Matrix Algebra on GPU and Multicore Architectures (MAGMA)

PART I : Introduction

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

CSCS-USI Tutorial on GPU-enabled numerical libraries
Lugano, Switzerland
September 14-15, 2013

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

Outline

- Part I
  - Introduction to MAGMA
  - Methodology
  - Performance
- Part II
  - Hands-on training
  - Using and contributing to MAGMA
  - Examples and exercises

MAGMA: LAPACK for GPUs

- **MAGMA**
  - Matrix algebra for GPU and multicore architecture
  - To provide LAPACK/ScaLAPACK on hybrid architectures
  - http://icl.cs.utk.edu/magma/

- **MAGMA for CUDA, Intel Xeon Phi, and OpenCL**
  - Hybrid dense linear algebra:
    - One-sided factorizations and linear system solvers
    - Two-sided factorizations and eigenproblem solvers
    - A subset of BLAS and auxiliary routines in CUDA

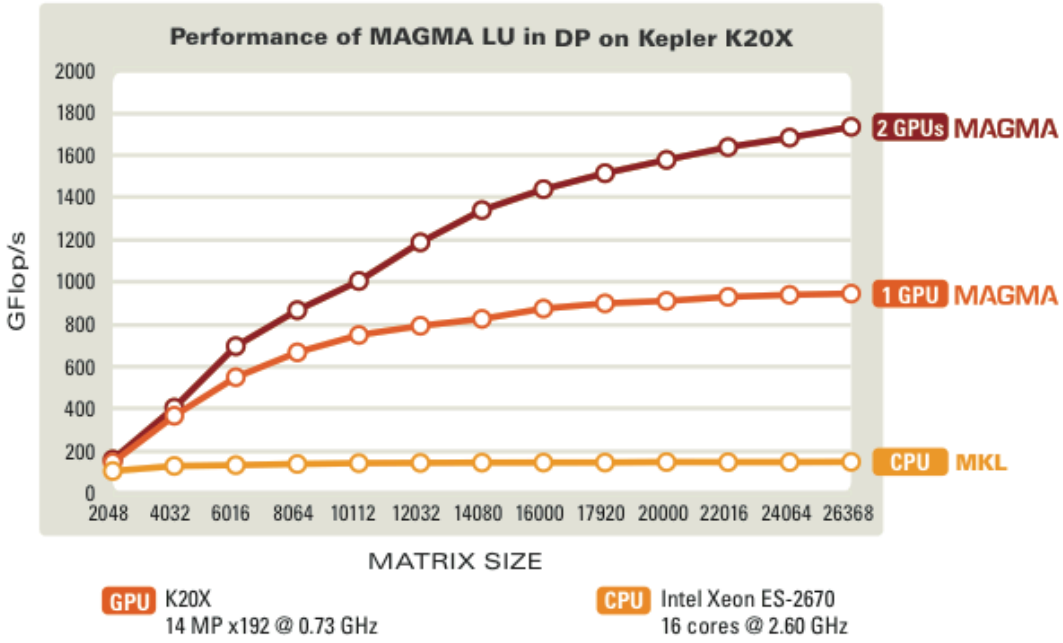
- **MAGMA developers & collaborators**
  - UTK, UC Berkeley, UC Denver, INRIA (France), KAUST (Saudi Arabia)
  - Community effort, similarly to LAPACK/ScaLAPACK

Key Aspects of MAGMA

HYBRID ALGORITHMS

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

PERFORMANCE



FEATURES AND SUPPORT

- MAGMA 1.4 FOR CUDA
- clMAGMA 1.0 FOR OpenCL
- MAGMA MIC 1.0 FOR Intel Xeon Phi

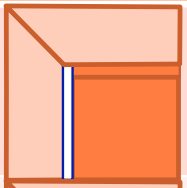
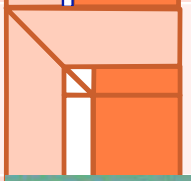
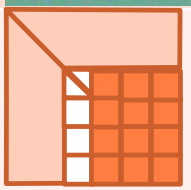
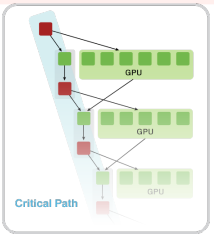
CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●	●	Eigenvalue problem solvers
●			MAGMA BLAS
●			CPU Interface
●	●	●	GPU Interface
●	●	●	Multiple precision support
●			Non-GPU-resident factorizations
●		●	Multicore and multi-GPU support
●			Tile factorizations with StarPU dynamic scheduling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

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

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

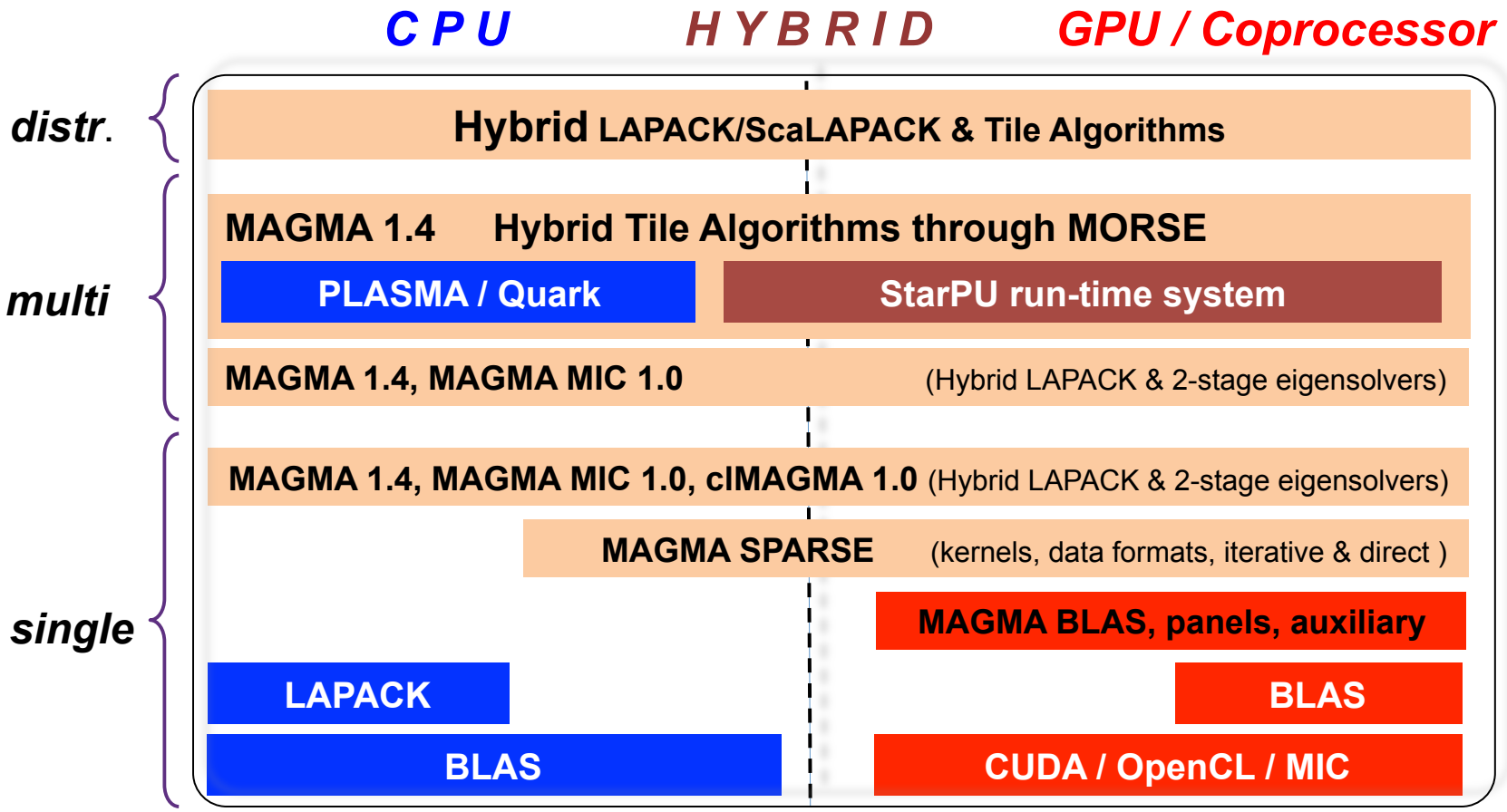
A New Generation of DLA Software

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 - hybrid kernels

Key Features of MAGMA 1.4

- High performance
- Multiple precision support (Z, C, D, S, and MP)
- Hybrid algorithms
- Out-of-GPU memory algorithms
- MultiGPU support

