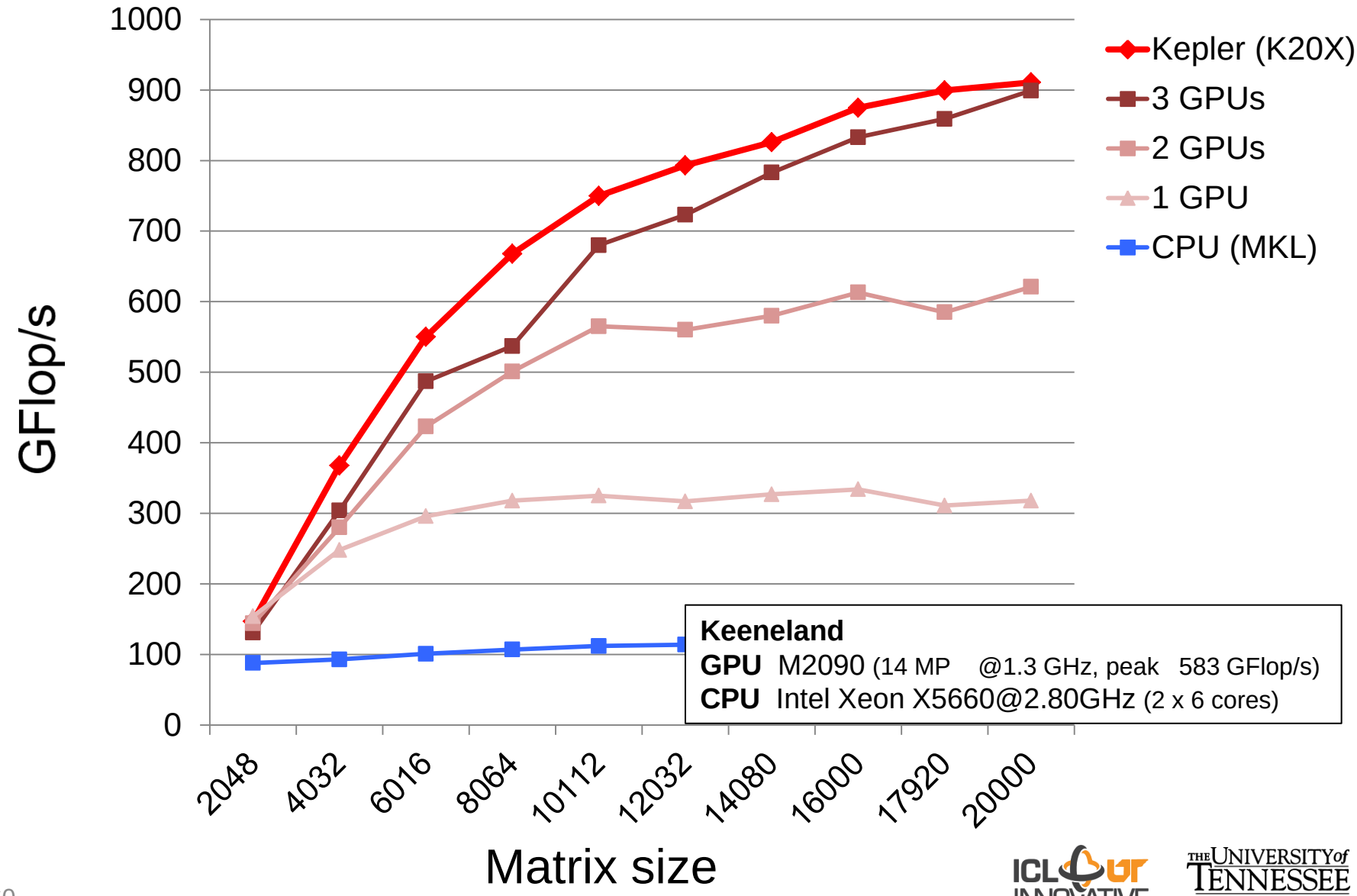
LU on multiGPUs in DP



LU on multiGPUs in DP

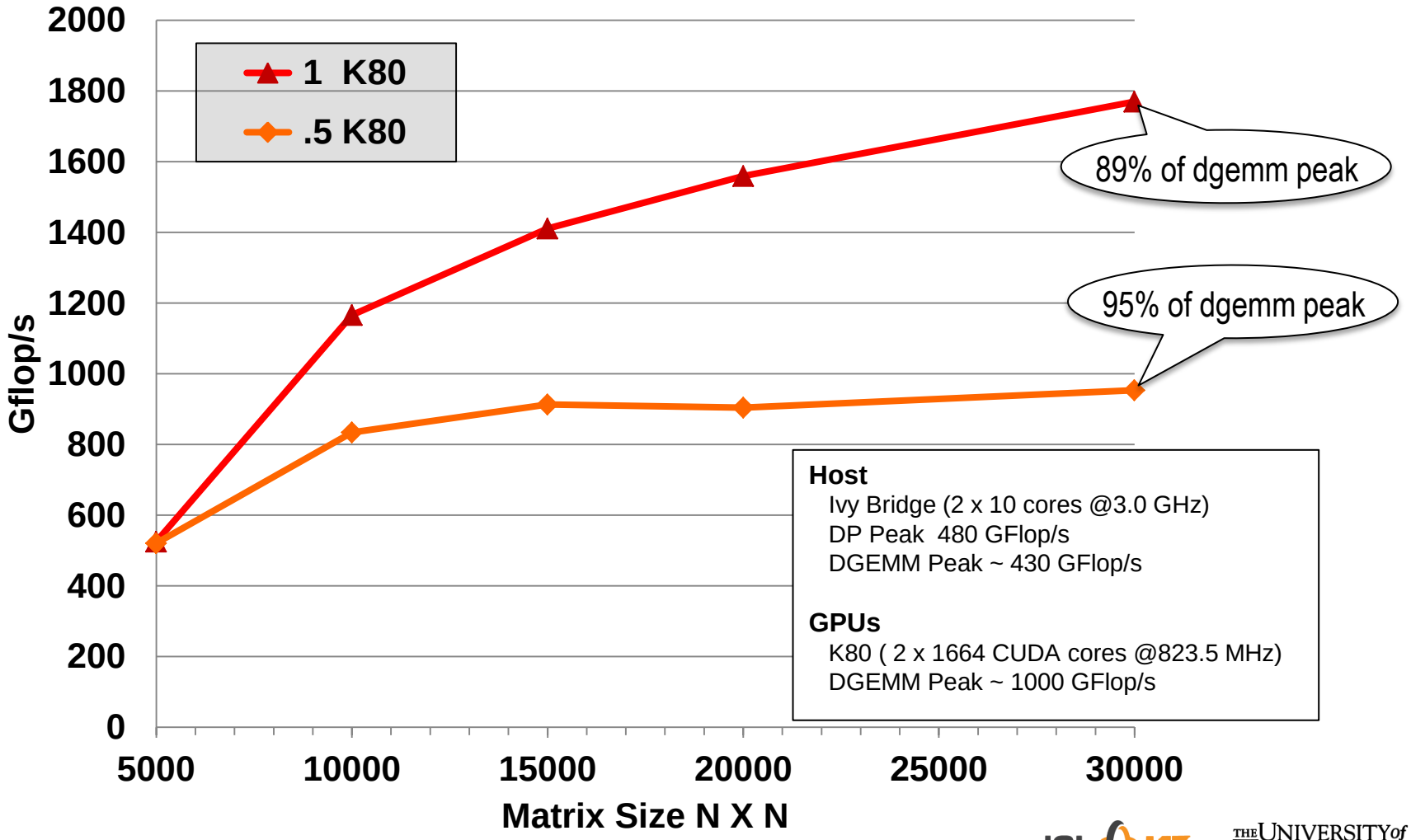


LU on Kepler in DP



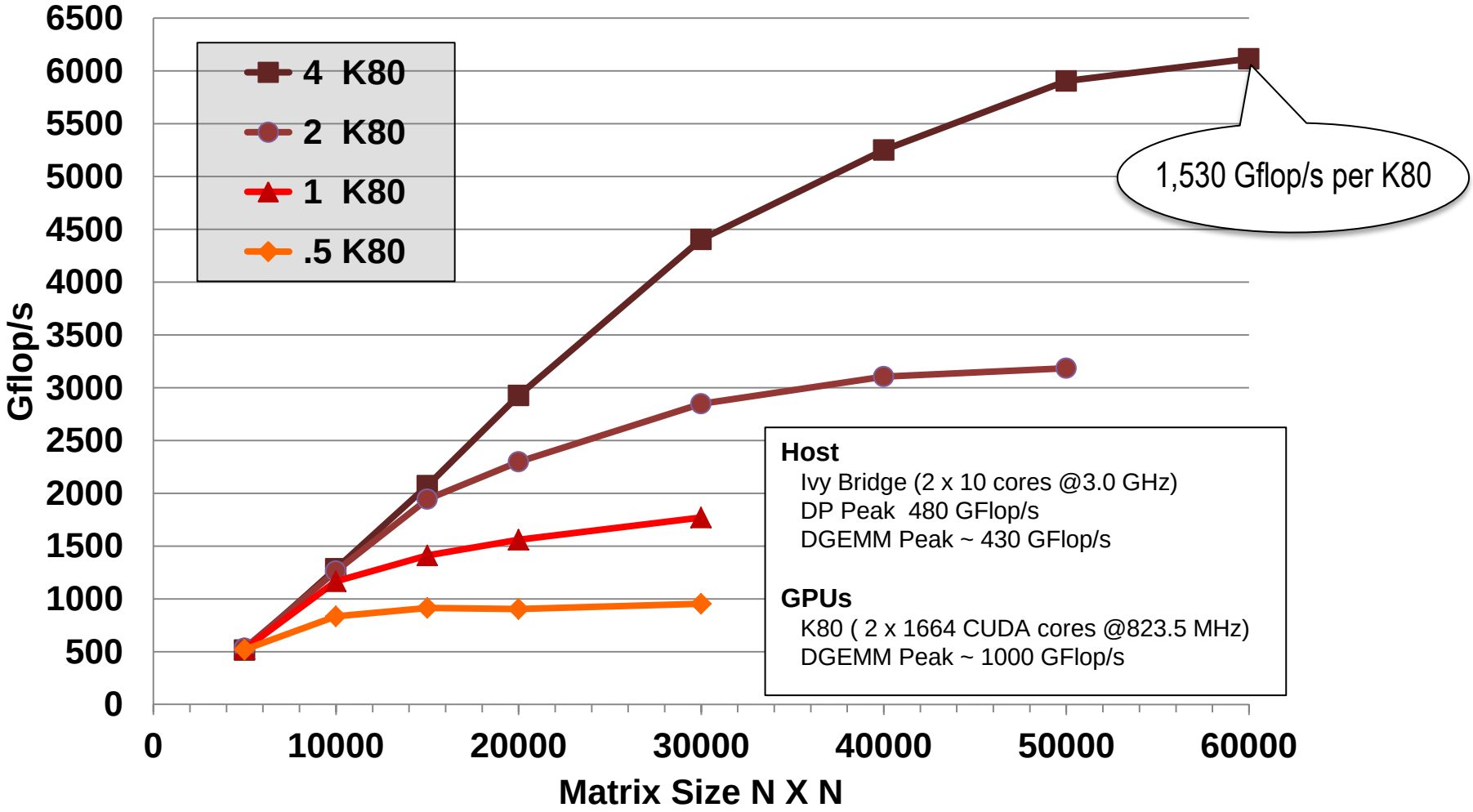
Support for K80 “GK210-Duo”

Performance of the MAGMA LU factorization in double precision



Support for K80 “GK210-Duo”

Performance of the MAGMA LU factorization in double precision

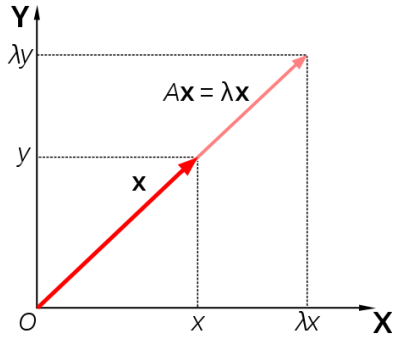


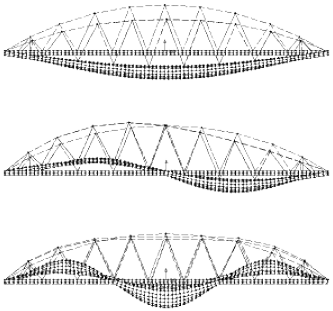
Eigenproblem Solvers in MAGMA

$A x = \lambda x$

- 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

. . .





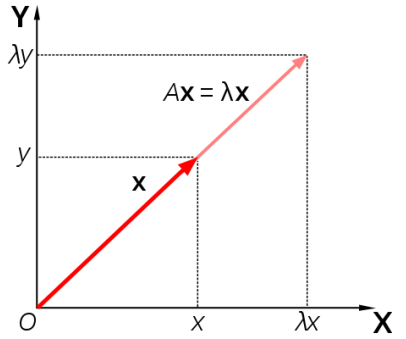
- Need to solve it fast

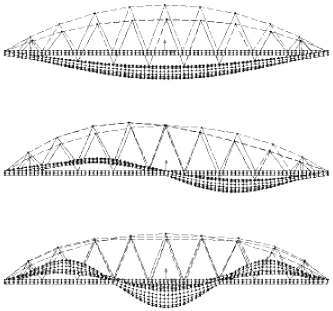
Eigenproblem Solvers in MAGMA

$A x = \lambda x$

- 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:

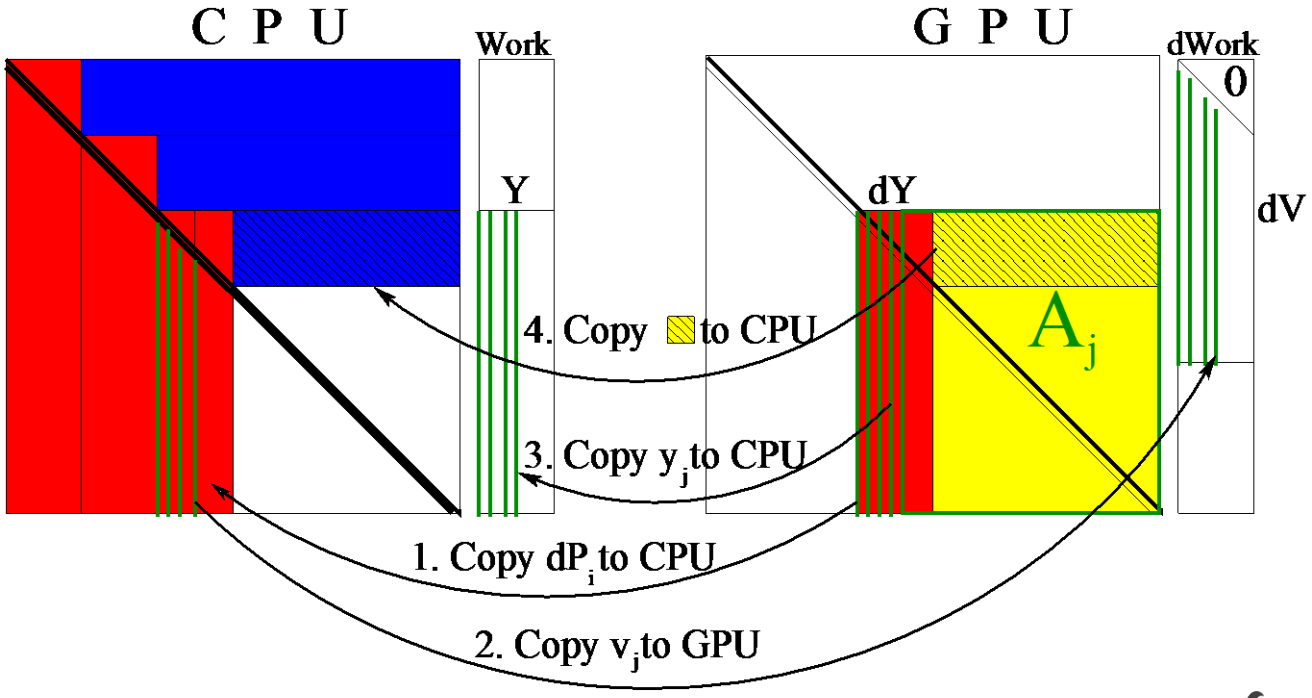
MAGMA with 1 GPU can be 12x faster vs. vendor libraries on state-of-art multicore systems

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.

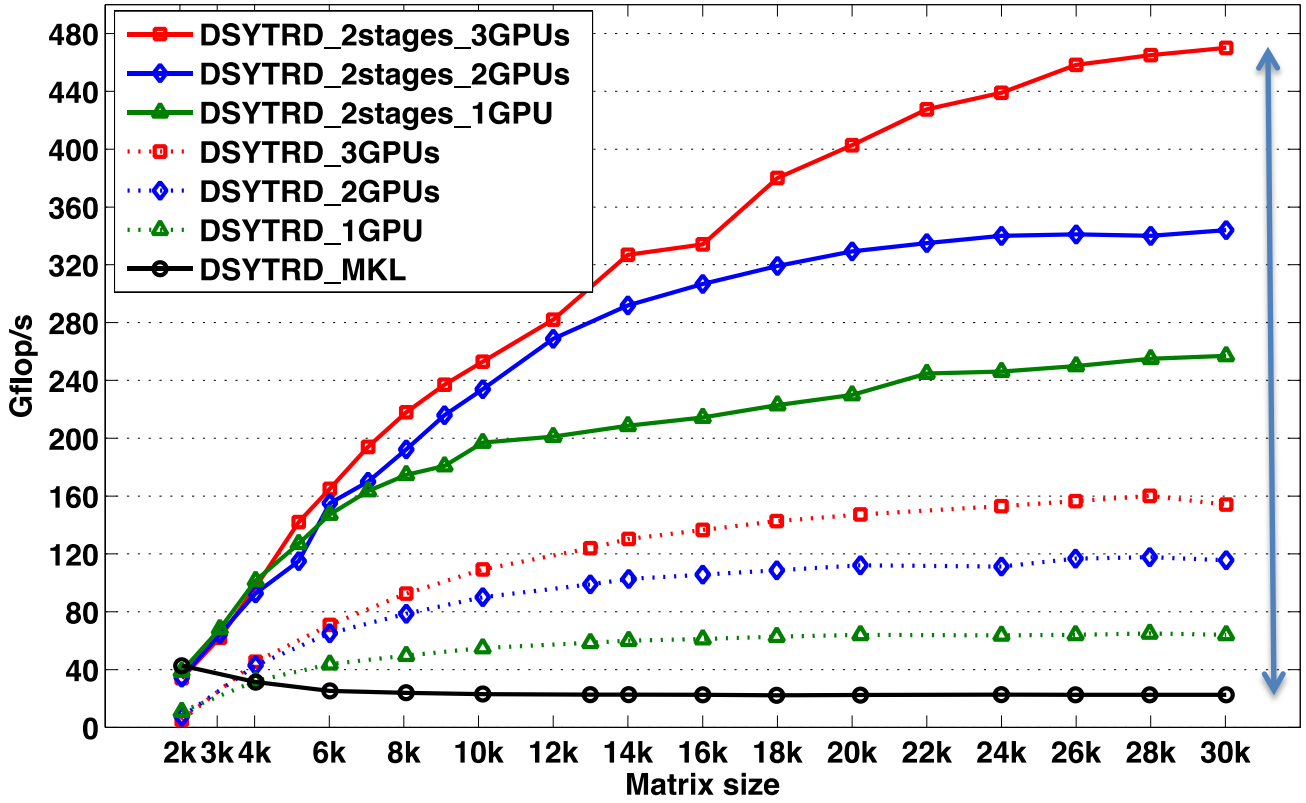
J. Dongarra, A. Haidar, T. Schulthess, R. Solca, and S. Tomov, *A novel hybrid CPU- GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks*, ICL Technical report, 03/2012.

MAGMA Two-sided Factorizations

- Panels are also hybrid, using both CPU and GPU (vs. just CPU as in the one-sided factorizations)
- Need fast Level 2 BLAS – use GPU's high bandwidth



Fast symmetric Eigensolvers

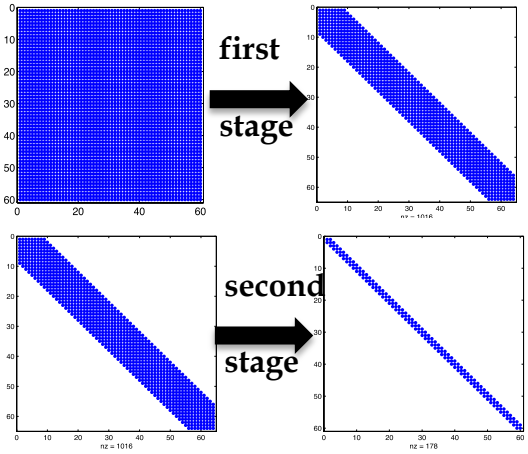


flops formula: $4n^3/3*time$

Higher is better

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

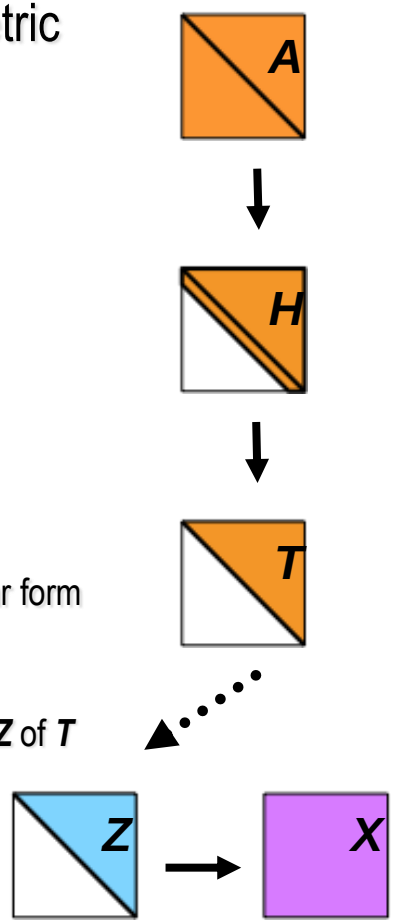


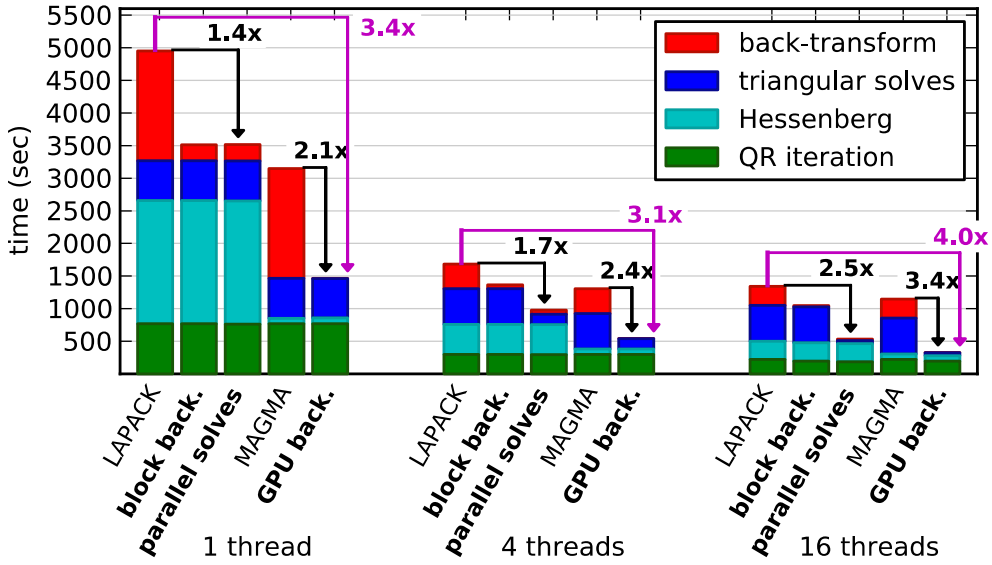
Characteristics

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

Fast non-symmetric Eigensolvers

- ***A*** is *n* x *n*, nonsymmetric
- $Ax = \lambda x$
- Three phases:
  - Hessenberg reduction
    $H = Q_1^T A Q_1$
  - QR iteration to triangular form
    $T = Q_2^T H Q_2$
  - Compute eigenvectors ***Z*** of ***T*** and back-transform to eigenvectors ***X*** of ***A***



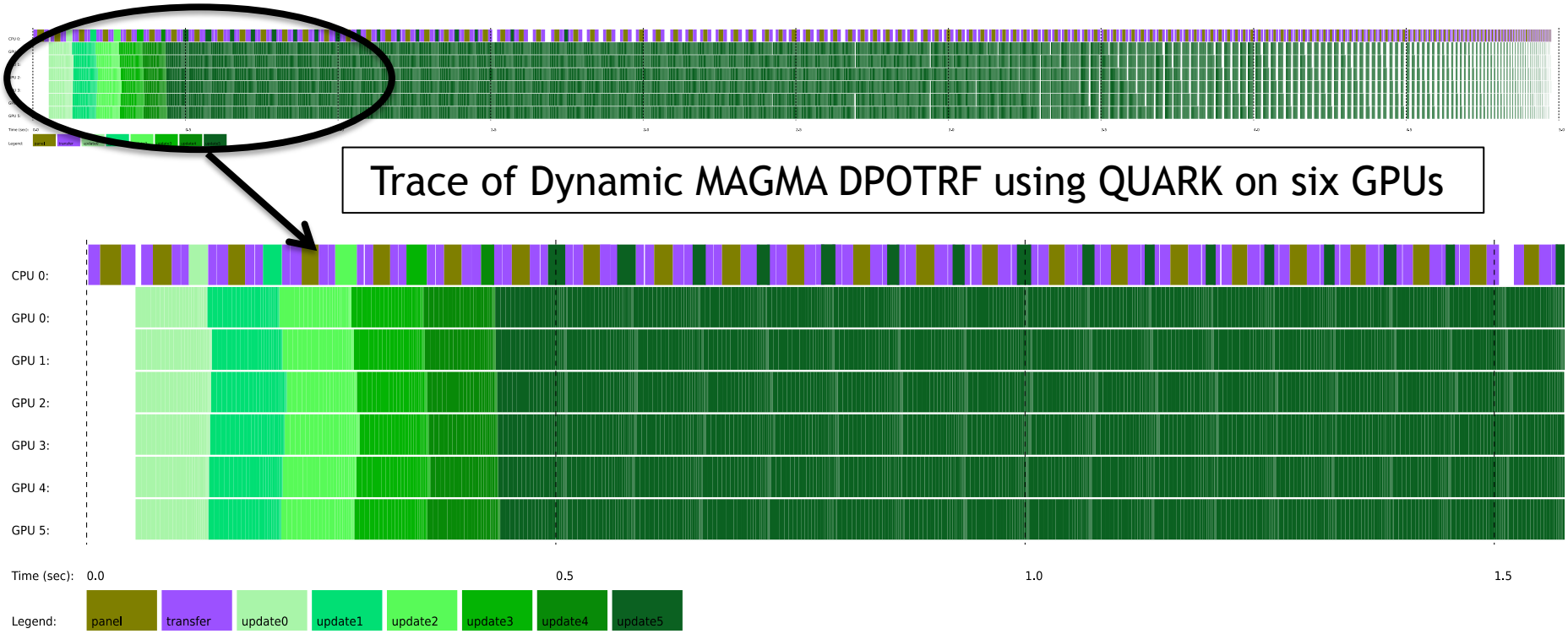


n = 16000,
- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU

• **M. Gates, A. Haidar, and J. Dongarra.** *Accelerating computation of eigenvectors in the nonsymmetric eigenvalue problem.* VecPar 2014 the 11th International Meeting High Performance Computing for Computational Science.

• **A. Haidar, M. Gates, S. Tomov, and J. Dongarra.** *Toward a scalable multi-GPU eigensolver via compute-intensive kernels and efficient communication.* International Conference on Supercomputing, ICS 2013, Oregon, USA

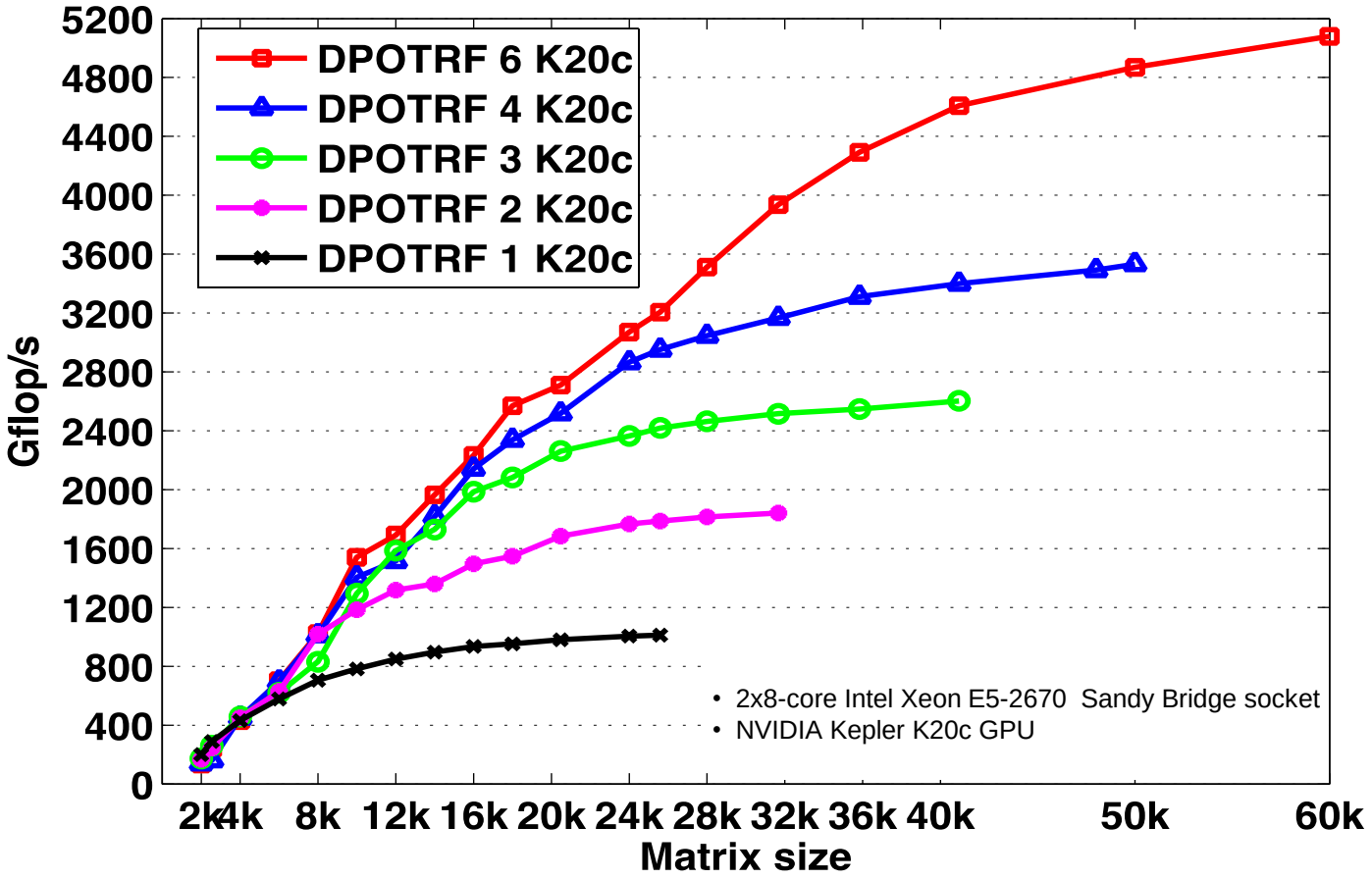
Dynamic MAGMA Scheduling



Scalability and efficiency :

- Snapshot of the execution trace of the Cholesky factorization on System A 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 GPU devices (six).

Dynamic MAGMA with QUARK

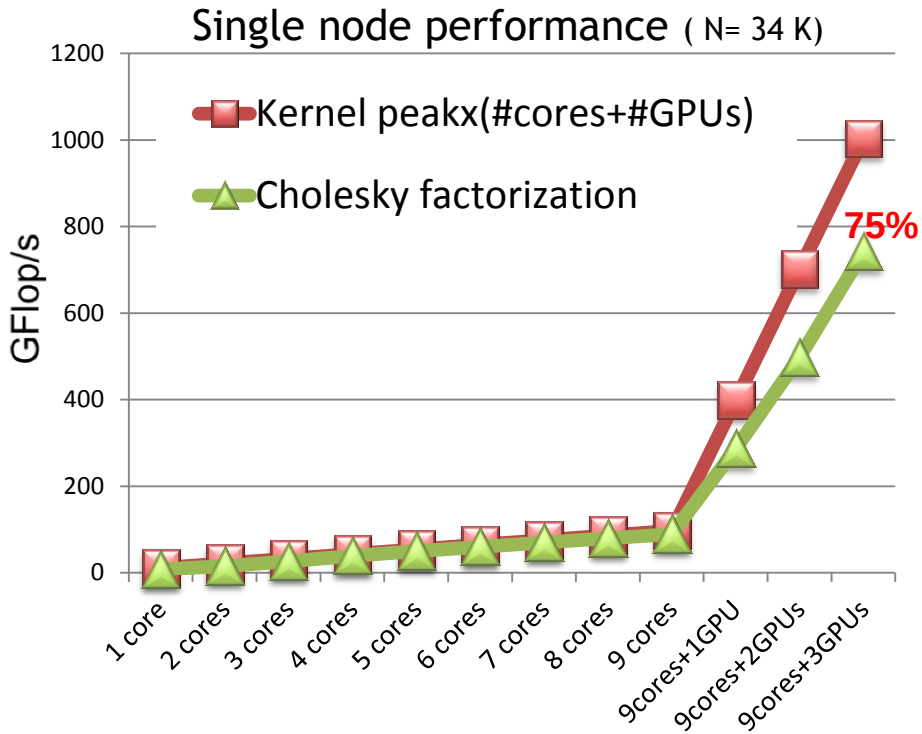


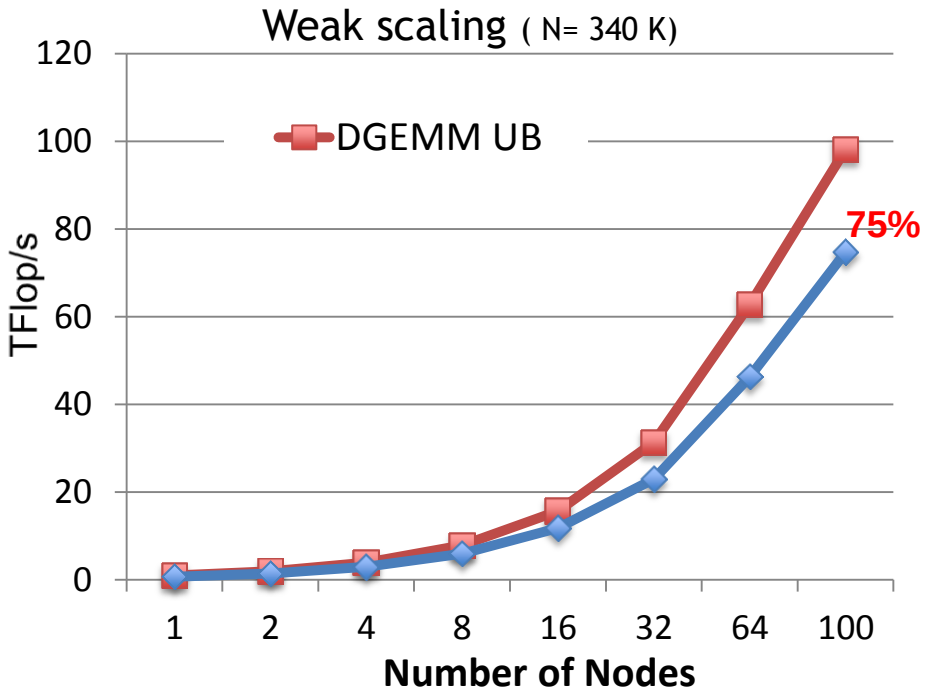
• **A. Haidar, C. Cao, A. YarKhan, P. Luszczek, S. Tomov, K. Kabir, and J. Dongarra.** *Unified Development for Mixed Multi-GPU and Multi-Coprocessor Environments using a Lightweight Runtime Environment.* IEEE IPDPS 2014, 28th IEEE International Parallel and Distributed Processing Symposium.