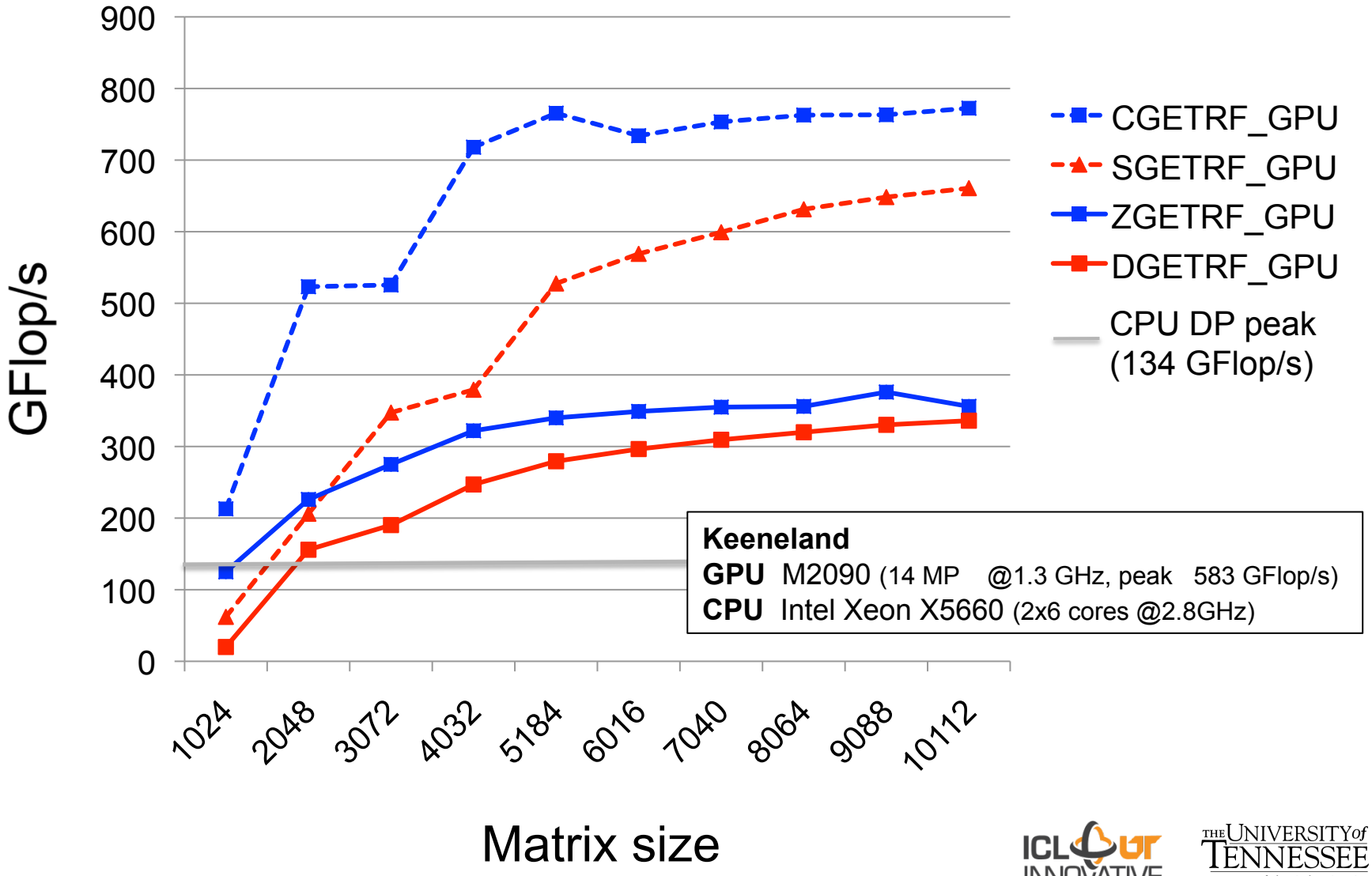
MAGMA Libraries & Software Stack



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

Multiple precision support

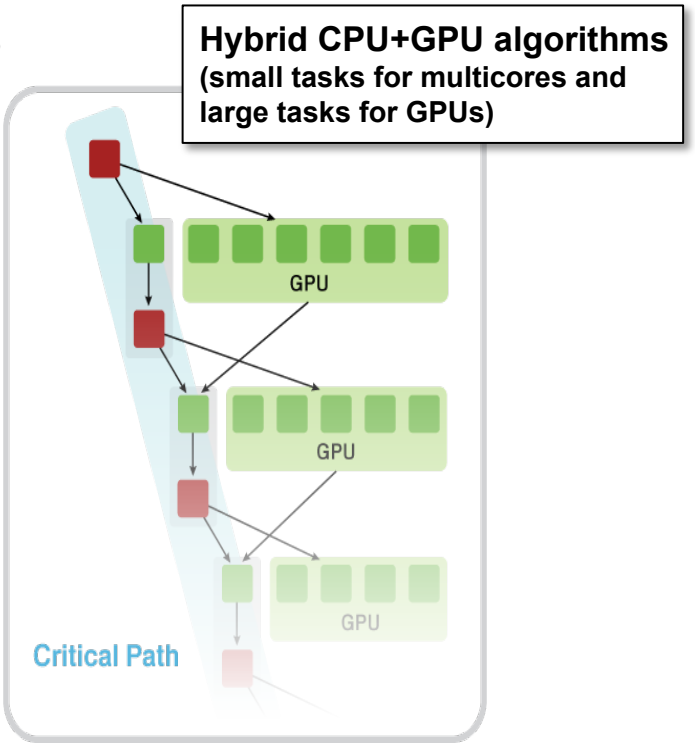
Performance of the LU factorization in various precisions



Methodology overview

A methodology to use all available resources:

- MAGMA uses hybridization methodology based on
  - Representing linear algebra algorithms as collections of tasks and data dependencies among them
  - Properly scheduling tasks' execution over multicore and GPU hardware components



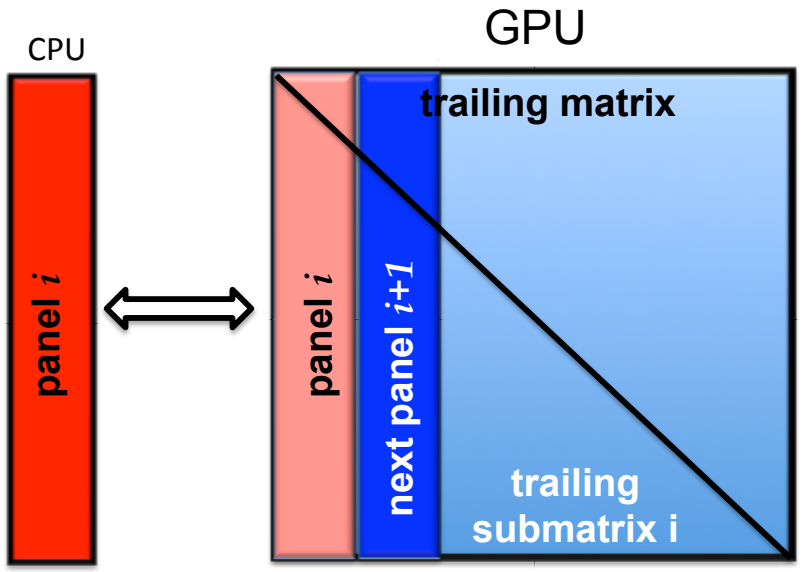
- Successfully applied to fundamental linear algebra algorithms
  - One- and two-sided factorizations and solvers
  - Iterative linear and eigensolvers

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

Hybrid Algorithms

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

- **Hybridization**
  - **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 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)

MAGMA programming model

Host program: the same sources to provide support to GPU, MIC, and OpenCL through wrappers

```
for ( j=0; j<n; j += nb)  {
      jb = min(nb, n – j);
      magma_zherk( MagmaUpper, MagmaConjTrans,
                   jb, j, m_one, dA(0, j), ldda, one, dA(j, j), ldda, qu
      magma_zgetmatrix_async( jb, jb, dA(j,j), ldda, work, 0, jb, queue
      if ( j+jb < n )
          magma_zgemm( MagmaConjTrans, MagmaNoTrans, jb, n-j-jb
                       dA(0, j ), ldda, dA(0, j+jb), ldda, z_one,  dA(j,
      magma_event_sync( event );
      lapackf77_zpotrf( MagmaUpperStr, &jb, work, &jb, info );
      if ( *info != 0 )
           *info += j;
      magma_zsetmatrix_async( jb, jb, work, 0, jb, dA(j,j), ldda, queue, &e
      if ( j+jb < n ) {
          magma_event_sync( event );
          magma_ztrsm( MagmaLeft, MagmaUpper, MagmaConjTrans, Mag
                       jb, n-j-jb, z_one, dA(j, j), ldda, dA(j, j+jb), ldda, queu
      }
}
```

Intel Xeon Phi interface

```
// ====================================
// 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, m
                              magmaDoubleComplex_const_ptr dB, size_t dB_offset, m
    magmaDoubleComplex beta,  magmaDoubleComplex_ptr       dC, size_t dC_offset, m
    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("s    send failed with err %d\n", errno);
        fflush(st
    }

    if ((err = scif                    emm_param, sizeof(gemm_param), 1)) <= 0) {
        err = err
```

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

Mixed-Precision Iterative Refinement

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

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

i.e., $x_{i+1} = x_i + P (b - A x_i)$, where $P \approx (LU)^{-1}$

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.

Mixed-Precision Iterative Refinement

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

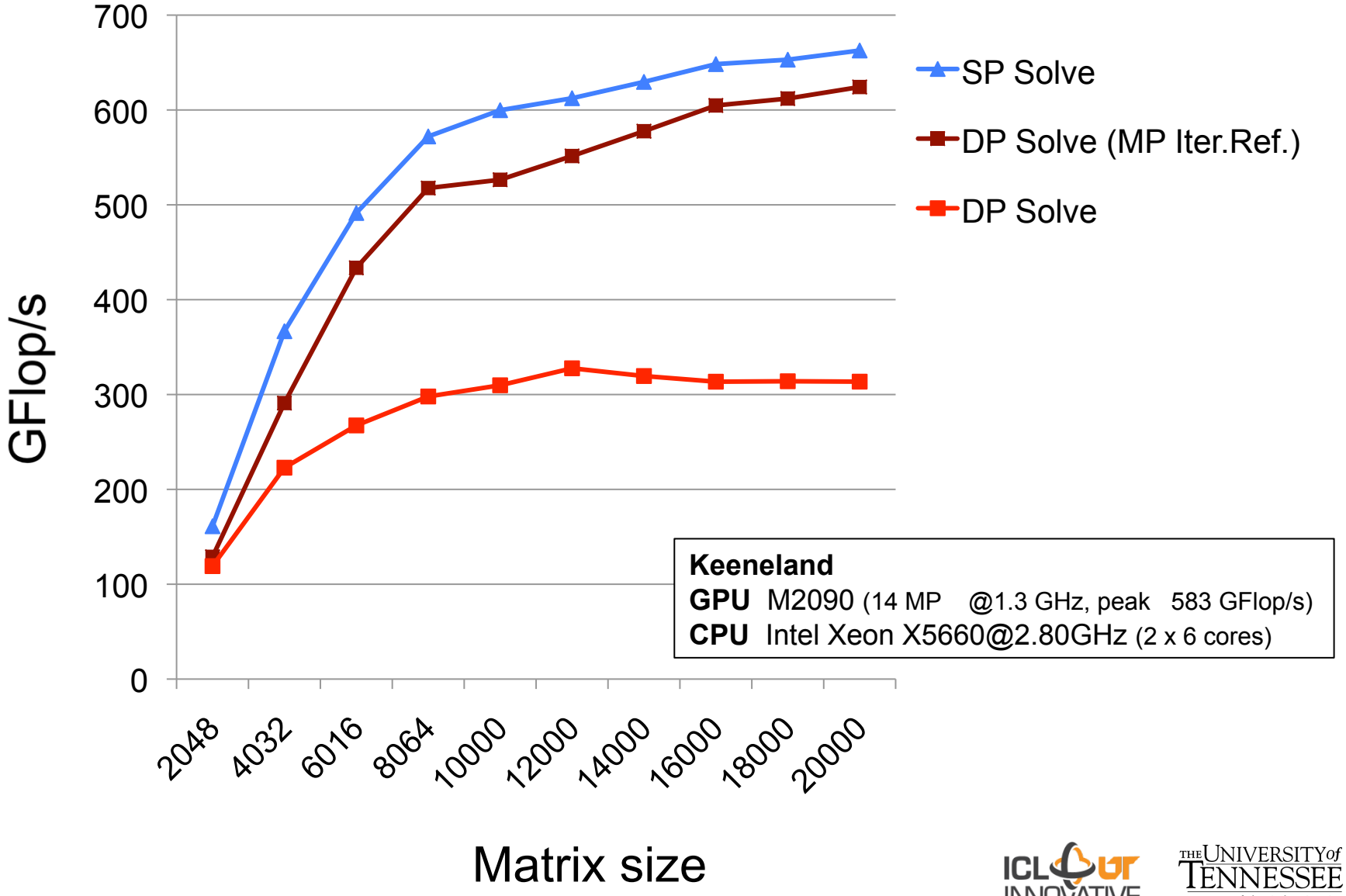
L U = lu(A)	SINGLE	$O(n^3)$
x = L\(U\b)	SINGLE	$O(n^2)$
r = b – Ax	DOUBLE	$O(n^2)$
WHILE || r || not small enough		
z = L\(U\r)	SINGLE	$O(n^2)$
x = x + z	DOUBLE	$O(n^1)$
r = b – Ax	DOUBLE	$O(n^2)$
END		