• **J. Dongarra, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and A. YarKhan,** *Model-Driven One-Sided Factorizations on Multicore Accelerated Systems.* Supercomputing frontiers and innovations journal 2014.

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

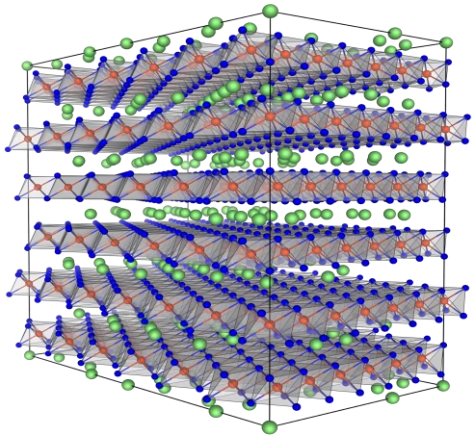




F. Song, S. Tomov, and J. Dongarra, *" Enabling and Scaling Matrix Computations on Heterogeneous Multi-Core and Multi-GPU Systems"*, ACM International Conference on Supercomputing (ICS 2012), San Servolo Island, Venice, Italy, June 2012.

A. Haidar, A. Yarkhan, C. Cao, P. Luszczek, S. Tomov, and J. Dongarra, , *"Flexible linear algebra development and scheduling with Cholesky factorization",* HPCC'15, New York, August 24-26, 2015.

Distributed-MAGMA Symmetric Eigensolvers with Thomas Schulthess CSCS



	Setup H20	Solve HC=4OC	The rest	total
28x28(2R:4T) ScaLAPACK	463.7	3839.7	61.6	4365.0
28x28(2R:4T) ELPA	471.7	1199.3	60.7	1731.7
14x14(1R:8T) MAGMA	166.7	911.9	79.9	1158.5

MAGMA is **1.5 times faster** than CPU

MAGMA is **2 times more energy efficient**

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.

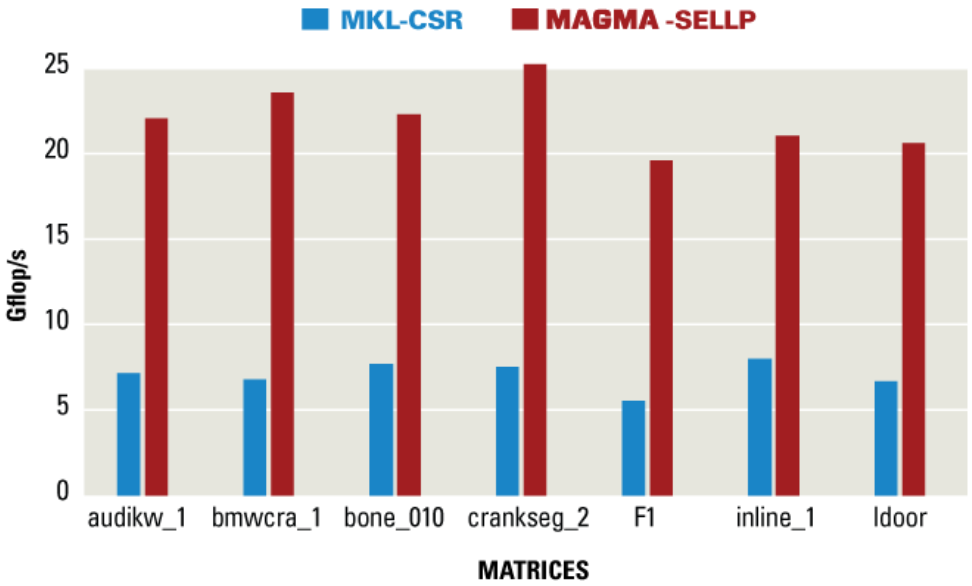
- **A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca,** *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks,* International Journal of High Performance Computing Applications , vol. 28, no 2, pp. 196—209, May 2014.

- **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.

MAGMA SPARSE

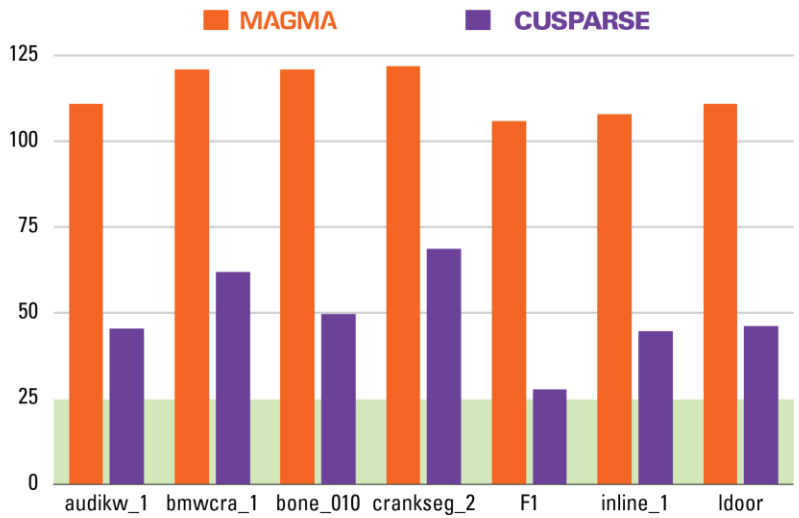
ROUTINES	CG, GMRES, BiCGSTAB, LOBPCG, Iterative refinement
PRECONDITIONERS	ILU / IC, Jacobi
KERNELS	SpMV, SpMM
DATA FORMATS	CSR, ELL, SELL-P

Sparse matrix - vector product (SpMV) in double precision arithmetic



The University of Florida Sparse Matrix Collection http://www.cise.ufl.edu/research/sparse/matrices/

SpMM on block of 32 vectors in double precision



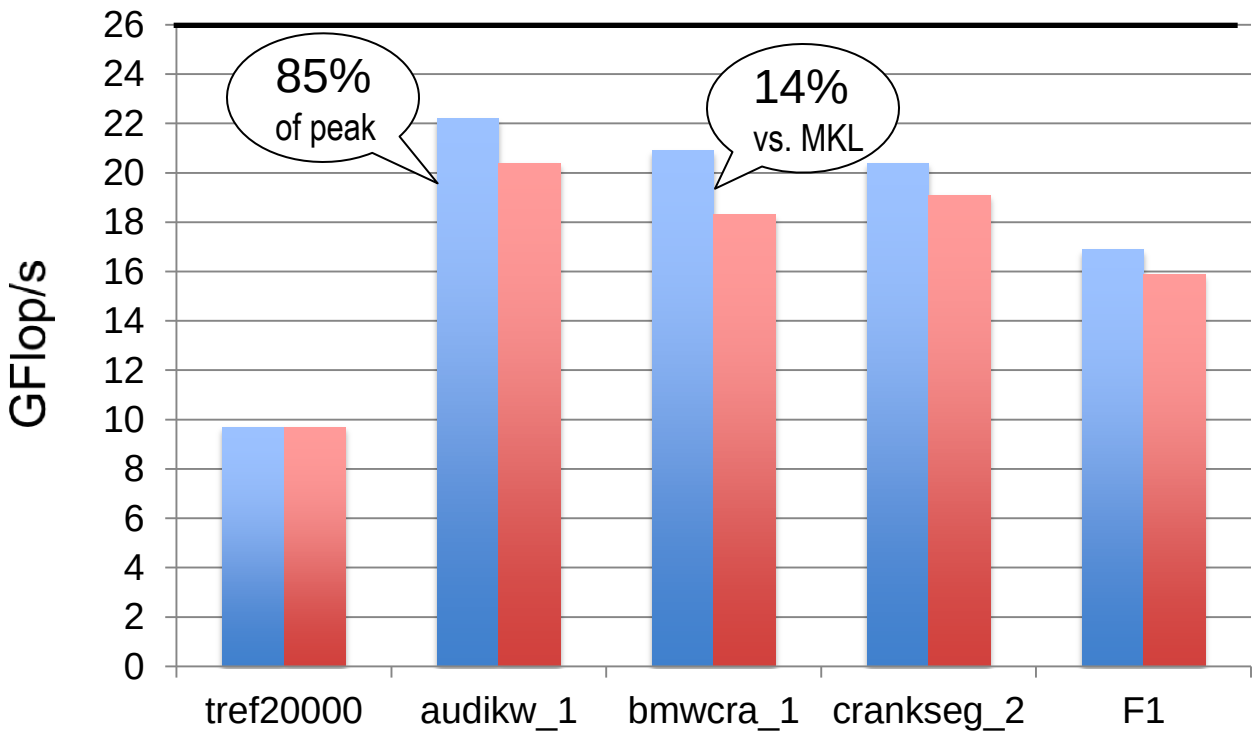
GPU NVIDIA K40 (Atlas)
15 MP x 192 @ 0.88 GHz

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

MAGMA MIC Sparse

FEATURES AND SUPPORT in MAGMA MIC 1.4	
ROUTINES	CG, GMRES, BiCGSTAB, Iterative Refinement
PRECONDITIONERS	Jacobi, user defined
KERNELS	SpMV, SpMM
DATA FORMATS	CSR, CPU converters from/to ELL and SELL-P
BENCHMARKS	CG, GMRES, BiCGSTAN, SpMV, SpMM, BLAS

PERFORMANCE (in double precision)



P_{max} ~ 26 GFlop/s

[based on achievable bandwidth limit of about 160 GB/s]

MAGMA SpMV

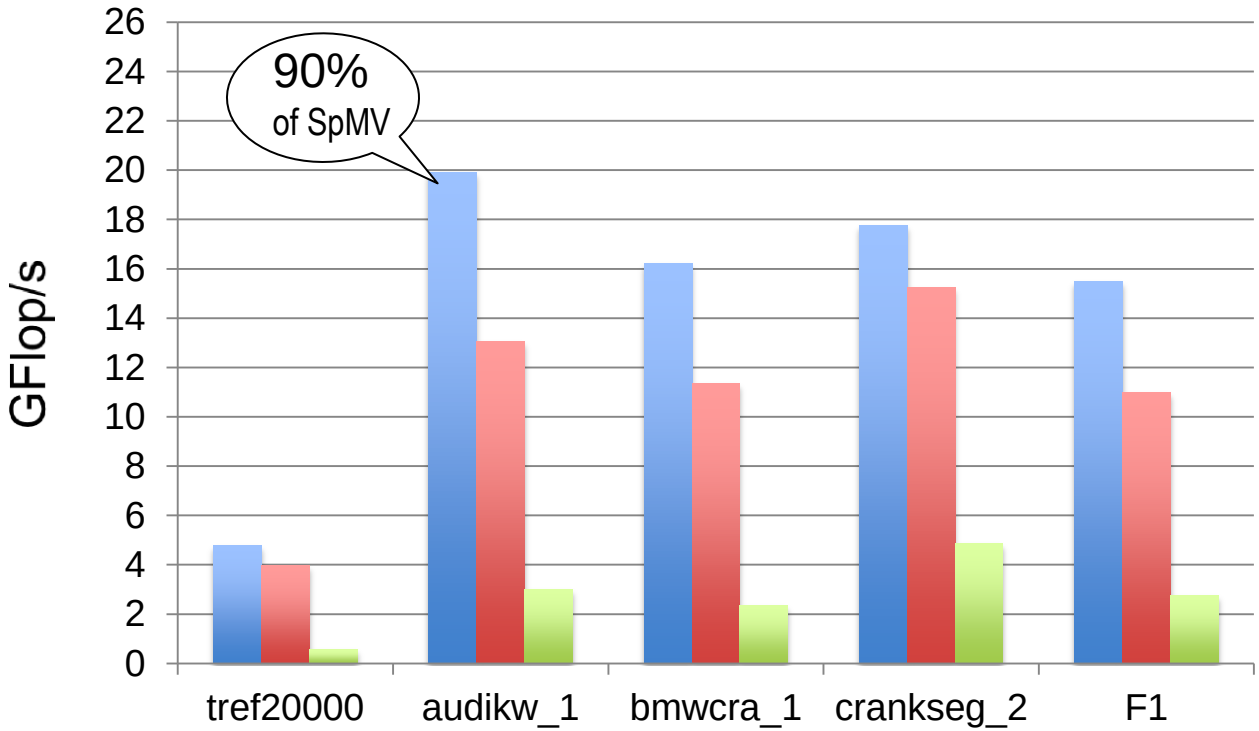
MKL SpMV

Coprocessor
Intel Xeon Phi (60 @ 1.09 GHz)
DP Peak 1046 GFlop/s

MAGMA MIC Sparse

FEATURES AND SUPPORT in MAGMA MIC 1.4	
ROUTINES	CG, GMRES, BiCGSTAB, Iterative Refinement
PRECONDITIONERS	Jacobi, user defined
KERNELS	SpMV, SpMM
DATA FORMATS	CSR, CPU converters from/to ELL and SELL-P
BENCHMARKS	CG, GMRES, BiCGSTAN, SpMV, SpMM, BLAS

PERFORMANCE (in double precision)

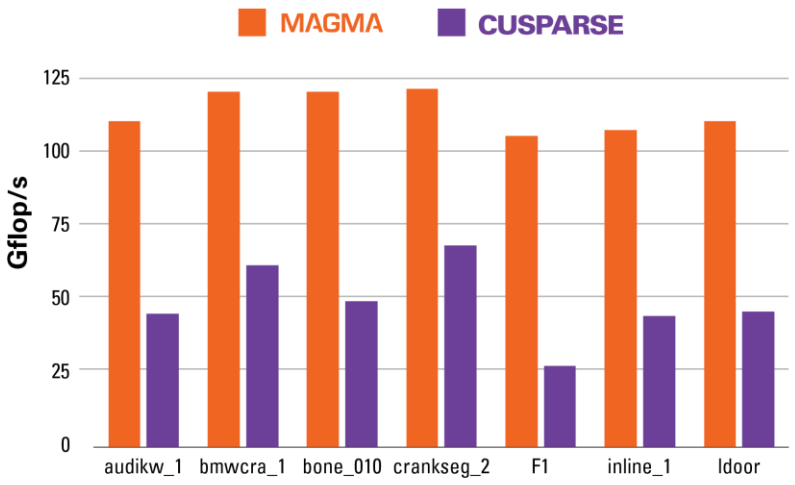


Coprocessor
Intel Xeon Phi (60 @ 1.09 GHz)
DP Peak 1046 GFlop/s

GPU-aware Sparse Eigen-Solvers

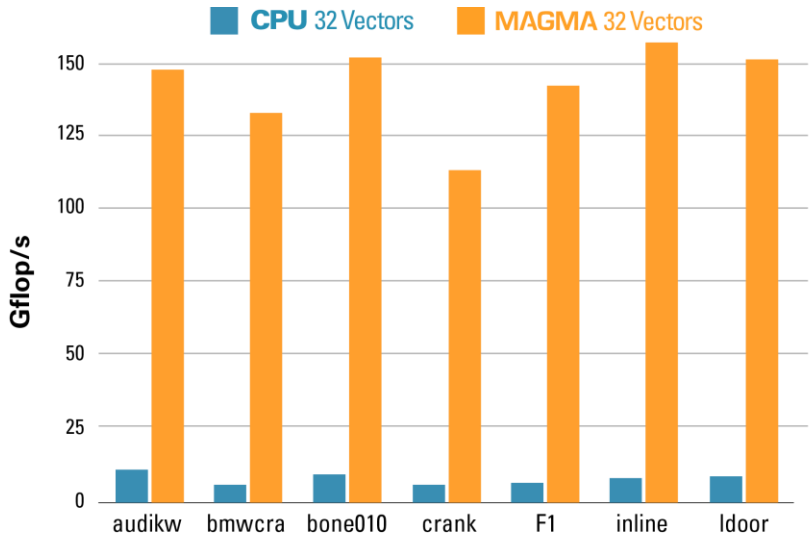
- Locally Optimal Block PCG (LOBPCG)
  - Find a set of smallest eigen-states of a sparse SPD matrix ($A x = \lambda x$)
  - Replace finding the states one-by-one by a block algorithm
    - finds them simultaneously; needs fast SpMM, re-orthogonalization, and GEMM of particular sizes

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



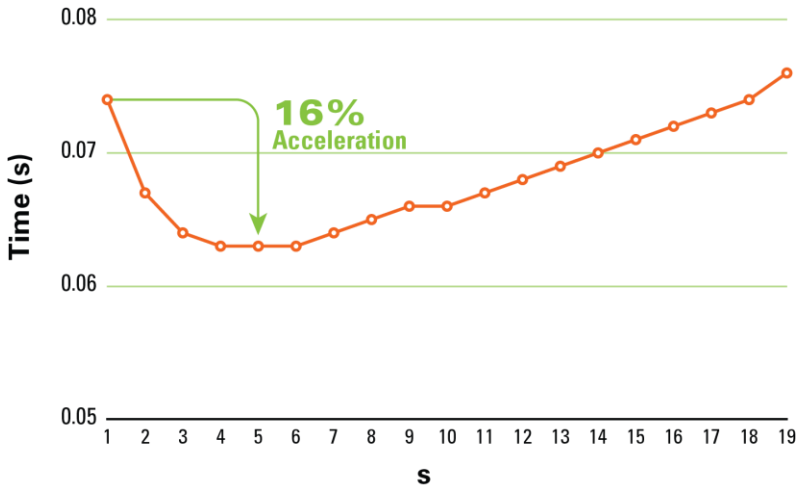
BLOPEX LOBPCG: uses CPU MAGMA Sparse: uses CPU

Algorithms to avoid communication

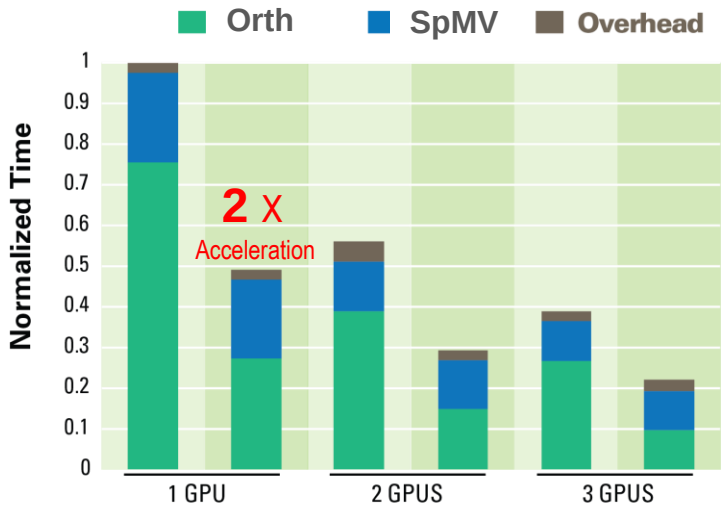
- Communication avoiding GMRES (CA-GMRES)
  - (1) Replacing GMRES' SpMV ➔ Matrix Powers Kernel (MPK):

    $v_{k+1} = A\, v_k$ for k = j, ... , j+s
  - (2) BLAS-2 ➔ BLAS-3 based orthogonalization (next …)

MPK to generate 100 vectors for various s



Overall performance improvement on up to 3 GPUs



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, Phoenix, Arizona, May 19—23, 2014.

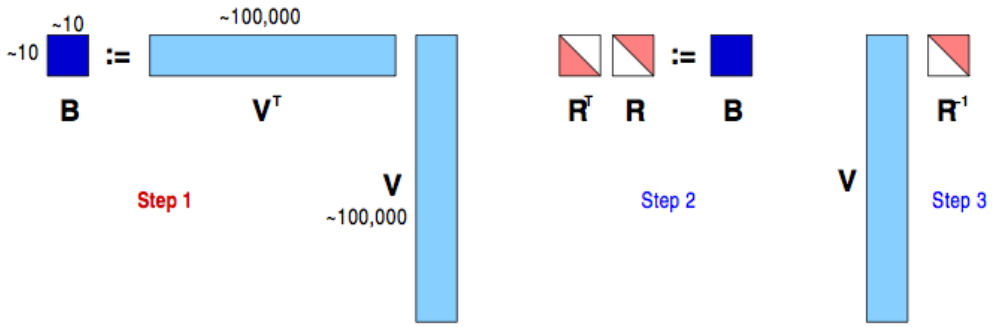
Orthogonalization procedures

- 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.

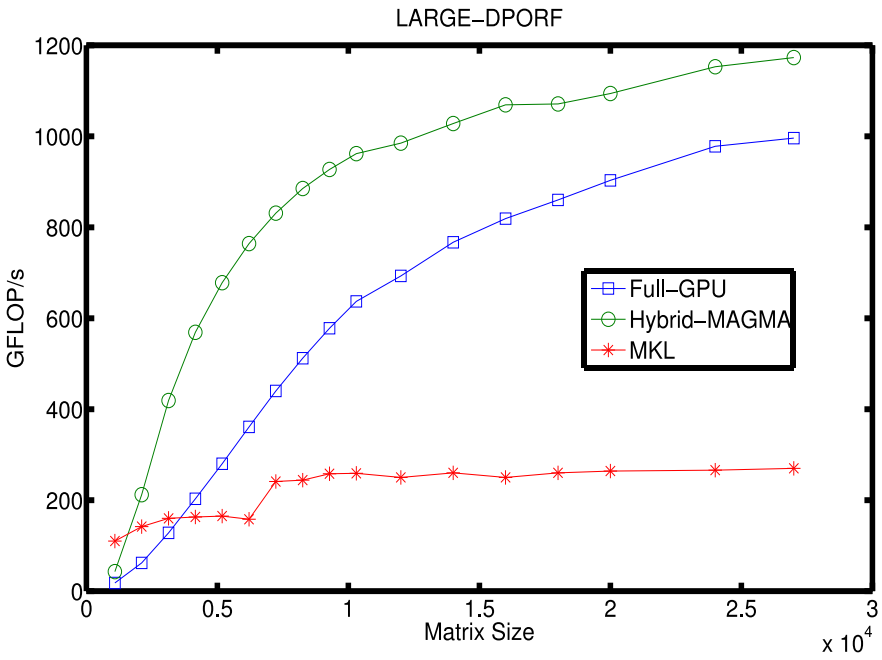
Step 3 Backward-substitution $Q := VR^{-1}$
on GPUs in standard-precision.



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

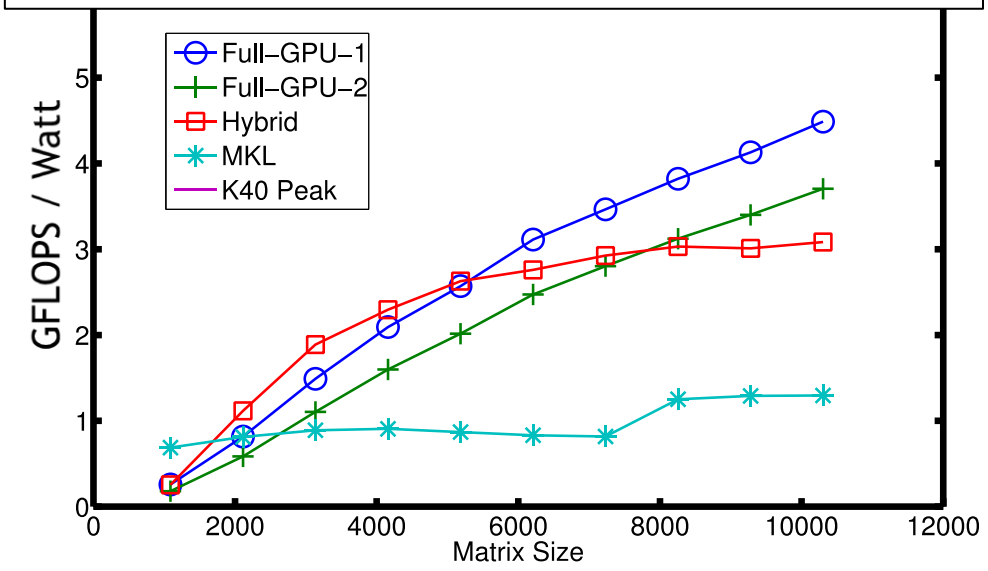
GPU only routines in MAGMA

Motivation: **power** and performance advantages



GPU: NVIDIA K40c
15 MP x 192 @ 0.88 GHz
DP Peak: 1690 GFlop/s

CPU: Intel Sandy Bridge
2 x 8 cores @ 2.6 GHz
DP Peak: 332.8 Gflop/s

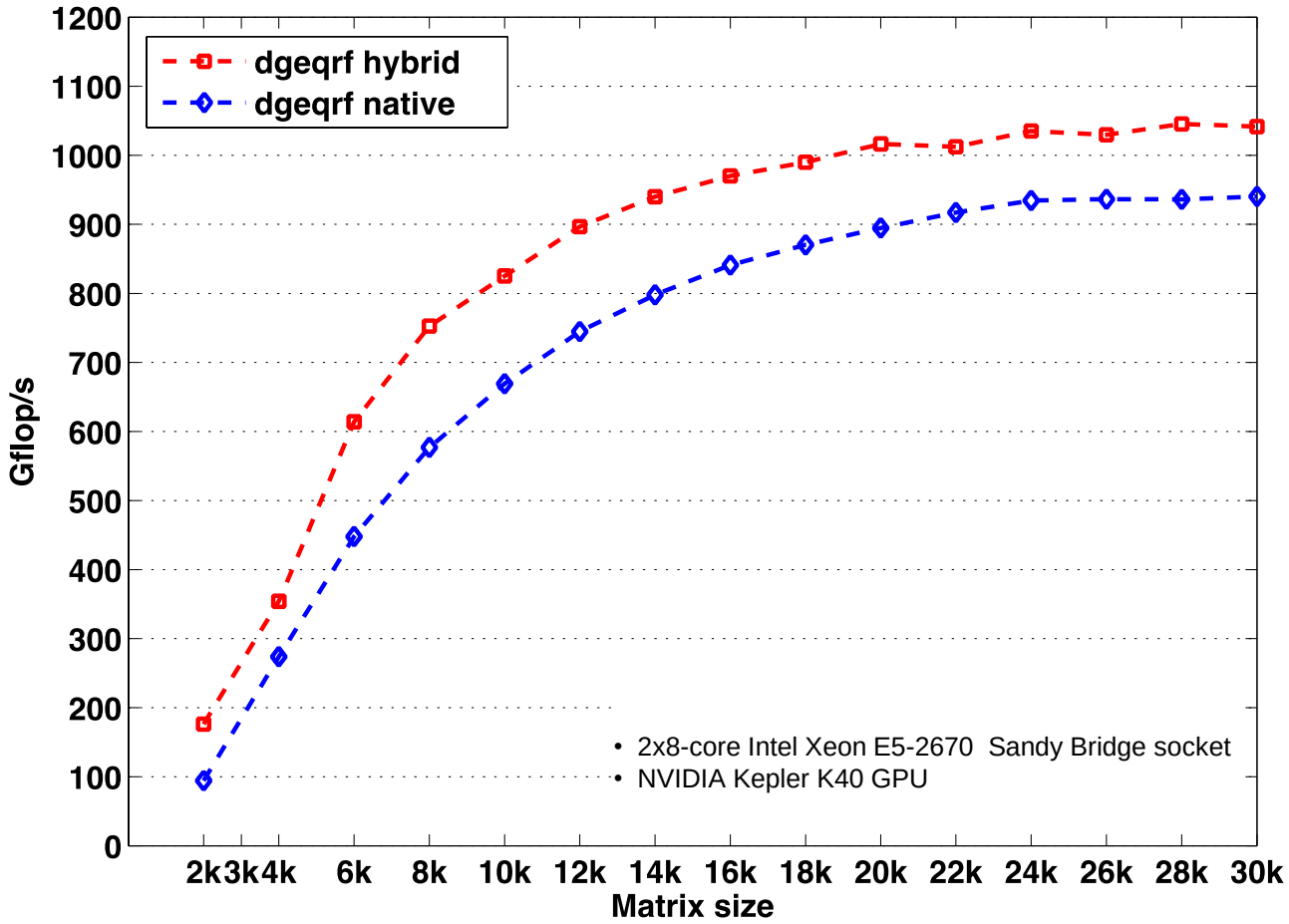


- Power consumptions of a K40c is about the same as 2 x 8 Sandy Bridge cores
- But the K40c is about 3 to 4 times faster, making it 3 to 4 times more energy efficient

=> to minimize energy consumption, use the entirely GPU implementation

GPU only routines in MAGMA

MAGMA QR in double precision on a K40 GPU: GPU only vs. Hybrid

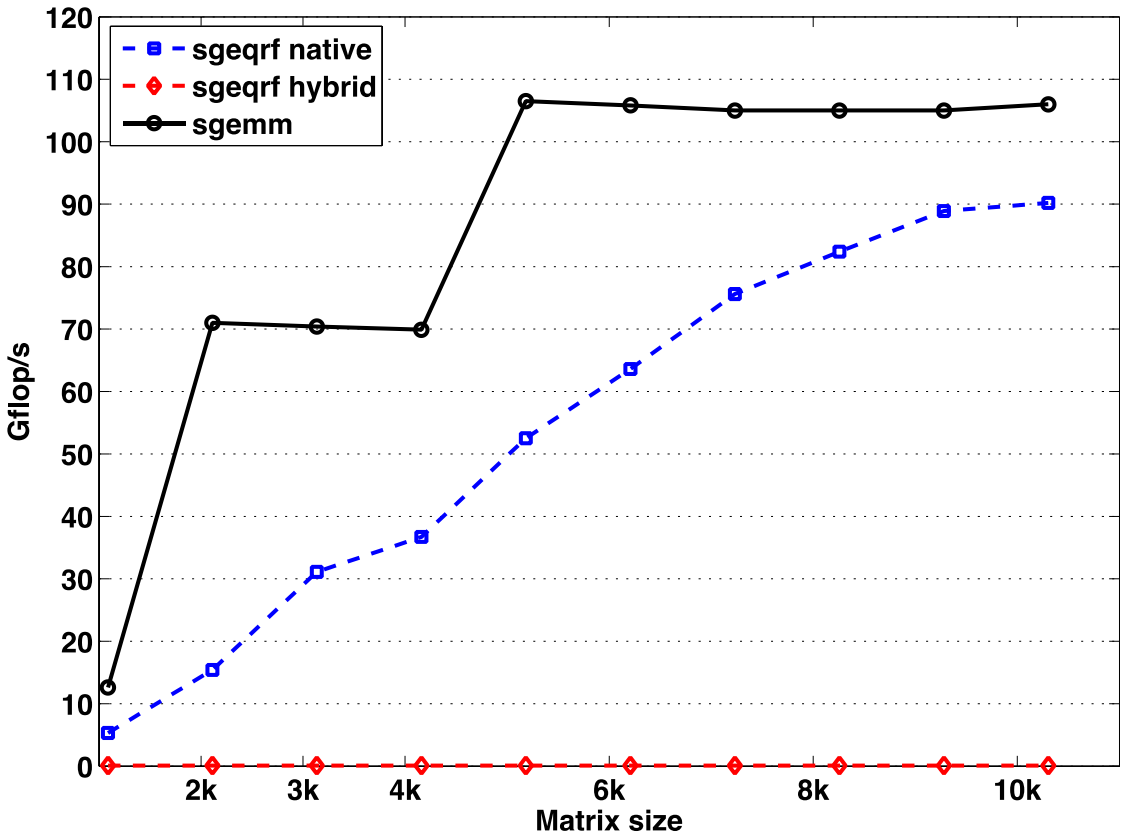


GPU only routines in MAGMA

Motivation: power and **performance** advantages

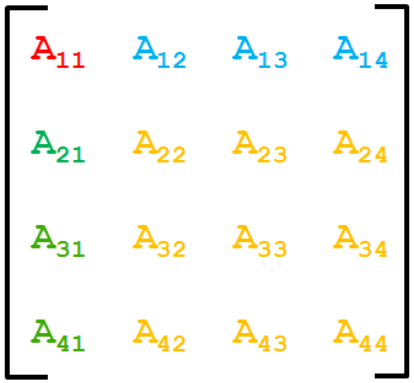
MAGMA Embedded

QR factorization in single precision on the Jetson TK1

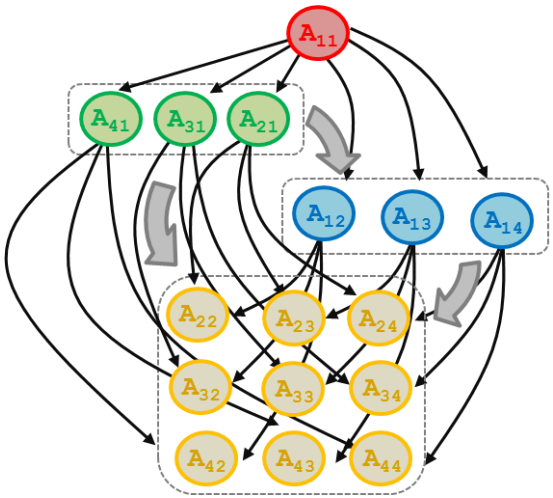


MAGMA Batched Computations

Sparse / Dense Matrix System



DAG-based factorization



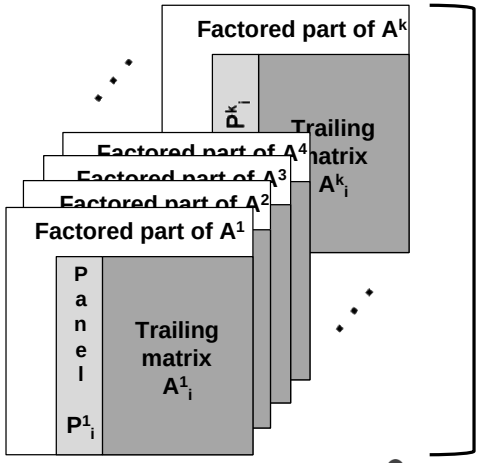
To capture main LA patterns needed in a numerical library for Batched LA