- 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; $O(10^8)$

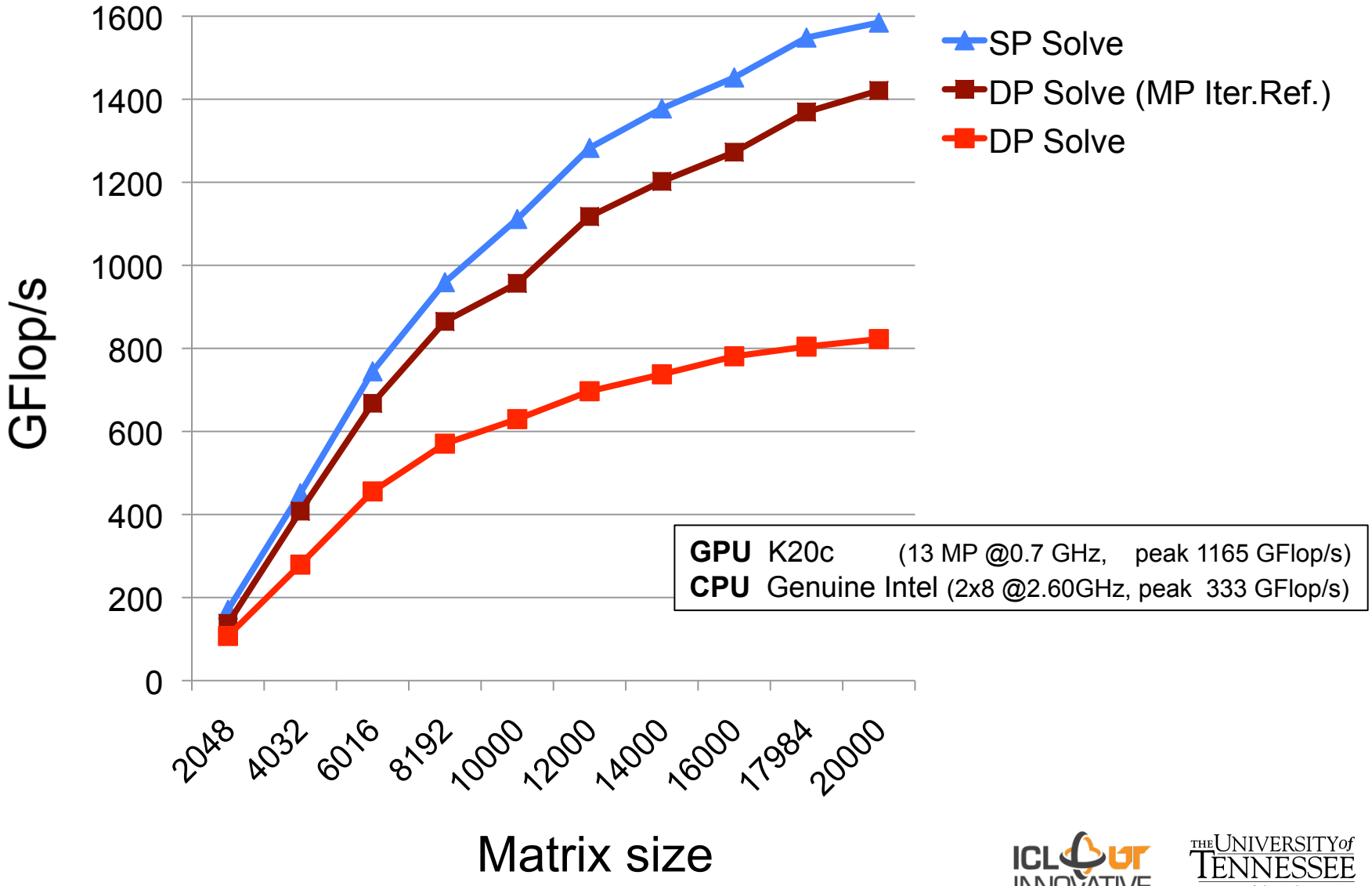
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



Mixed precision iterative refinement

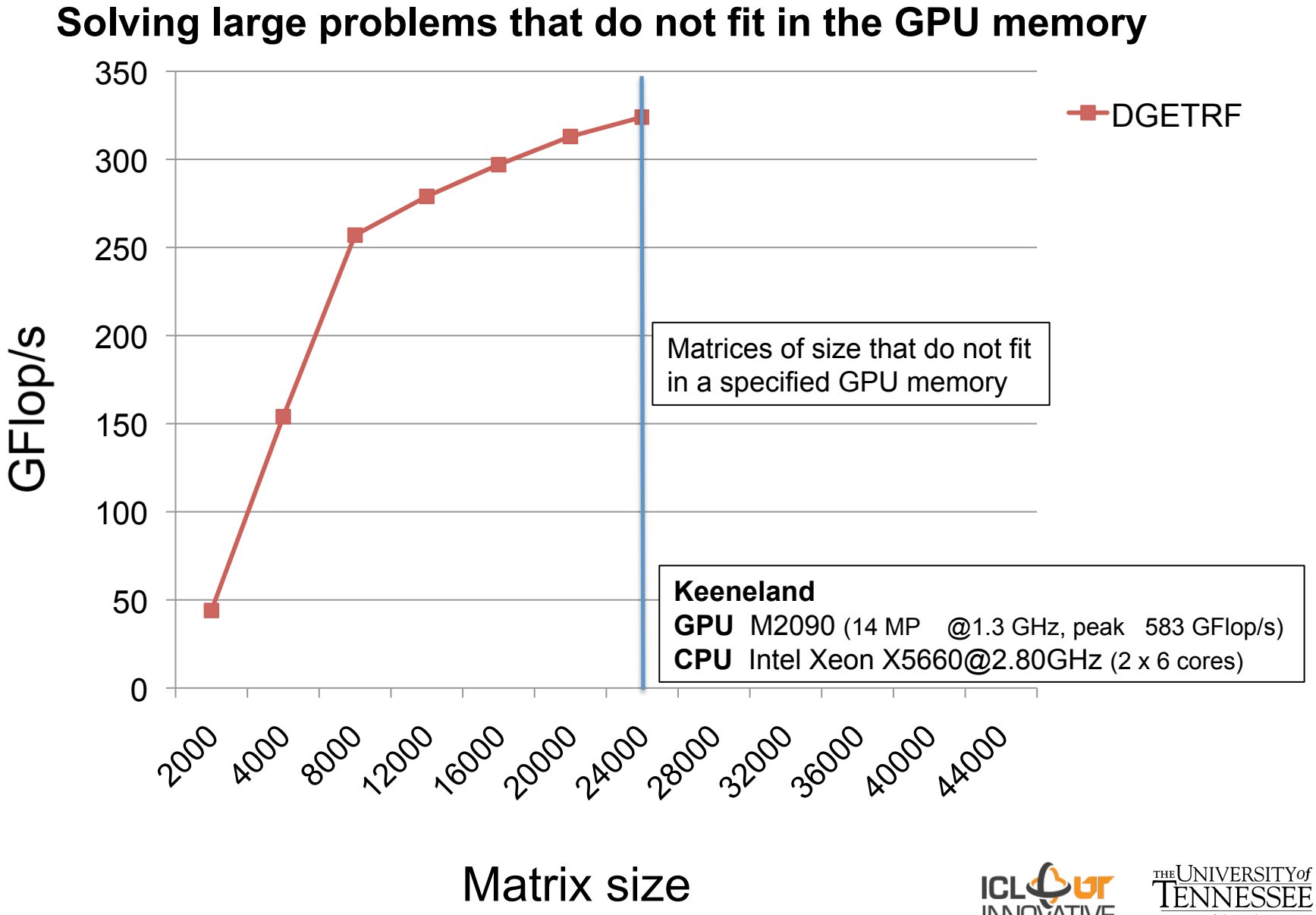
Solving general dense linear systems using mixed precision iterative refinement



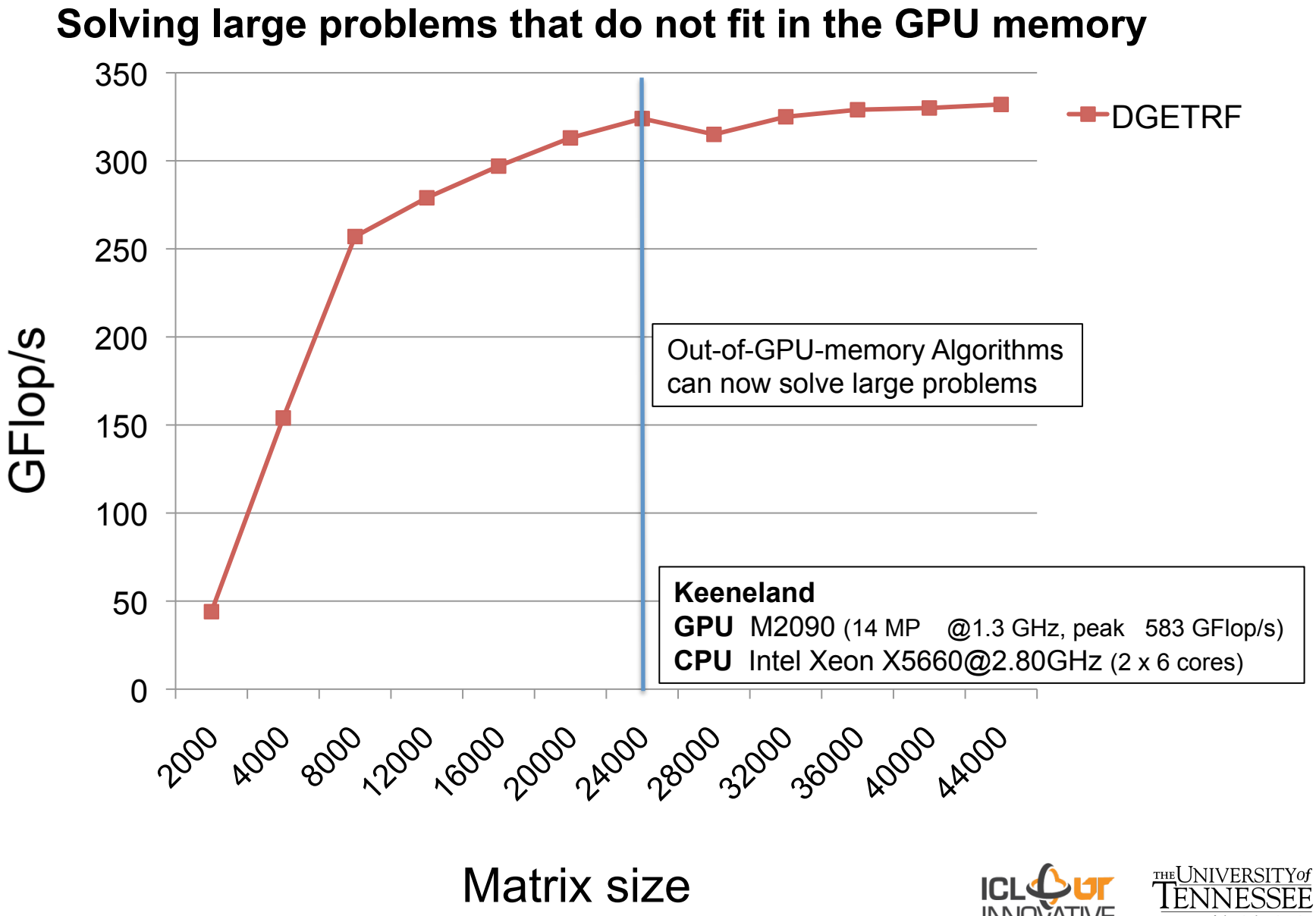
ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Out of GPU Memory Algorithms

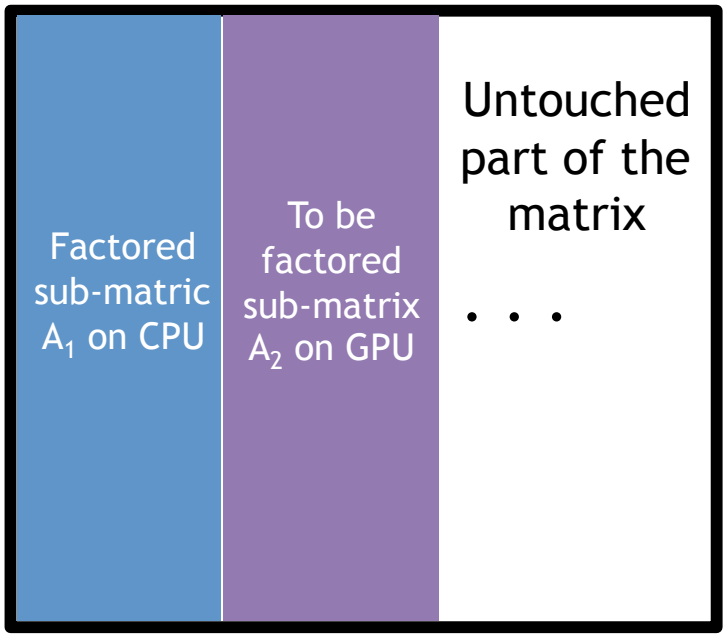


Out of GPU Memory Algorithms



Out of GPU Memory Algorithms

- 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



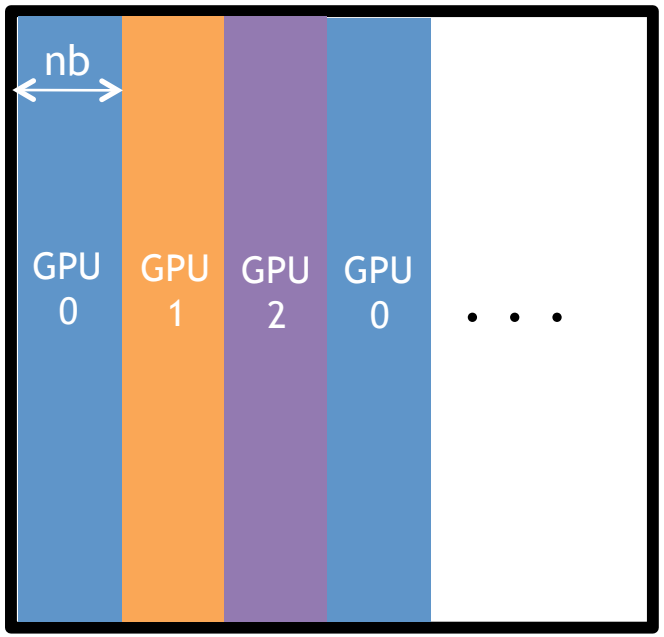
1) Copy A_2 to the GPU
2) Update A_2 using A_1 (a panel of A_1 at a time)
3) Factor the updated A_2 using existing hybrid code
4) Copy factored A_2 to the CPU

Trivially extended to multiGPUs:

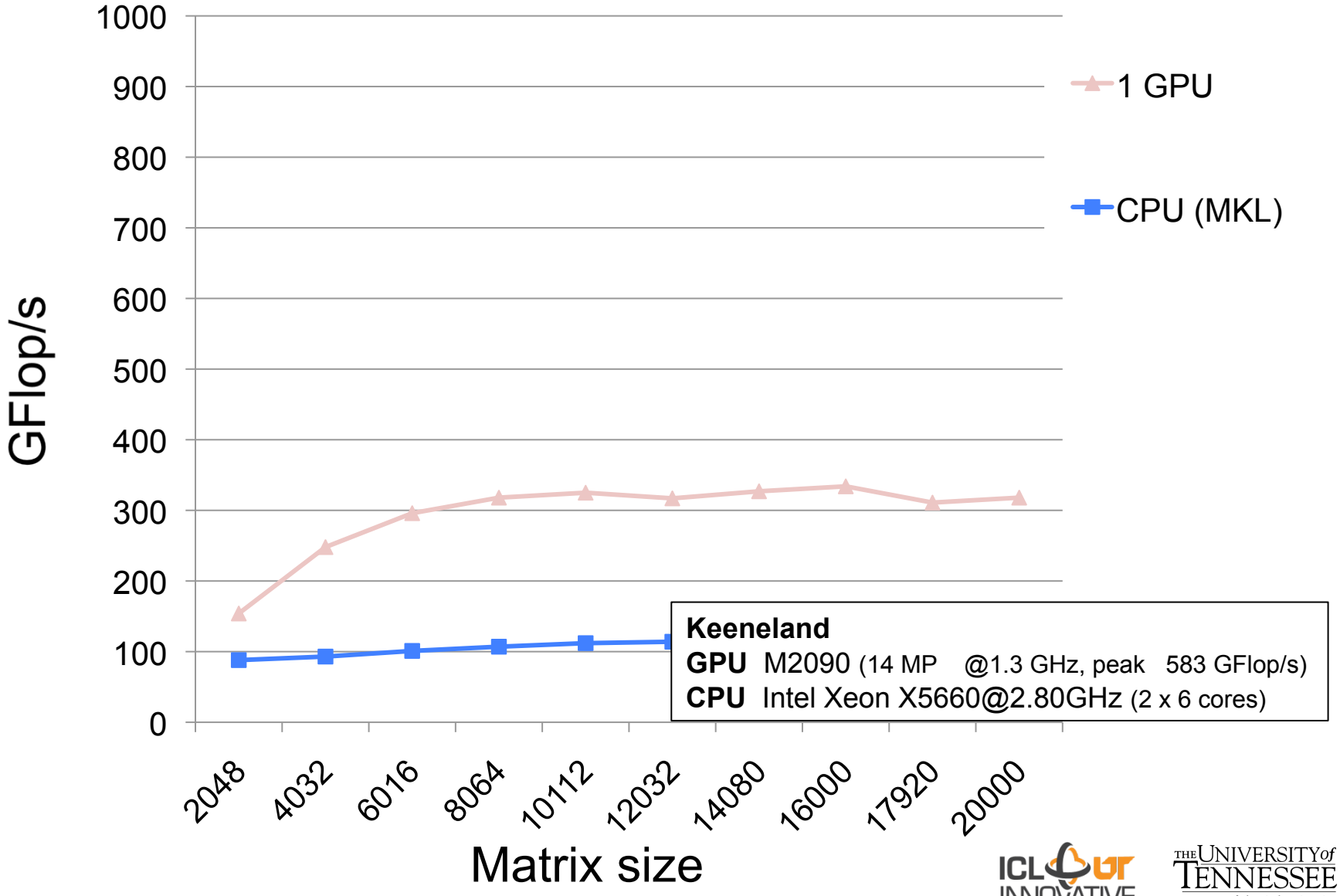
A_2 is "larger" with 1-D block cyclic distribution, again reusing existing algorithms