- **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

And many other BLAS/LAPACK, e.g., for application specific solvers, preconditioners, and matrices

Motivation: factorization of thousands of small matrices

- Structural mechanics
- Astrophysics
- High order FEM
- Tensor contractions
- Sparse direct solver
- Hydrodynamics
- Image processing
- Ranking and recommender systems, etc

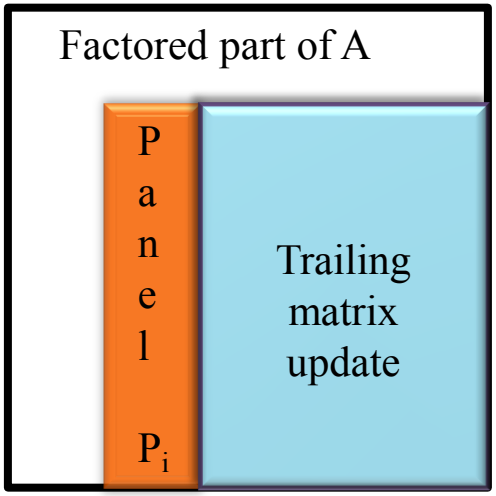


Batched factorization of a set of k matrices $A^1, A^2, \ldots, A^k$

MAGMA Batched Computations

Algorithmic basics:

- Linear solver *Ax=b* follow the LAPACK-style algorithmic design blocking algorithm
- Two distinctive phases
  - panel factorization: latency-bound workload
  - trailing matrix update: compute-bound operation



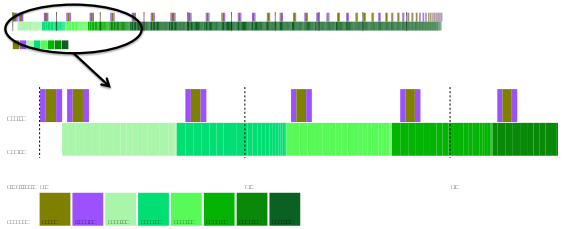
Hardware characteristics and limit:

- GPU memory is limited (48KB of shared per SMX, limited number of register)
- Prefer implementation that extensively uses large number of thread/block (a warp is 32 threads)
- Prefer coalescent memory access (32 threads can read in parallel 32 elements)

MAGMA Batched Computations

Classical strategies design

- For standard problems the strategy is to prioritize the data-intensive operations to be executed by the accelerator and keep the memory-bound ones for the CPUs since the hierarchical caches are more appropriate to handle it



Difficulties

- Cannot be used here since matrices are very small and communication becomes expensive

Proposition

- Go on and have a native GPU implementation

MAGMA Batched Computations

Classical strategies design

- For standard problem performance is driven by the update operations,

Difficulties

- For batched small matrices it is more complicated and requires both phases to be efficient

Proposition

- Redesign both phases in a tuned efficient way

MAGMA Batched Computations

Classical strategies design

- A recommended way of writing efficient GPU kernels is to use the GPU's shared memory – load it with data and reuse that data in computations as much as possible.

Difficulties

- Our study and experience shows that this procedure provides very good performance for classical GPU kernels but is not that appealing for batched algorithm for different reasons.

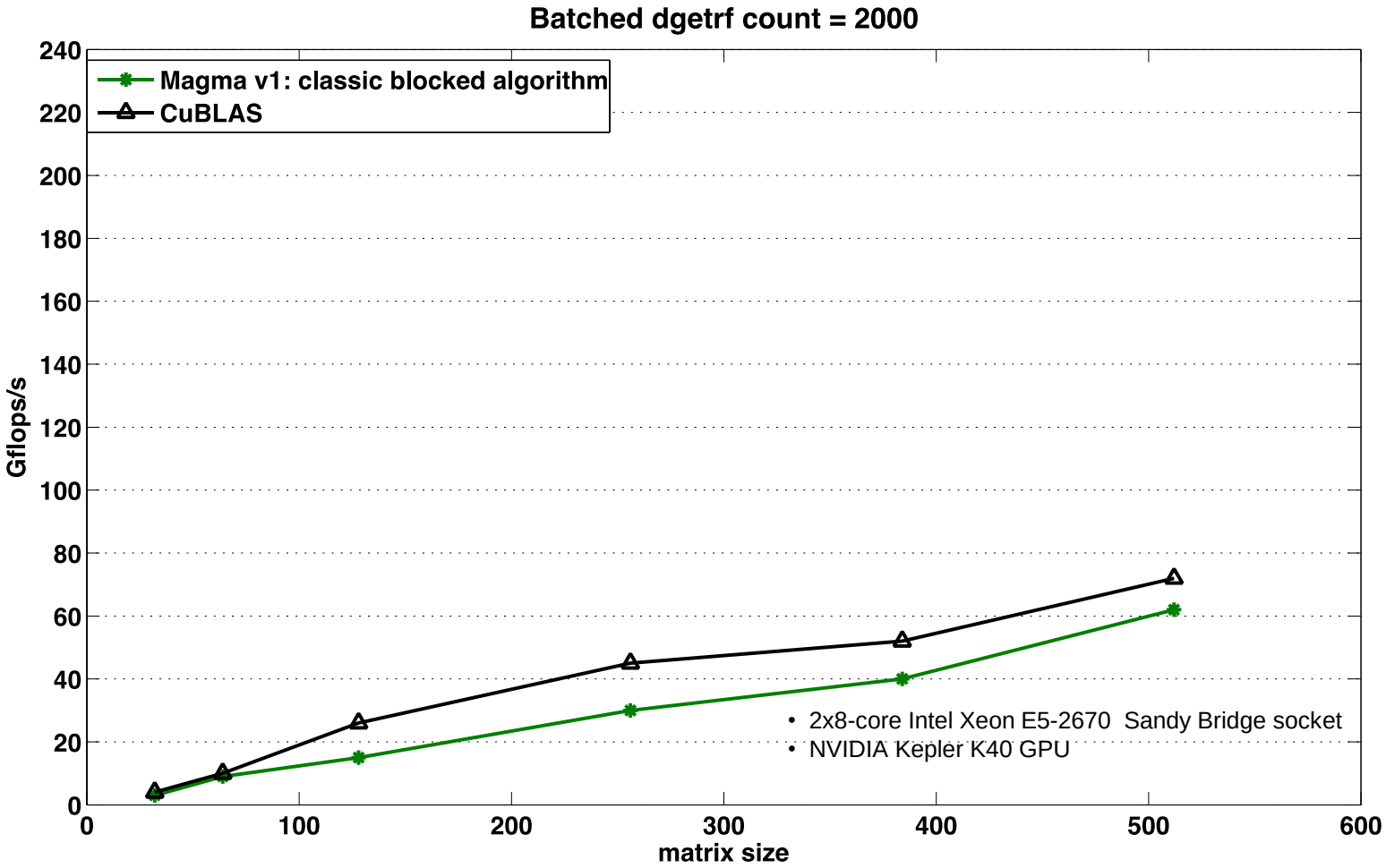
MAGMA Batched Computations

Difficulties

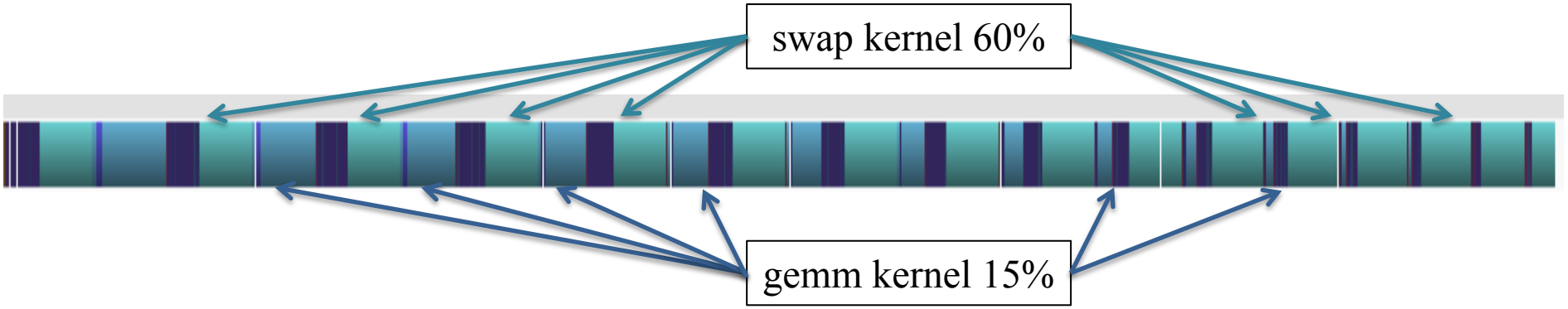
- Completely saturating the shared memory per SMX can decrease the performance of memory bound operations, since only one thread-block will be mapped to that SMX at a time (low occupancy)
- due to a limited parallelism in the panel computation, the number of threads used in the thread block will be limited, resulting in low occupancy, and subsequently poor core utilization
- Shared memory is small (48KB/SMX) to fit the whole panel
- The panel computation involves different type of operations:
  - Vectors column (find the max, scale, norm, reduction)
  - Row interchanges (swap)
  - Small number of vectors (apply)

Proposition: **custom design per operations type**

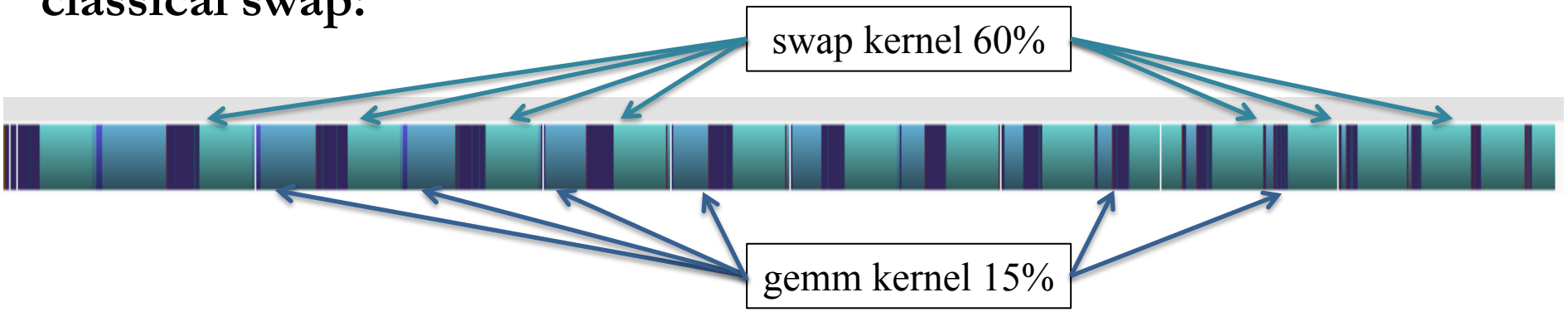
MAGMA Batched Computations



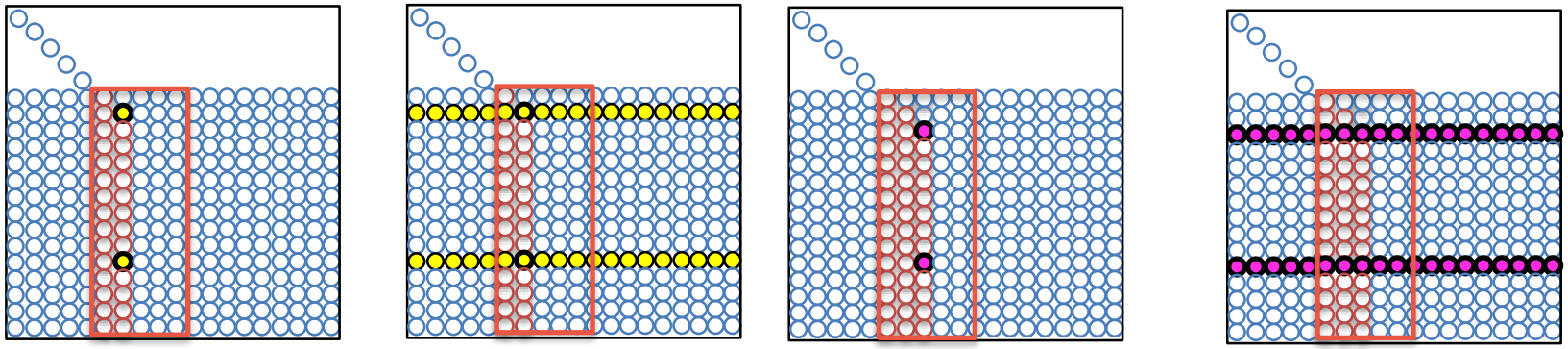
MAGMA Batched Computations



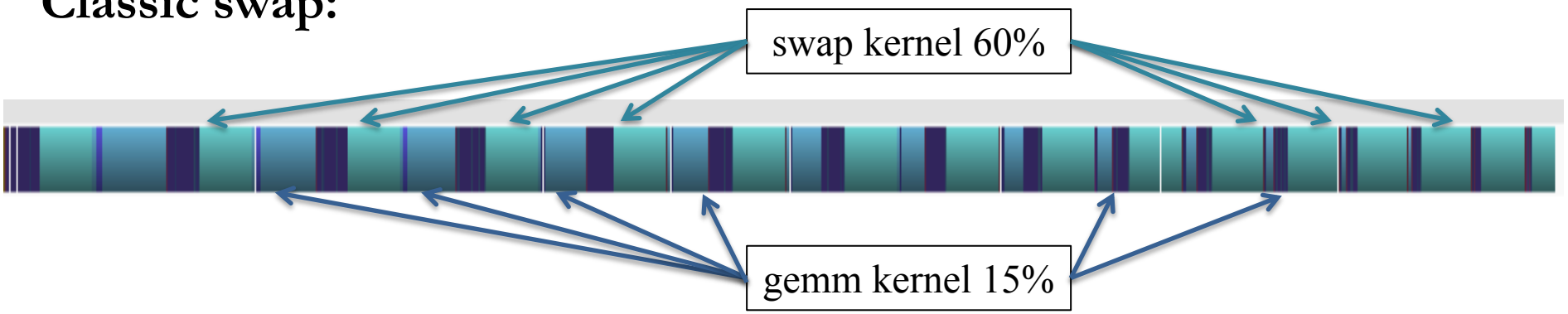
classical swap:



How does the swap work?



Classic swap:



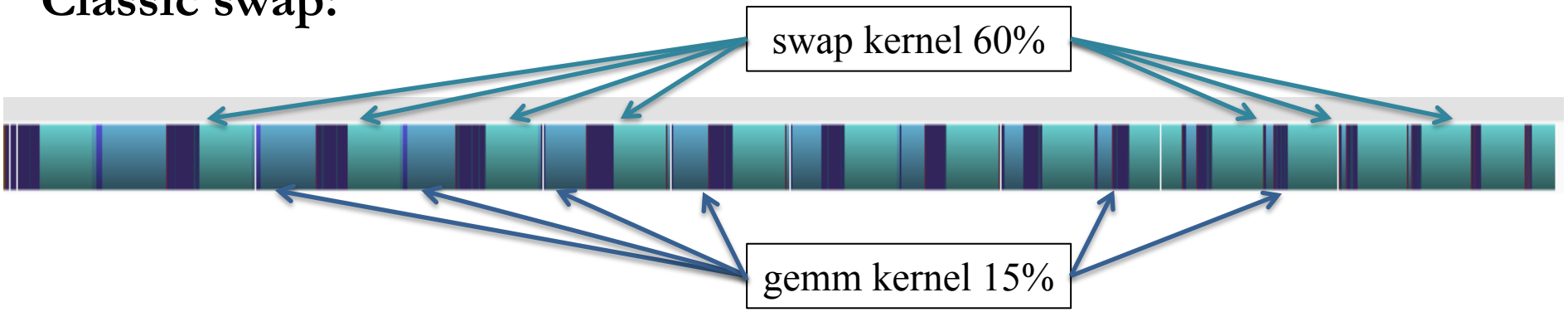
Bottlenecks:

- The swapping consists of *nb* successive interchanges of two rows of the matrices (serial).
- Data reading is not coalescent: a GPU warp cannot read 32 value at the same time unless matrix is stored in transpose form. However if matrix is stored in transpose form the swap is fast BUT the other components become very slow.