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



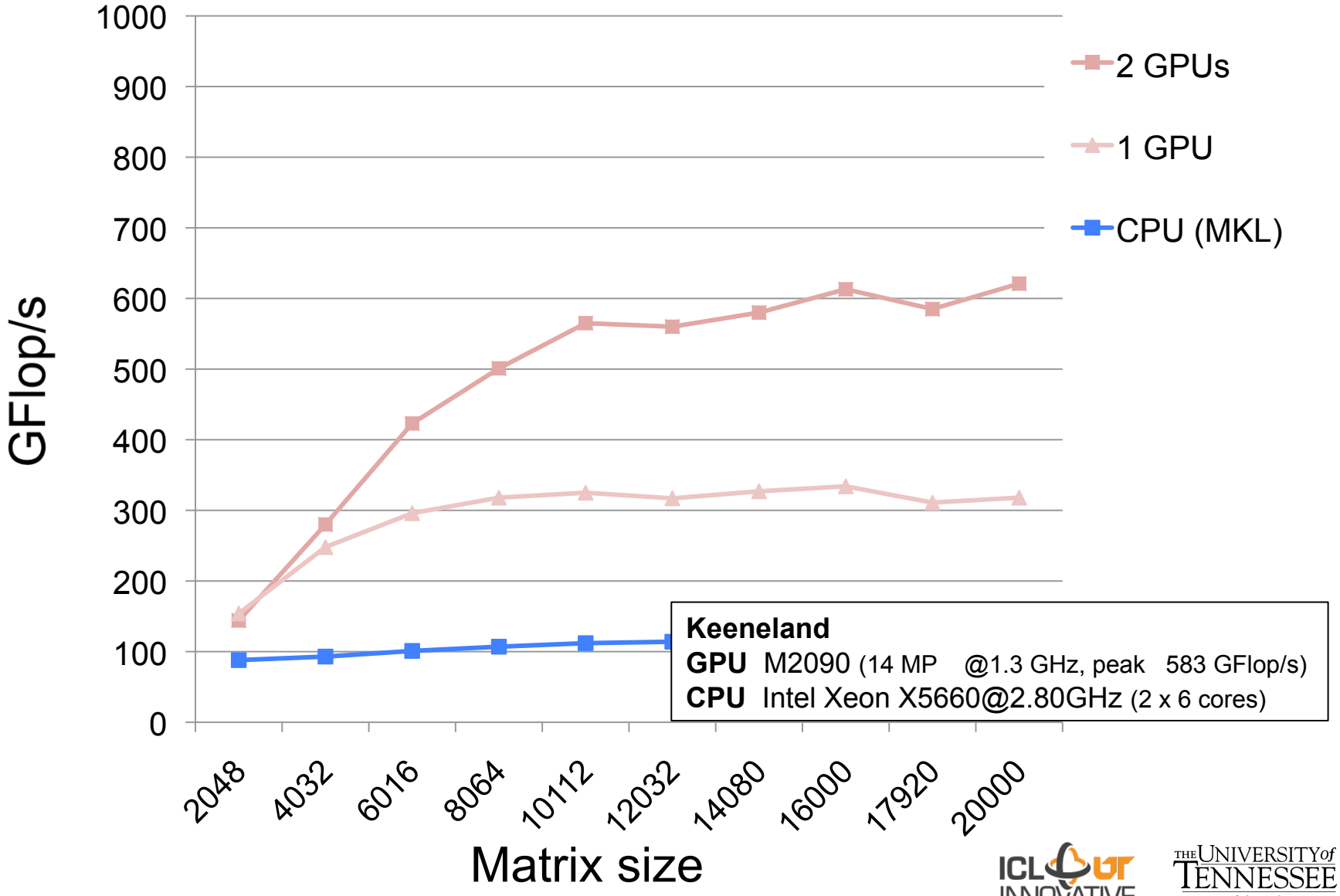
LU Factorization on multiGPUs in DP



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

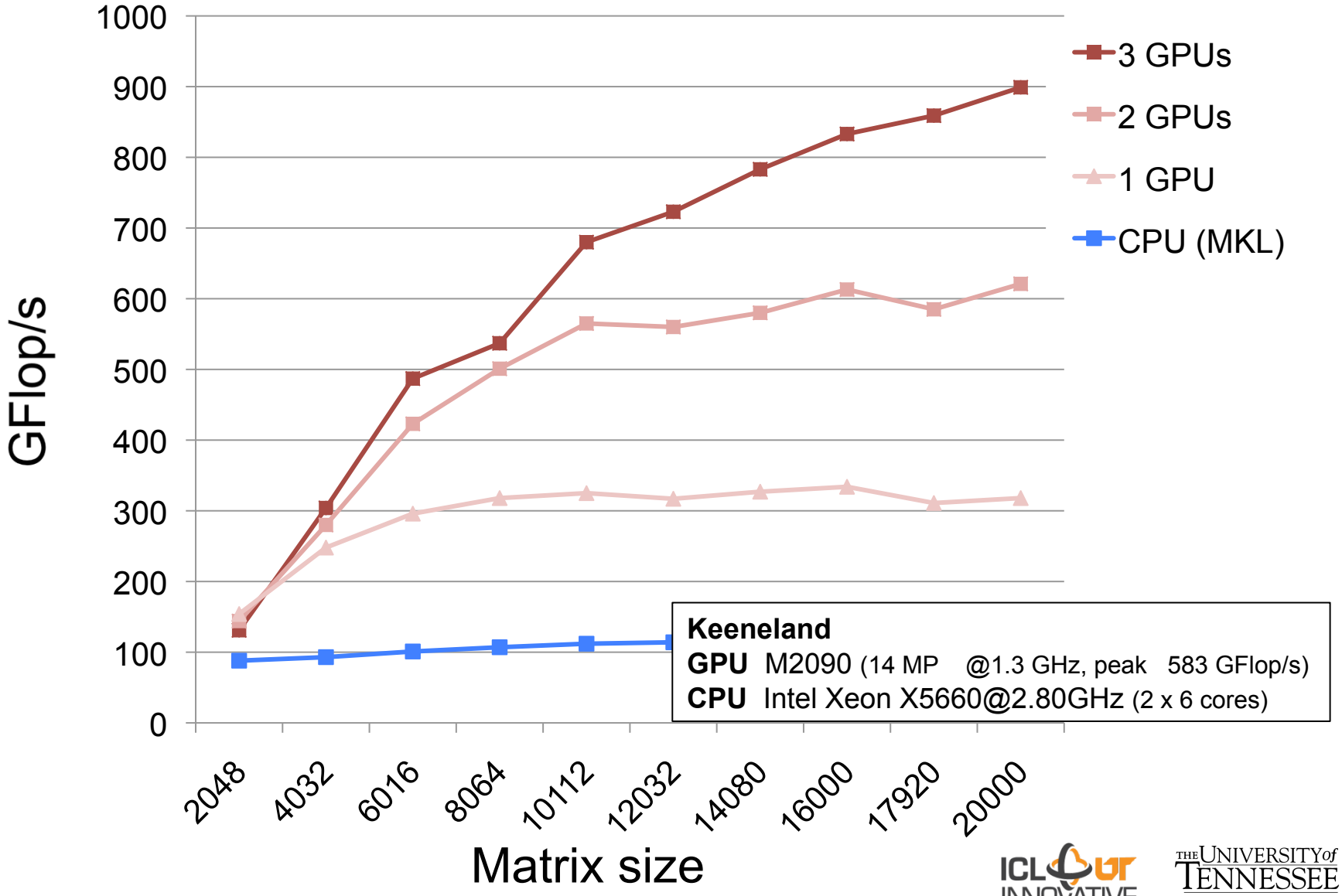
LU Factorization on multiGPUs in DP



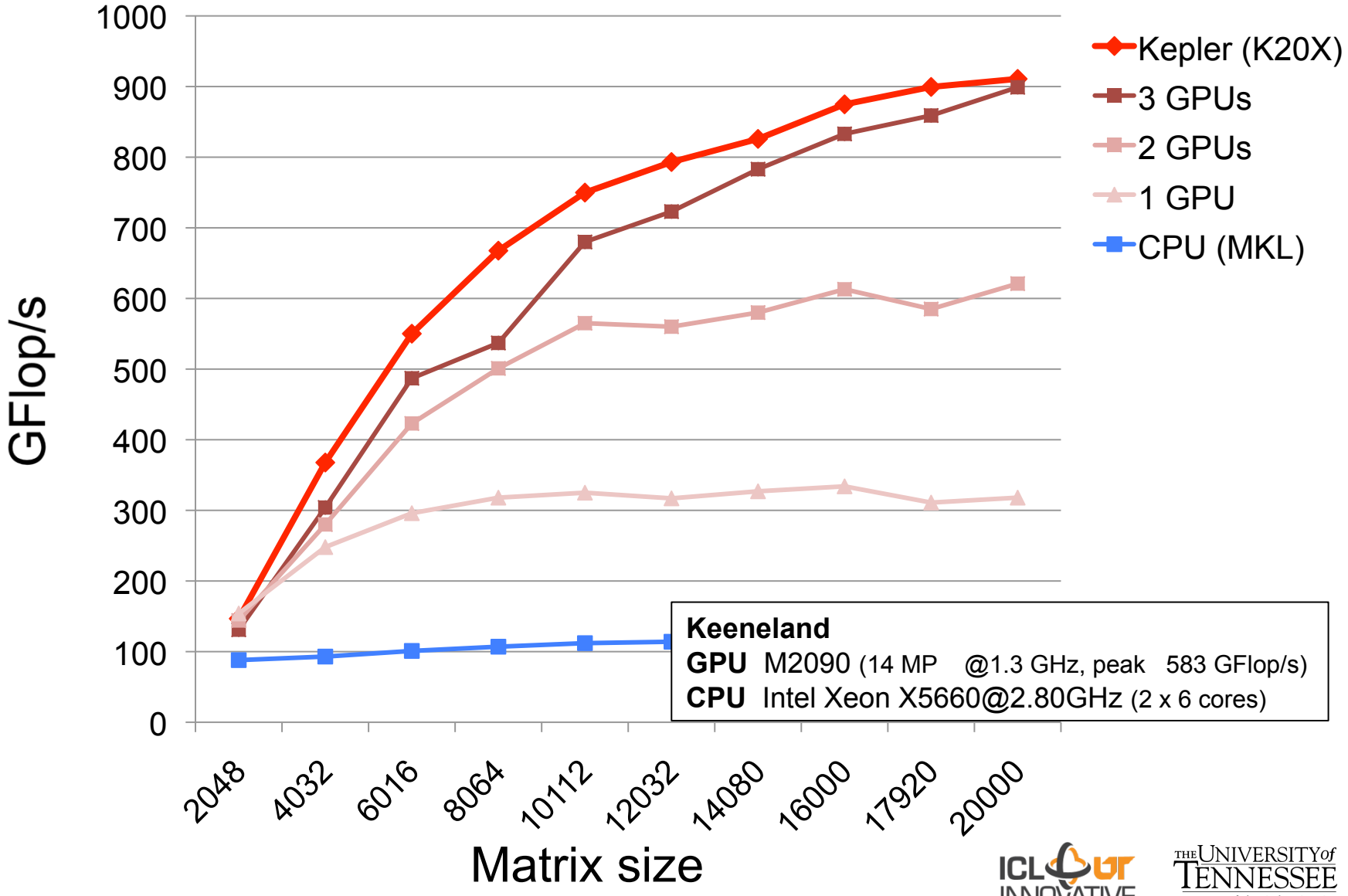
ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

LU Factorization on multiGPUs in DP



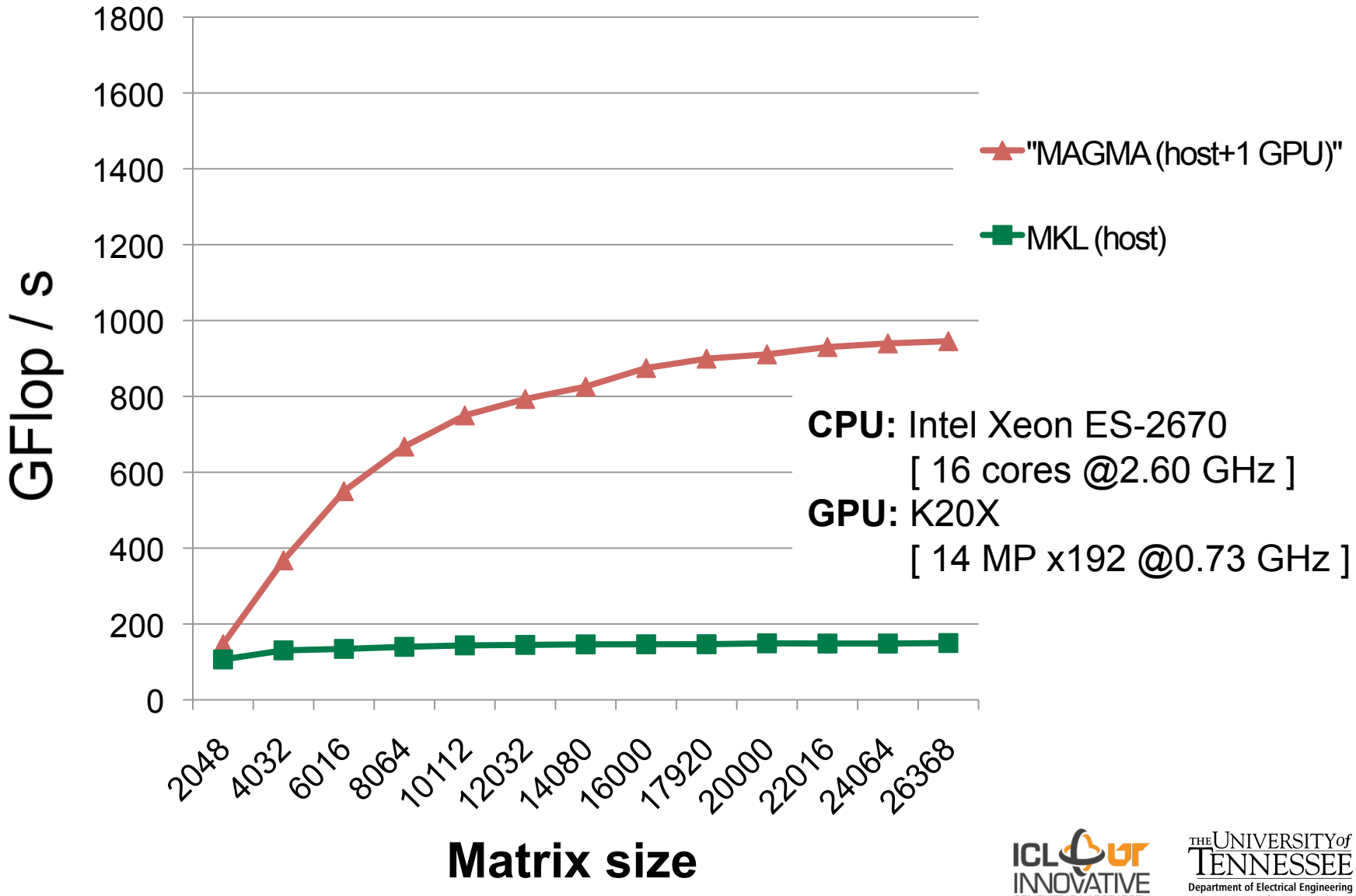
LU Factorization on Kepler in DP



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

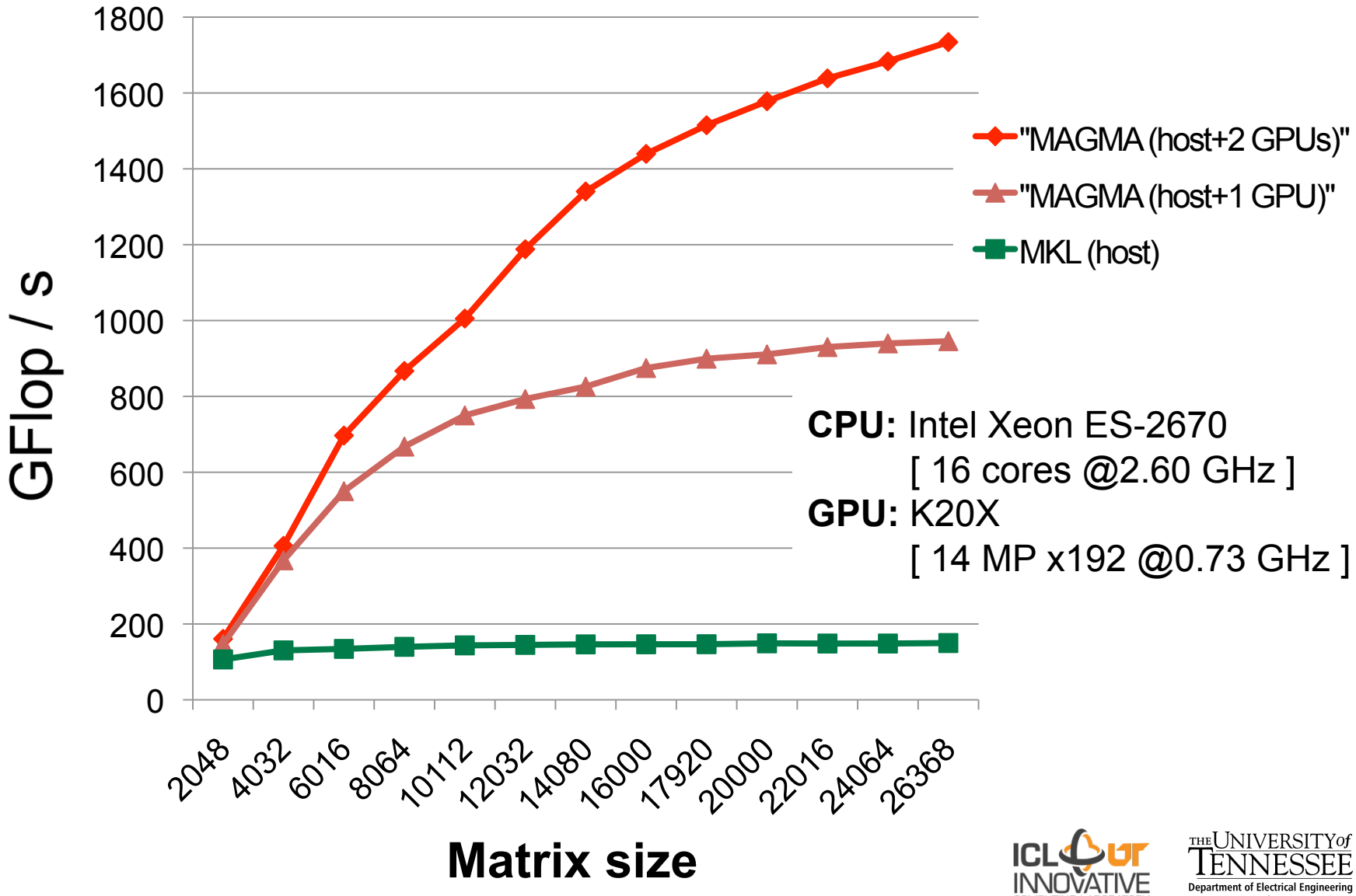
LU Scalability on Kepler in DP



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

LU Scalability on Kepler in DP



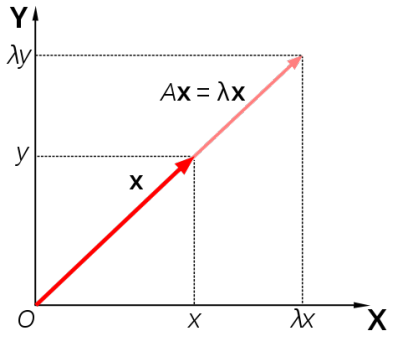
ICL UT INNOVATIVE COMPUTING LABORATORY

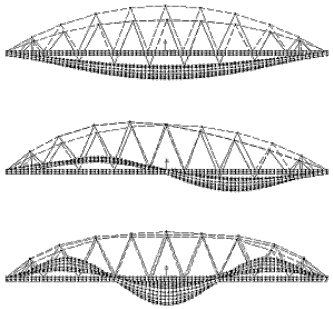
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

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.