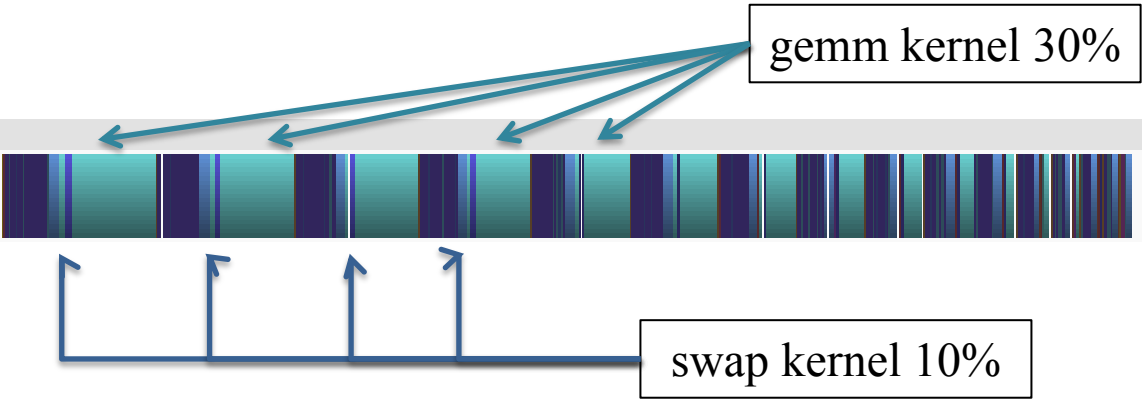
Proposition:

- We propose to modify the kernel to apply all ***nb*** row swaps in parallel
- This modification will also allow the coalescent write back of the top ***nb*** rows of the matrix
- Note that the top ***nb*** rows are those used by the ***dtrsm*** kernel that is applied right after the ***dlaswp***, so one optimization is to use shared memory to load a chunk of the ***nb*** rows

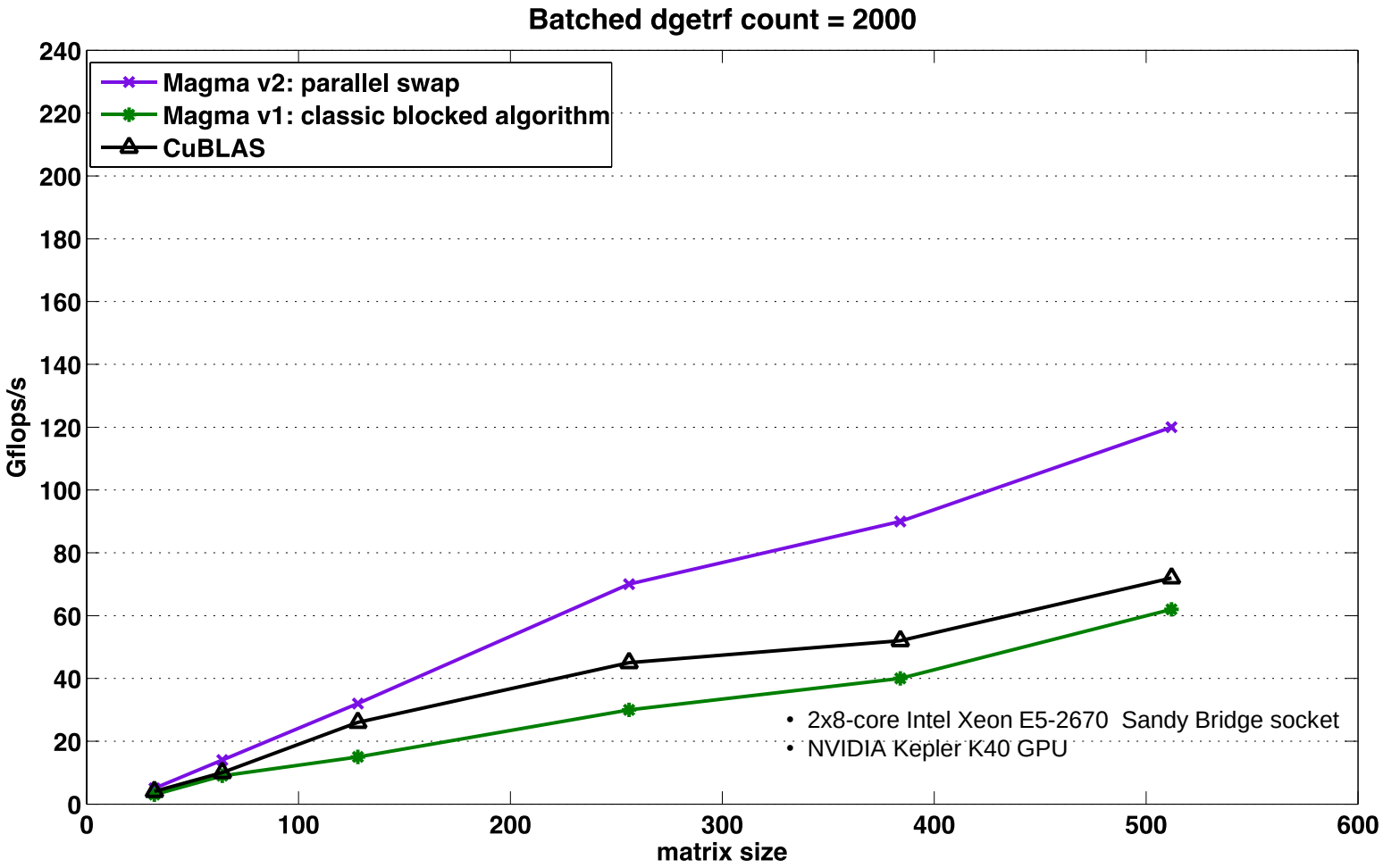
Classic swap:



Parallel swap:



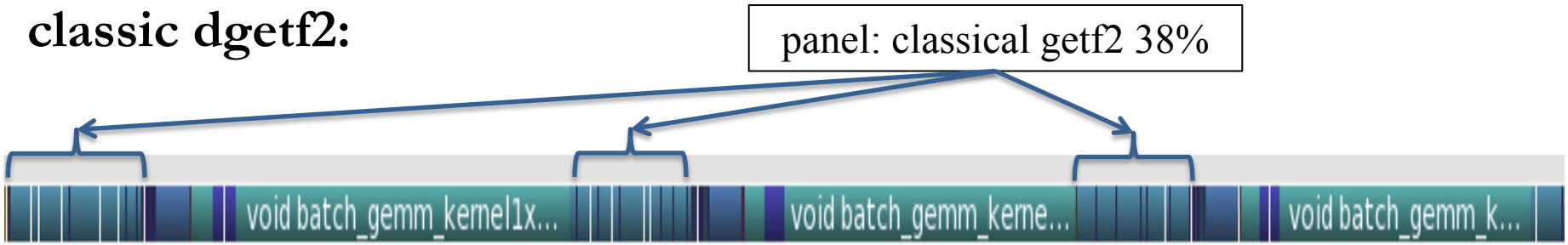
MAGMA Batched Computations

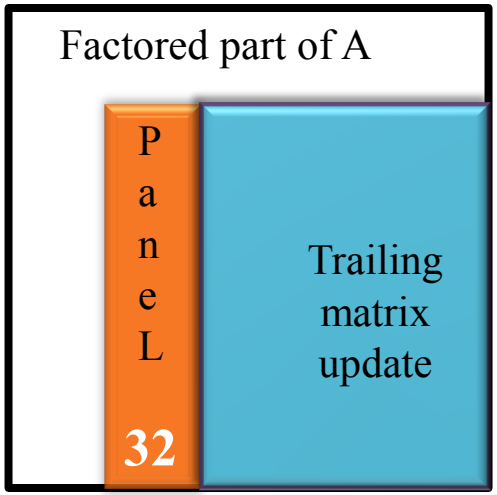


MAGMA Batched Computations

Panel factorization

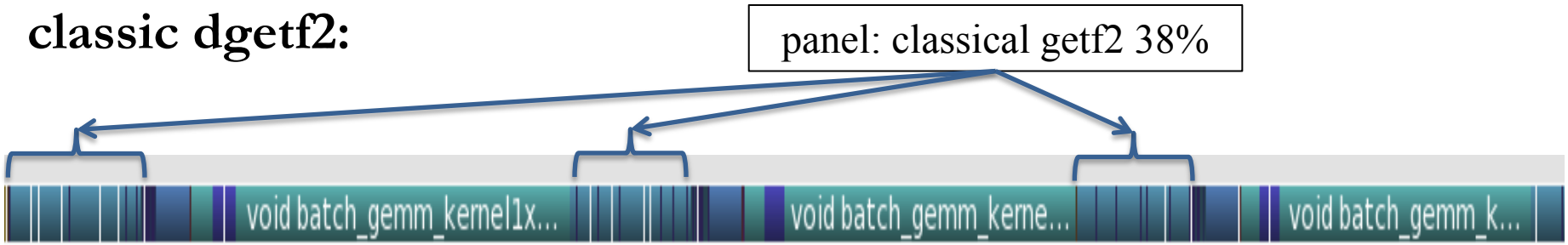
classic dgetf2:

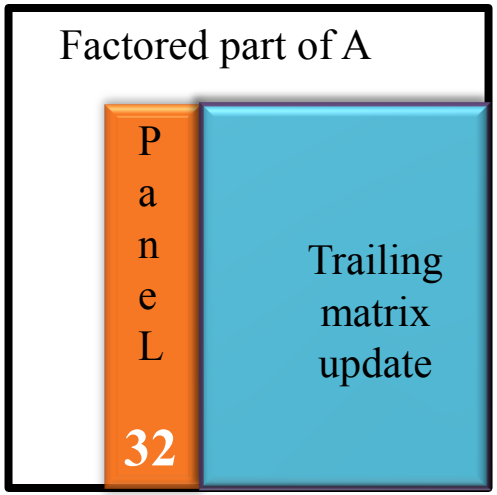




MAGMA Batched Computations

Panel factorization
classic dgetf2:





Bottlenecks:

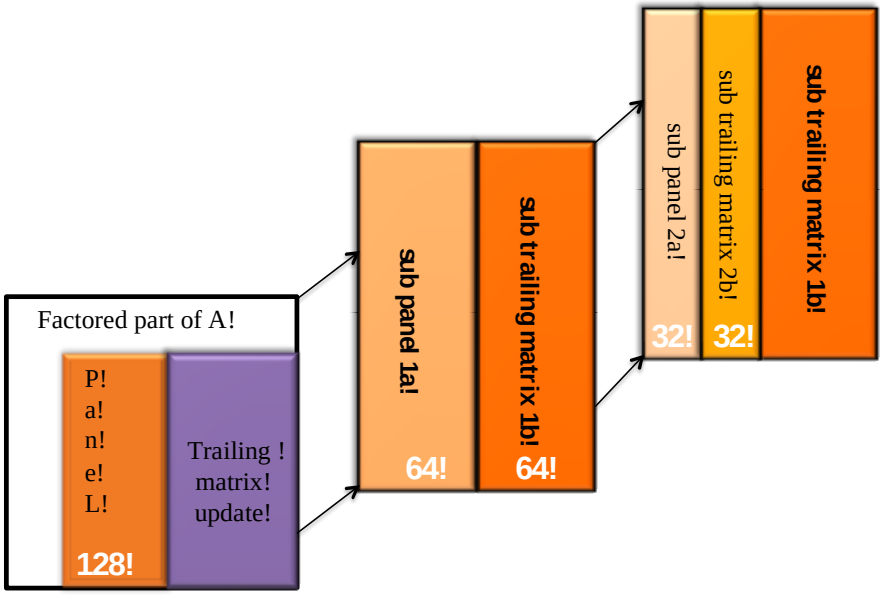
- *nb* large: panel get slower
  --> very bad performance.
- *nb* small: panel get faster but the update is not anymore efficient since dealing with gemm's of small sizes
  --> very bad performance.
- trade-off ? No effect, since we are talking about small size.

Proposition:

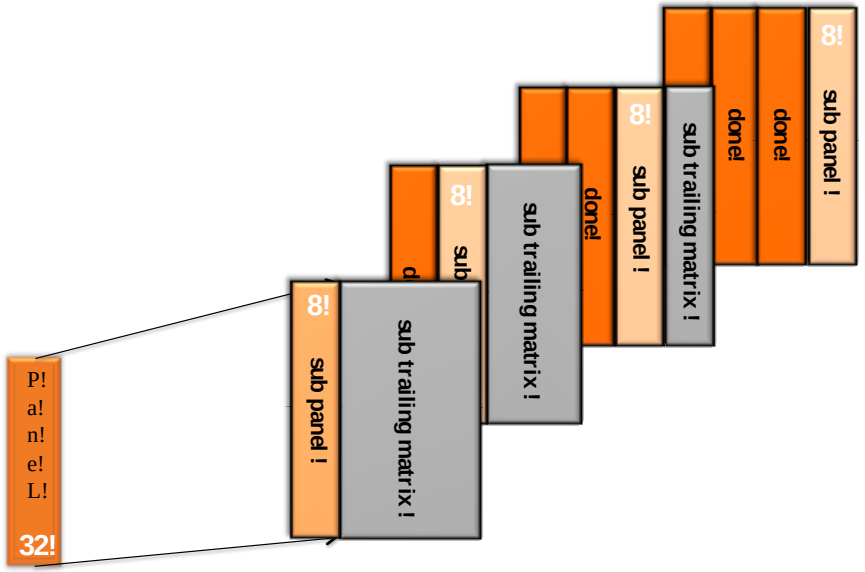
- We propose to develop two layers blocking: a recursive and nested blocking technique that block also the panel.

MAGMA Batched Computations

Two-layers blocking:



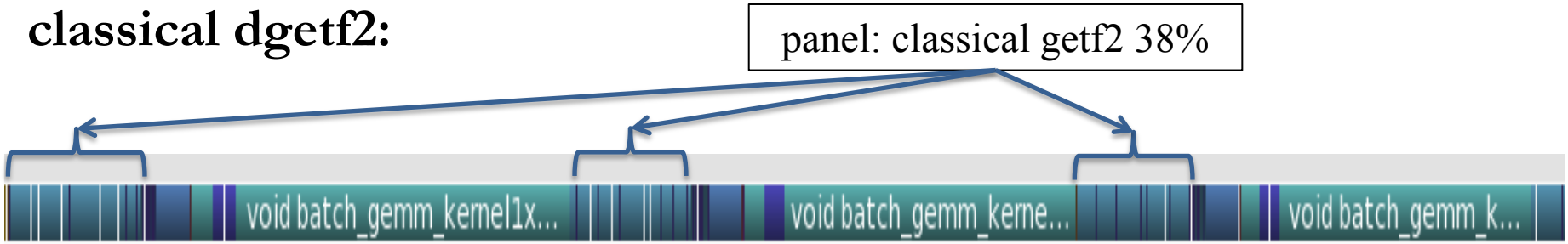
(a) Recursive nested blocking fashion.



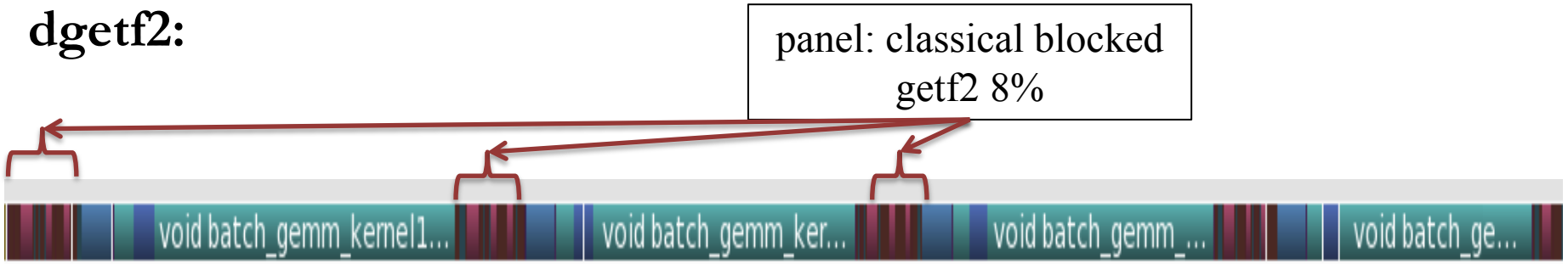
(b) Classical blocking fashion.