Hermitian Generalized Eigenproblem

$$A x = \lambda B x$$

1) Compute the Cholesky factorization of $B = LL^H$
2) Transform the problem to a standard eigenvalue problem $\tilde{A} = L^{-1}AL^{-H}$
3) Solve Hermitian standard Eigenvalue problem $\tilde{A} y = \lambda y$
   - Tridiagonalize $\tilde{A}$ SYMV
   - Solve the tridiagonal eigenproblem
   - Transform the eigenvectors of the tridiagonal to eigenvectors of $\tilde{A}$
4) Transform back the eigenvectors $x = L^{-H} y$

MAGMA Two-sided Factorizations

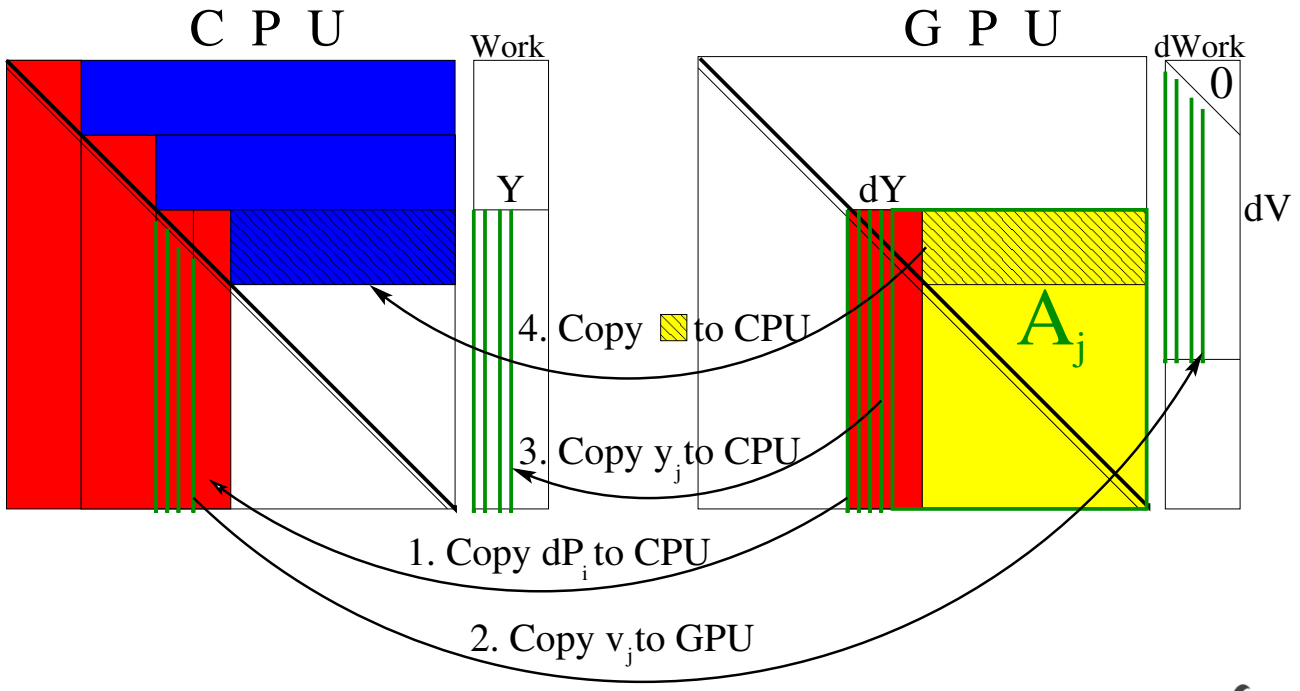
- Two-sided factorizations
  $Q' A Q = H$, H – upper Hessenberg / tridiagonal,
  $Q' A P = B$, B – bidiagonal
  Q and P – orthogonal similarity transformations
- **Importance**

  One-sided factorizations
  - bases for linear solvers

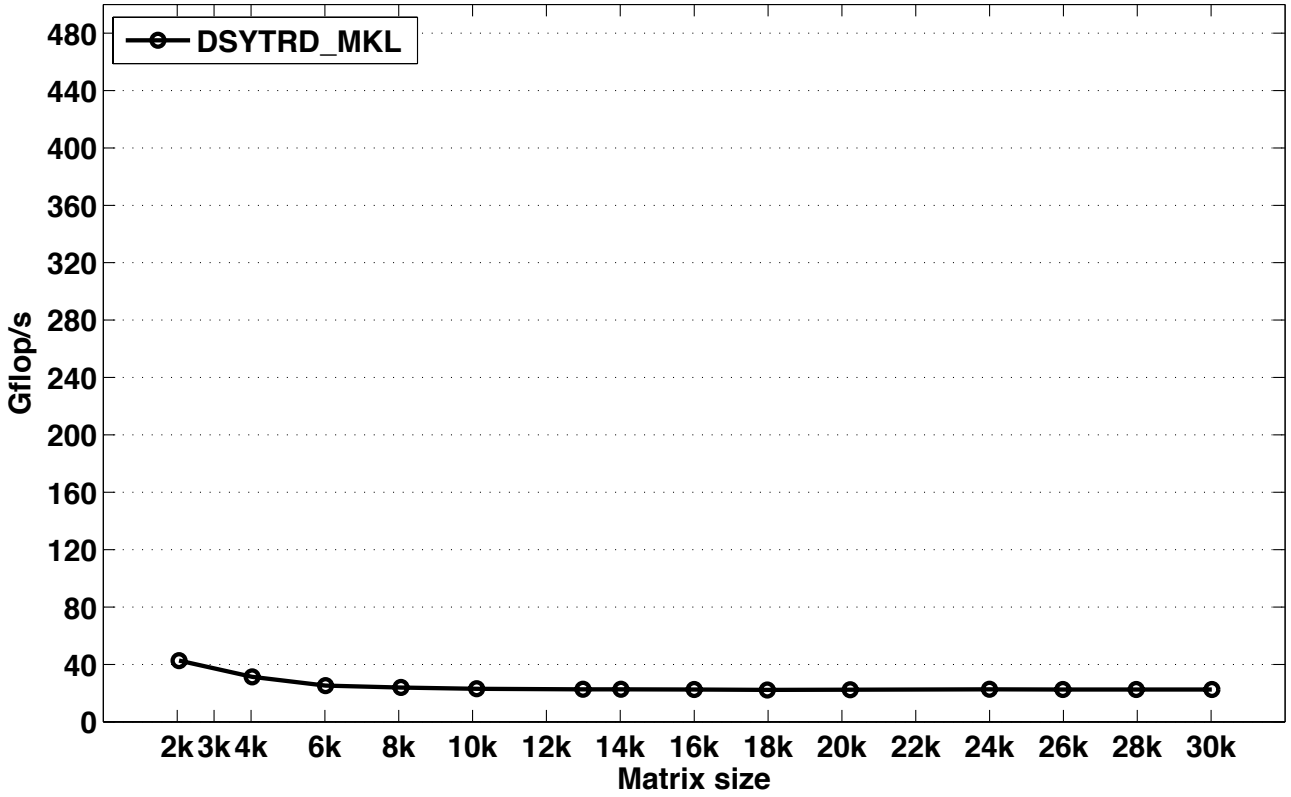
  Two-sided factorizations
  - bases for eigen-solvers
- Block algorithm
  Q – a product of n-1 elementary reflectors
  $Q = H_1 H_2 \dots H_{n-1}$, $H_i = I - \tau_i v_i v_i'$
  $H_1 \dots H_{nb} = I - V T V'$ (*WY transform*; the bases for delayed update or block algorithm)
- **Can we accelerate it ?**
  [similarly to the one-sided using hybrid GPU-based computing]
  [to see **much higher acceleration** due to a removed bottleneck]

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



Toward fast Eigensolver



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

Higher is faster

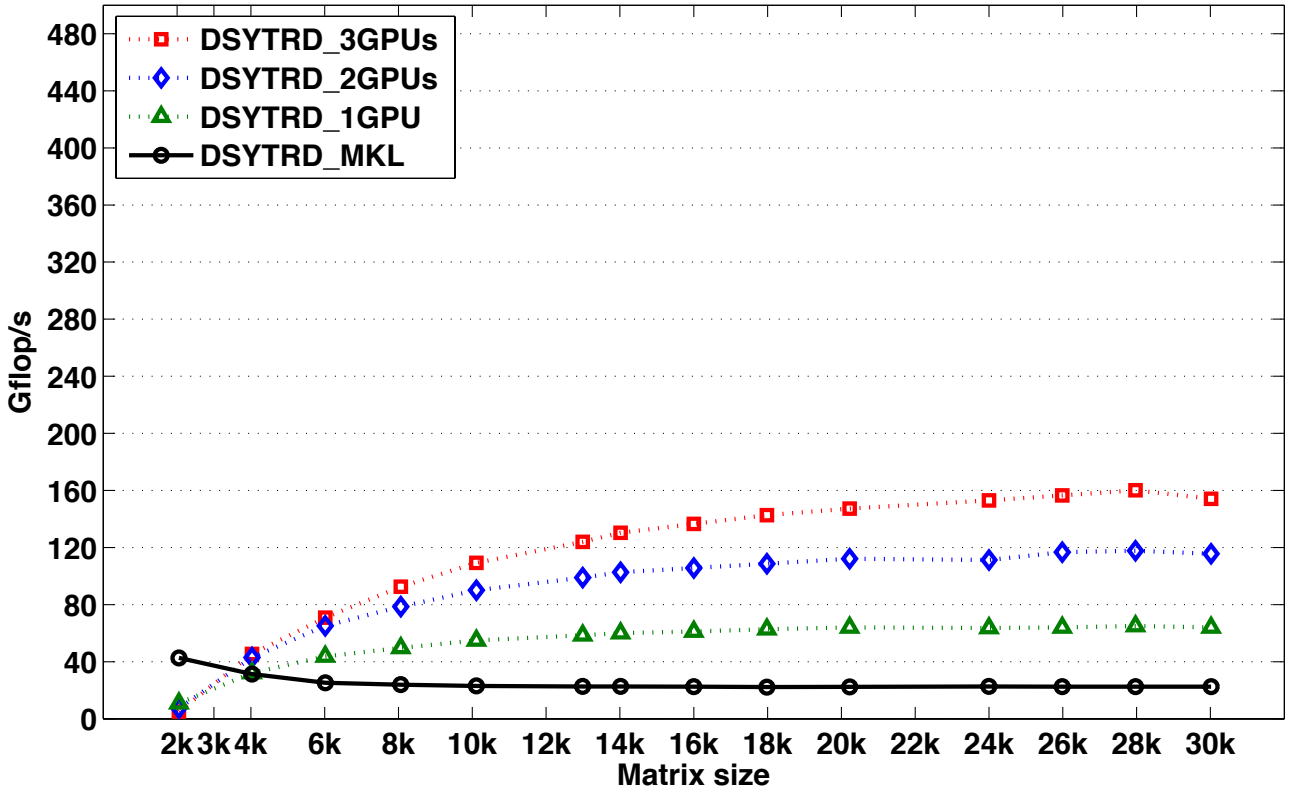
Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

- Characteristics
  - Too many Blas-2 op,
  - Relies on panel factorization,
  - ➔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.

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

Toward fast Eigensolver



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

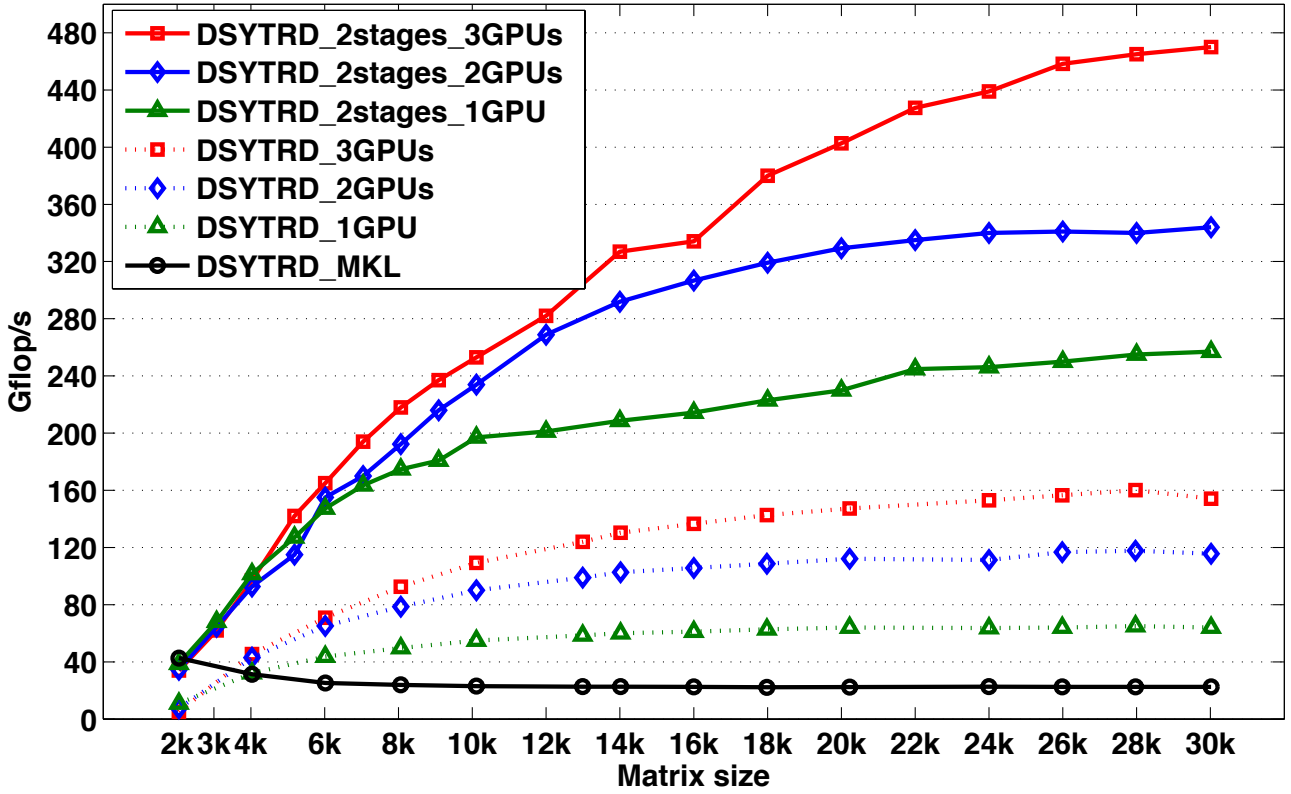
Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

- 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.

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

Toward fast Eigensolver

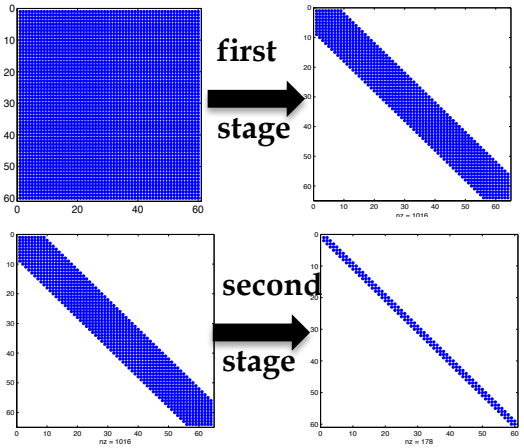


flops formula: $n^3/3*time$
Higher is faster

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

- 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.



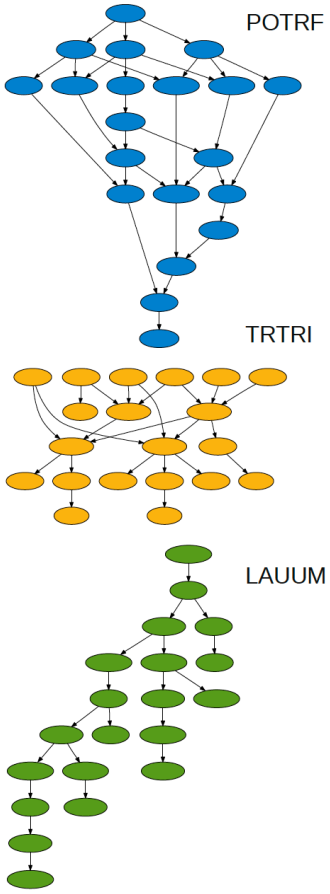
A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks*, ICL Technical report, 03/2012.

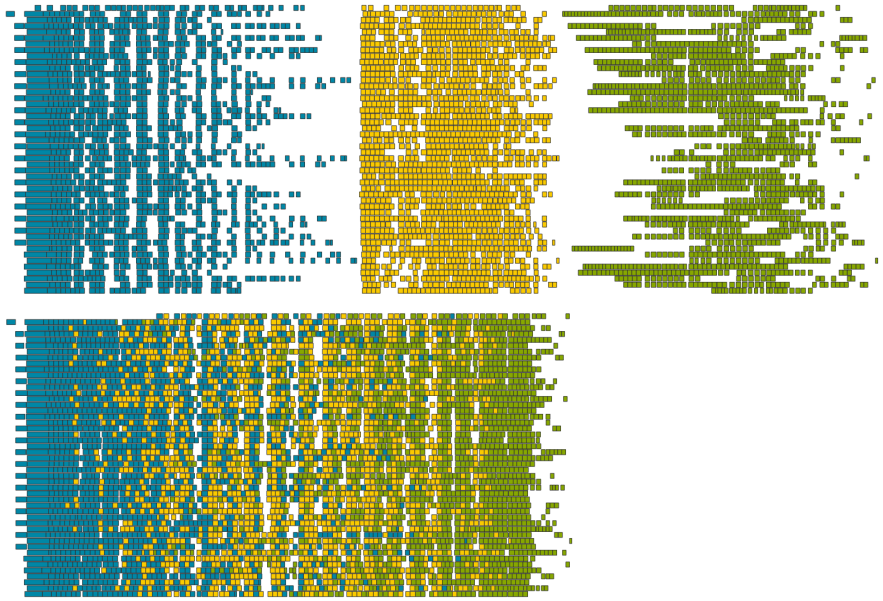
ICL UT INNOVATIVE COMPUTING LABORATORY

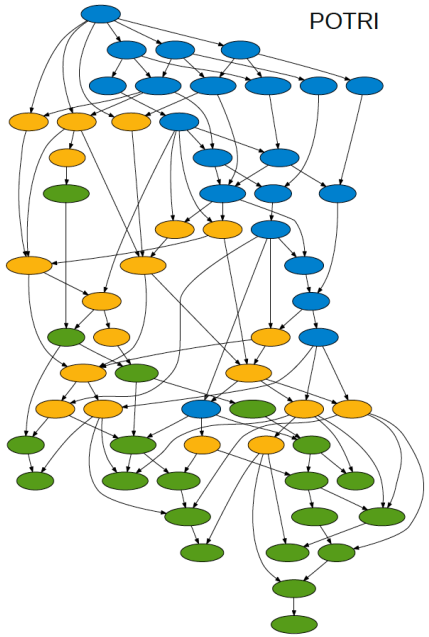
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Current and Future Directions

- **Synchronization avoiding algorithms using Dynamic Runtime Systems**







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

Current and Future Directions

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 and Future Directions

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

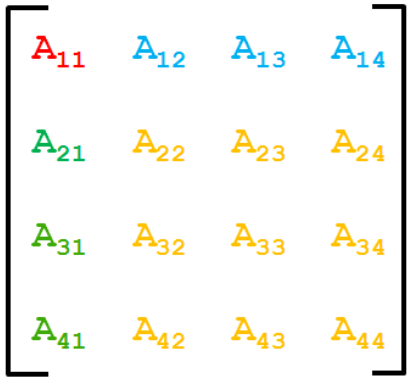
```
for (k = 0; k < min(MT, NT); k++){
    Insert_Task(&cl_zgeqrt, k , k, ...);
    for (n = k+1; n < NT; n++)
            Insert_Task(&cl_zunmqr, k, n, ...);
    for (m = k+1; m < MT; m++){
            Insert_Task(&cl_ztsqrt, m, k, ...);
            for (n = k+1; n < NT; n++)
                    Insert_Task(&cl_ztsmqr,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

Use runtime systems to schedule tasks execution on heterogeneous architectures (multicore + GPUs)

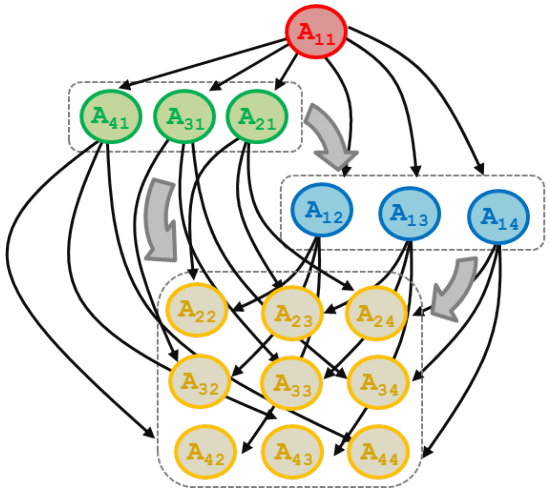
Current and Future Directions

- **Sparse linear algebra**
  - Direct multi-frontal solvers
  - Iterative linear solvers and eigenproblem solvers

Sparse / Dense Matrix System



DAG-based solvers



GPU algorithms/implementations of DAG tasks

- **LU, QR, and Cholesky** on small diagonal matrices
- **Triangular matrix solvers** (TRSM) and **panel LU, QR** (in combination with A_{11})
- **TRSMs on rectangular matrices** (or on matrices in compressed form; batched TRSMs for parallel TRSMs)
- **Updates (Schur complement) for matrices in compressed form** (batched matrix-matrix multiplication and other BLAS, QR and SVD)

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

Collaborators and Support

Microsoft

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

ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science