MAGMA Batched Computations

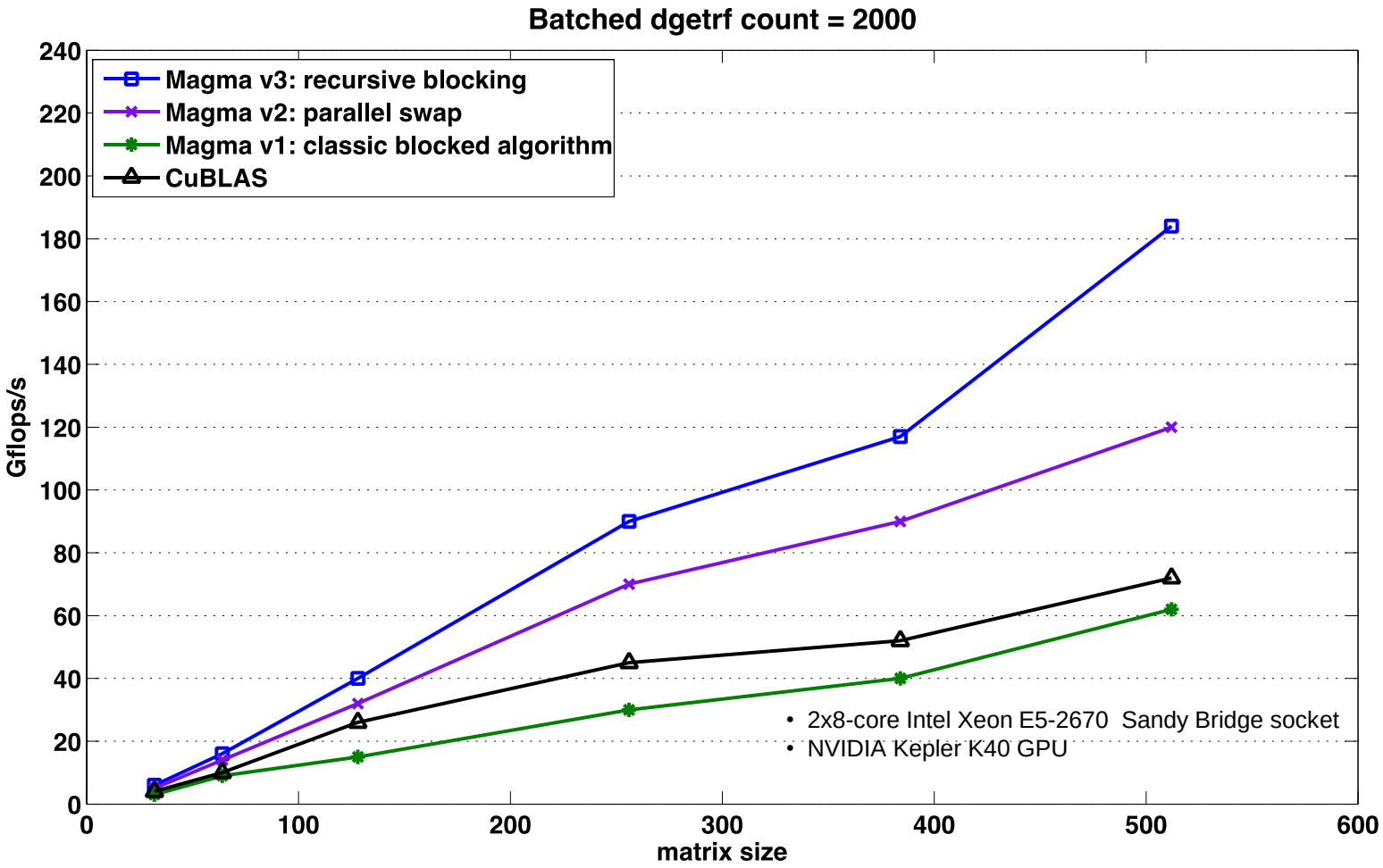
panel factorization classical dgetf2:



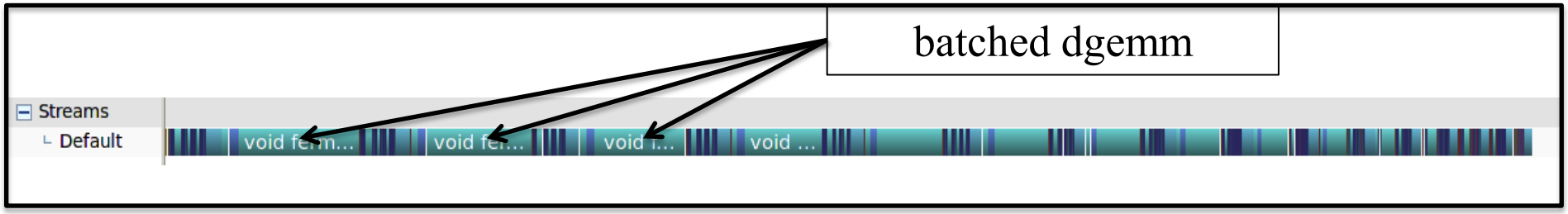
Recursive blocking of dgetf2:



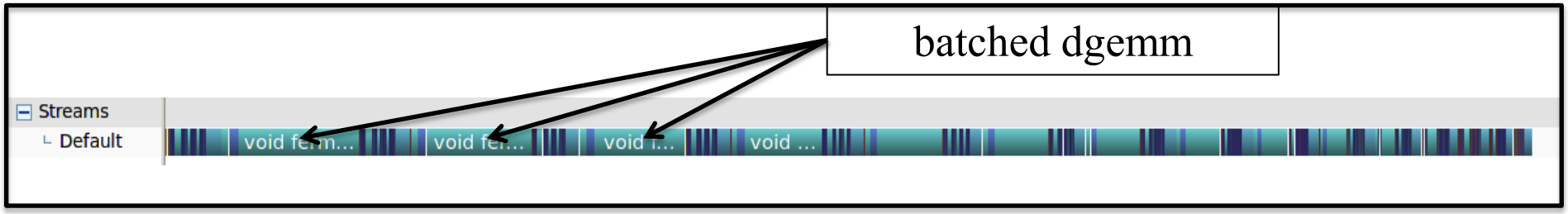
MAGMA Batched Computations

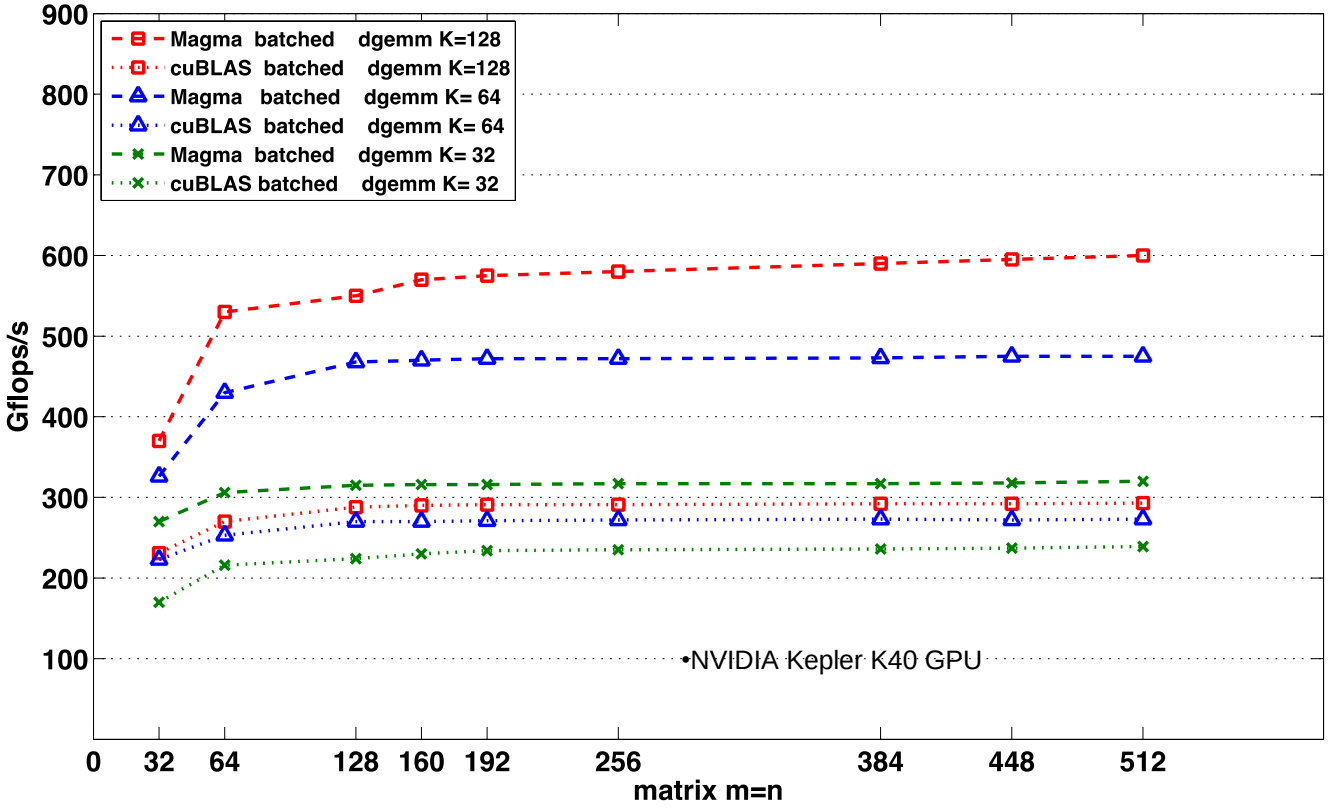


MAGMA Batched Computations

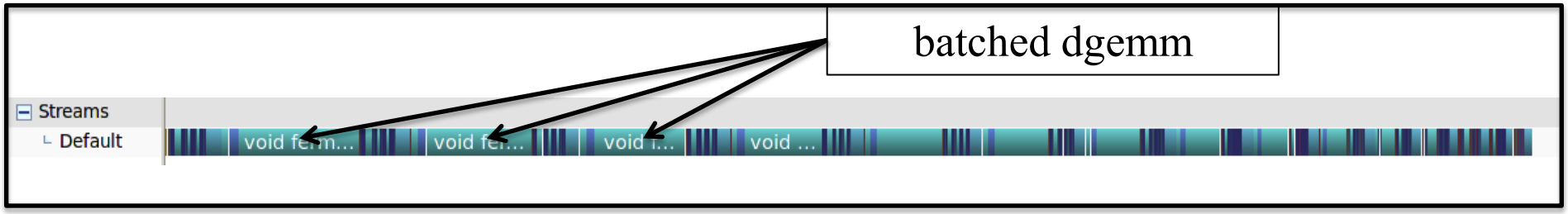


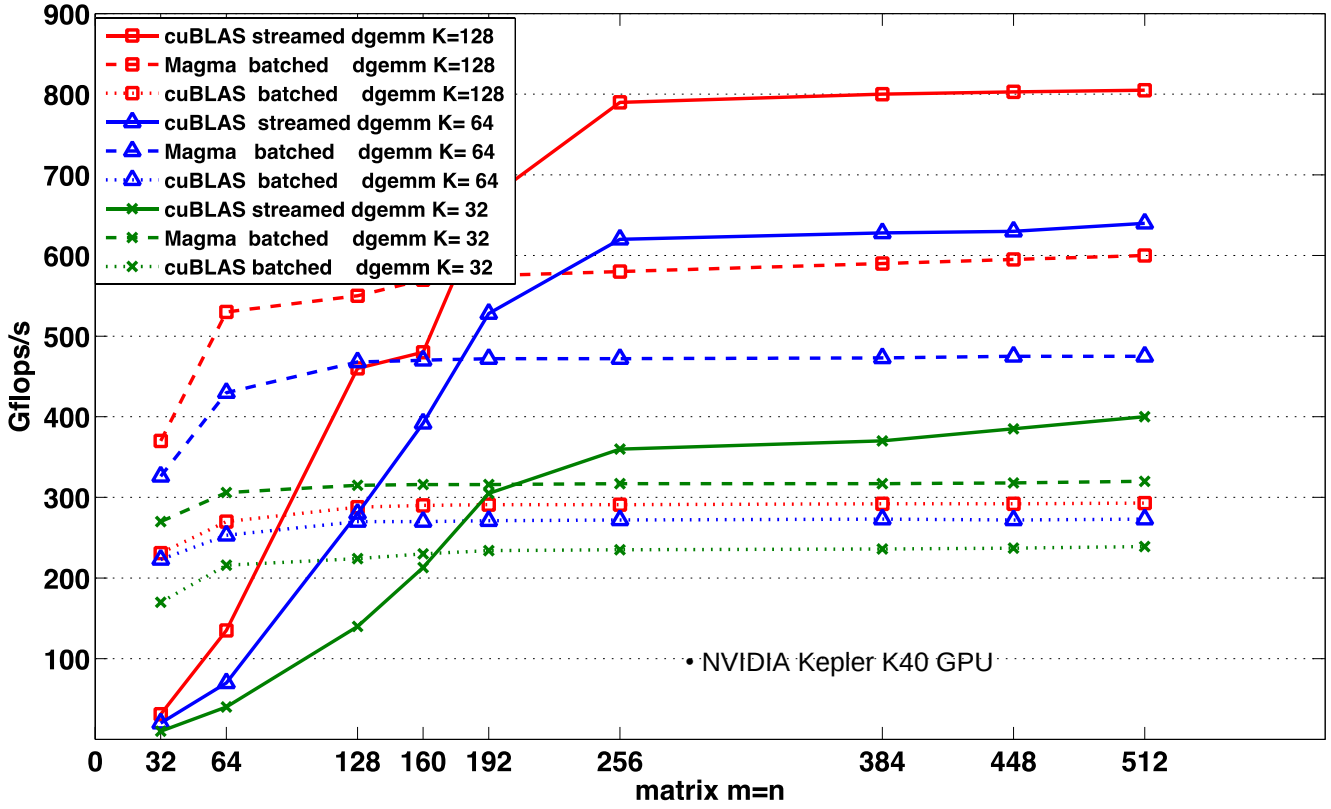
MAGMA Batched Computations



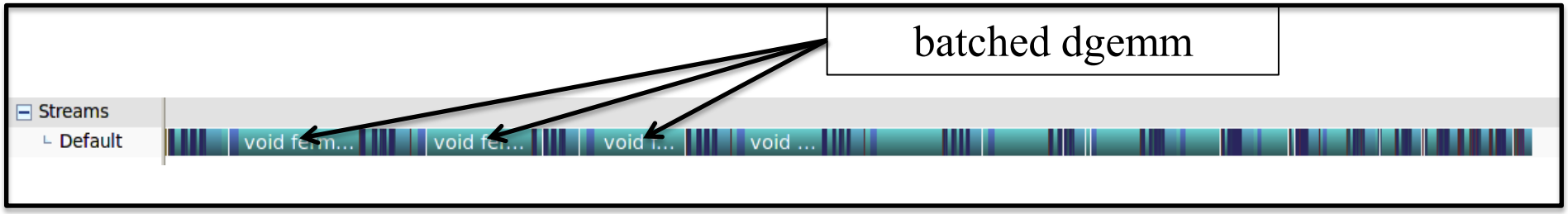


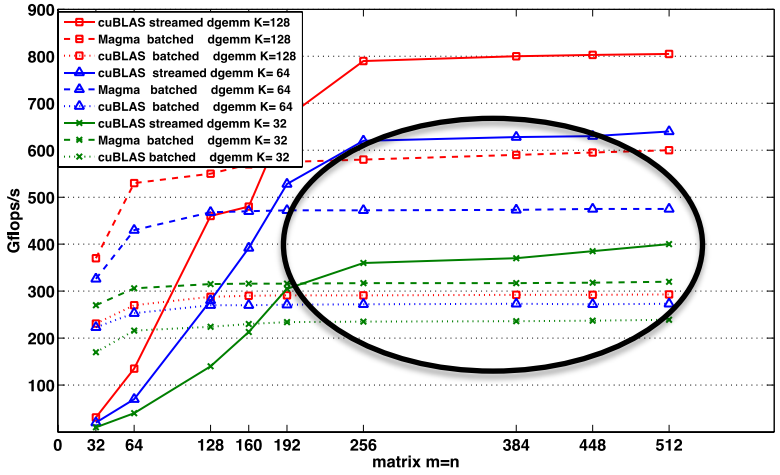
MAGMA Batched Computations





MAGMA Batched Computations





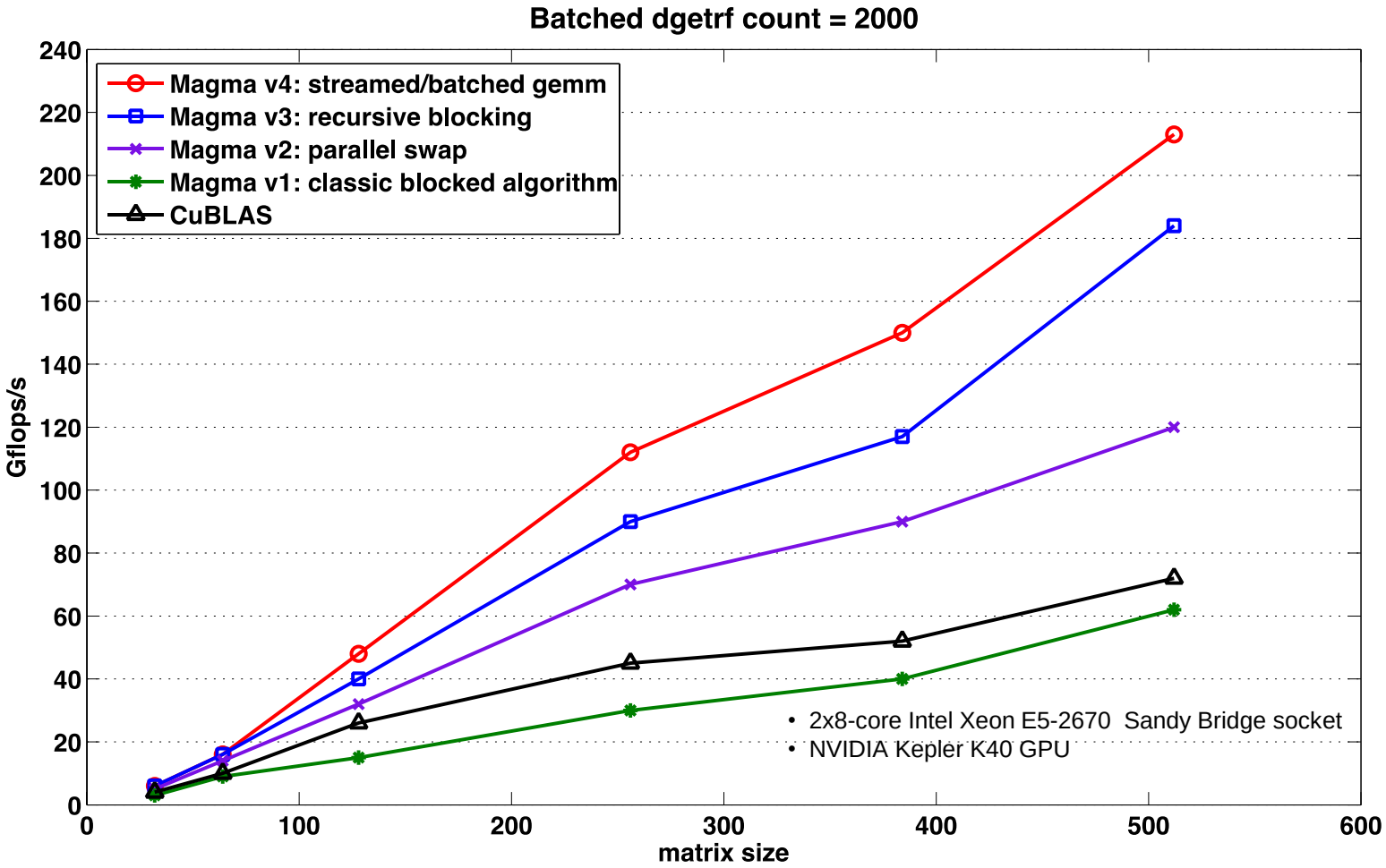
Bottlenecks:

- Batched gemm kernel from cuBLAS and Magma are well suited for small matrix sizes (128) but stagnate for larger sizes (>128)

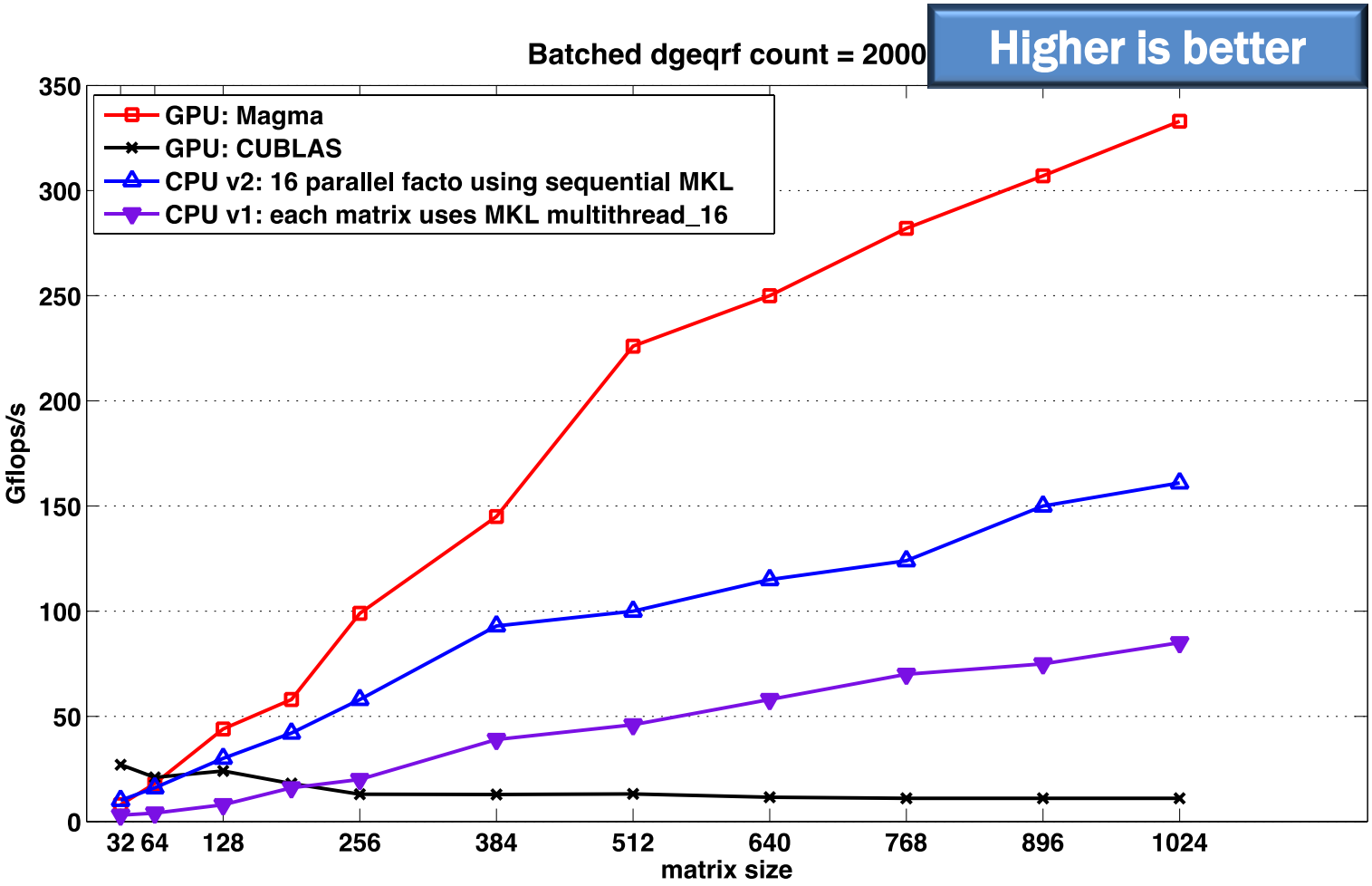
Proposition:

- Streamed gemm can provide higher performance for large matrix size (>128) and thus we propose to use both streamed and batched according to the size of the trailing matrix

MAGMA Batched Computations

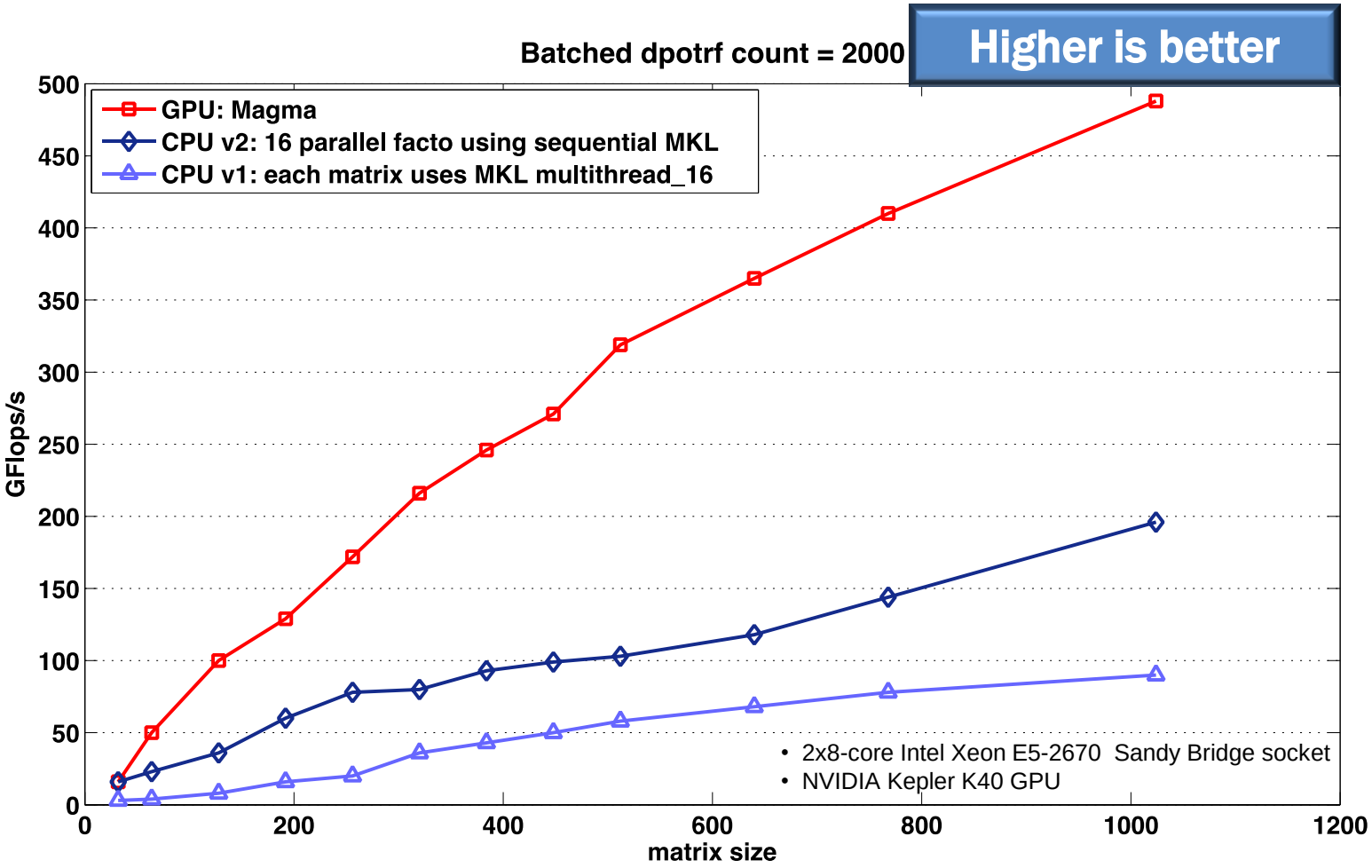


MAGMA Batched Computations

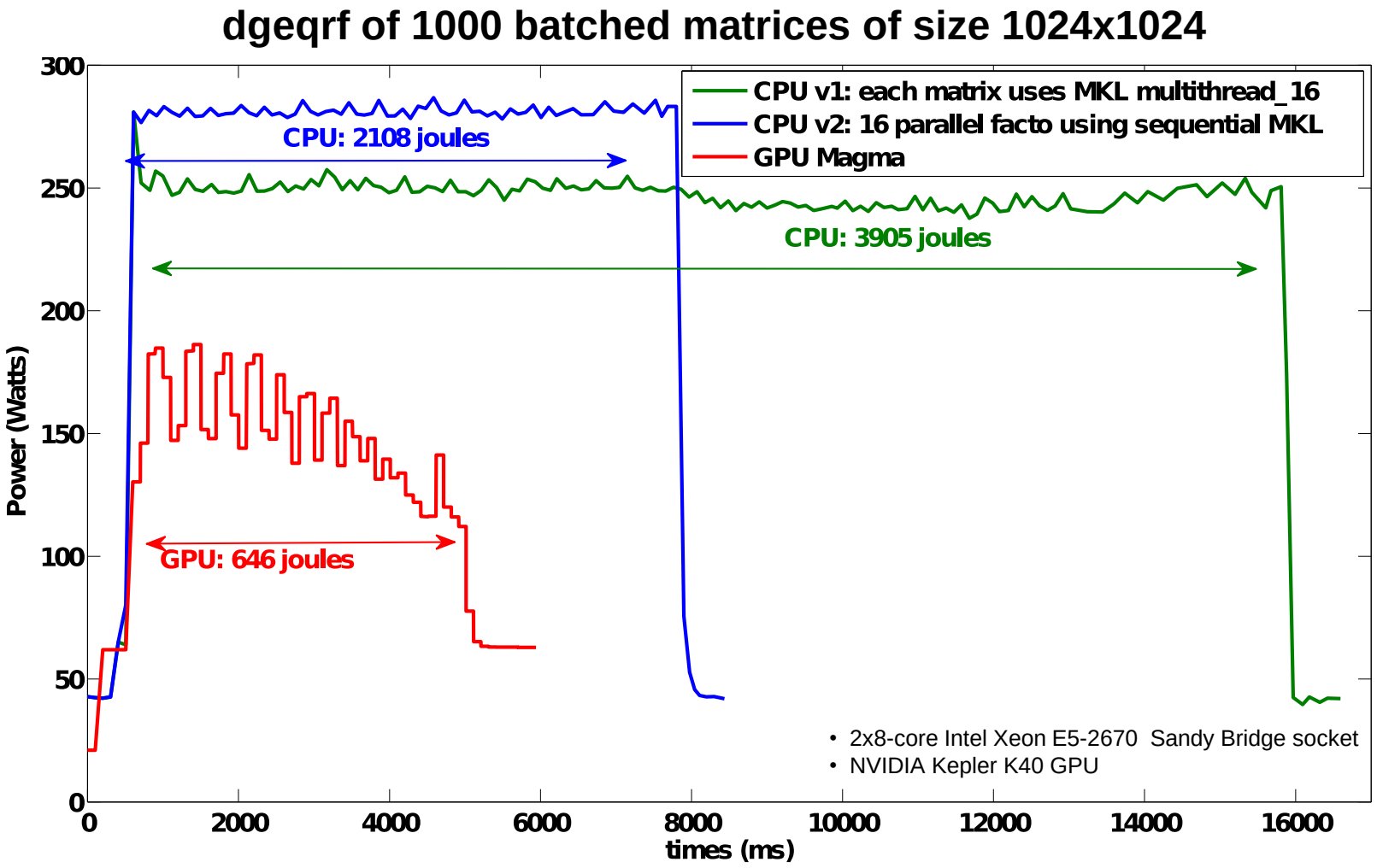


- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU

MAGMA Batched Computations



MAGMA Batched Computations

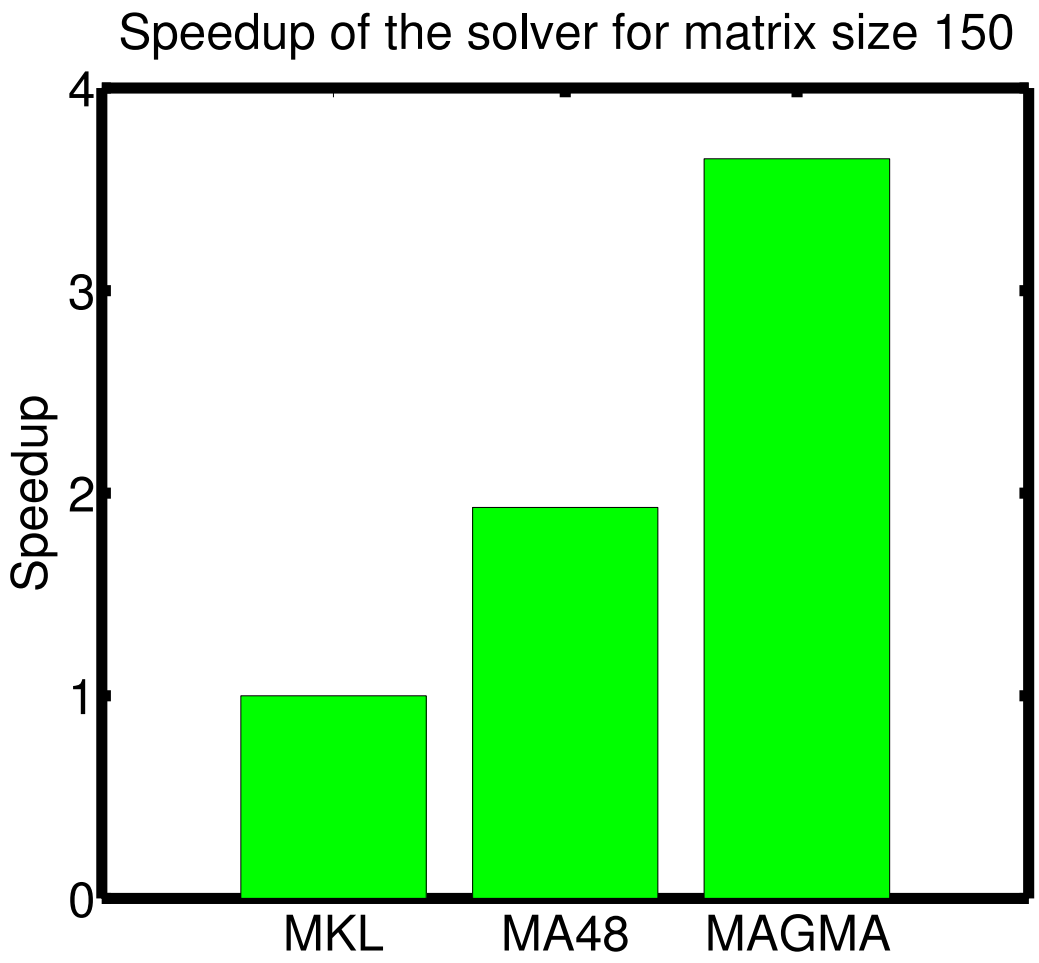


CPU does not include DRAM power

Applications

- Accelerate XNet, a fully implicit, general purpose solver for thermonuclear reaction networks in astrophysical applications
  - Simulates evolution of the nuclear kinetics where for any single time step on a single zone there is need to solve a small problem with LU
  - LU factorizations, including the backward substitution, account for 75% of the time
  - Zones can be solved independently
  - Number of zones can grow with domain size and dimension to tens of thousands
  - Can set for example to have 3900 zones per GPU
  - Problem sizes are small, in the range 14 to 150

Acceleration Results



Titan supercomputer
at ORNL – hybrid nodes with
CPU : 16 AMD Opteron Cores
GPU : Nvidia K20x

T. Dong, A. Haidar, P. Luszczek, J. Harris, S. Tomov, and J. Dongarra.
LU Factorization of Small Matrices: Accelerating Batched DGETRF on the GPU.
HPCC 2014, The 16th IEEE International Conferences on high performance computing and communications.

MAGMA Batched Computations

References

•**A. Haidar, T. Dong, P. Luszczek, S. Tomov and J. Dongarra.**
Batched Matrix Computations on Hardware Accelerators Based on GPUs.
International Journal of High Performance Computing Applications 2014.

• **A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra.**
Optimization for Performance and Energy for Batched Matrix Computations on GPUs
20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Workshop on General Purpose Processing Using GPUs. GPGPU/PPoPP 2015.

• **T. Dong, A. Haidar, P. Luszczek, J. Harris, S. Tomov, and J. Dongarra.**
LU Factorization of Small Matrices: Accelerating Batched DGETRF on the GPU.
HPCC 2014, The 16th IEEE International Conferences on high performance computing and communications.

• **T. Dong, A. Haidar, S. Tomov, and J. Dongarra.**
A Fast Batched Cholesky Factorization on a GPU
ICPP 2014, The 43rd International Conference on Parallel Processing 2014.

Future Directions

- Extended functionality
  - Variable sizes
  - Mixed-precision techniques
  - Sparse direct multifrontal solvers & preconditioners
  - Applications
- Further tuning
  - autotuning
- GPU-only algorithms and implementations
- MAGMA Embedded

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