THE UNIVERSITY of TENNESSEE
NATIONAL LEADERSHIP IN HPC

MAGMA
1.3
SC12
Salt Lake City, Utah
Release

MAGMA: a New Generation of Linear Algebra Libraries for GPU and Multicore Architectures

Jack Dongarra

T. Dong, M. Gates, A. Haidar, S. Tomov, and I. Yamazaki

University of Tennessee, Knoxville

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

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 1.3**
  - For NVIDIA CUDA GPUs on shared memory systems
  - 80+ hybrid algorithms have been developed (total of 320+ routines)
    - 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

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

Key Features of MAGMA 1.3

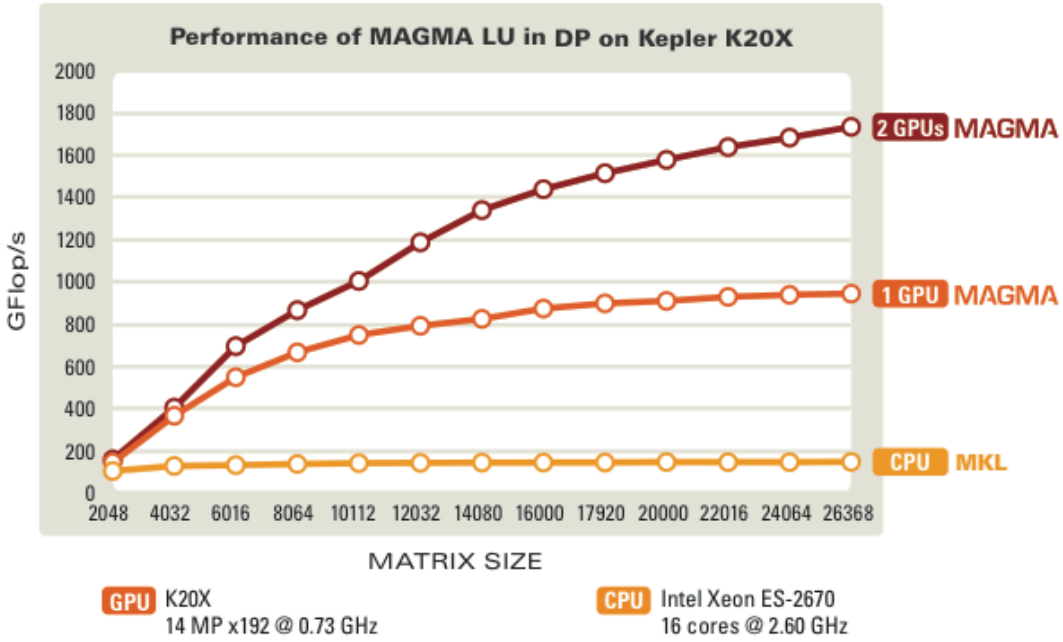
- High performance
- Multiple precision support (Z, C, D, S, and MP)
- Hybrid algorithms
- Out-of-GPU memory algorithms
- MultiGPU support

Key Features of MAGMA 1.3

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

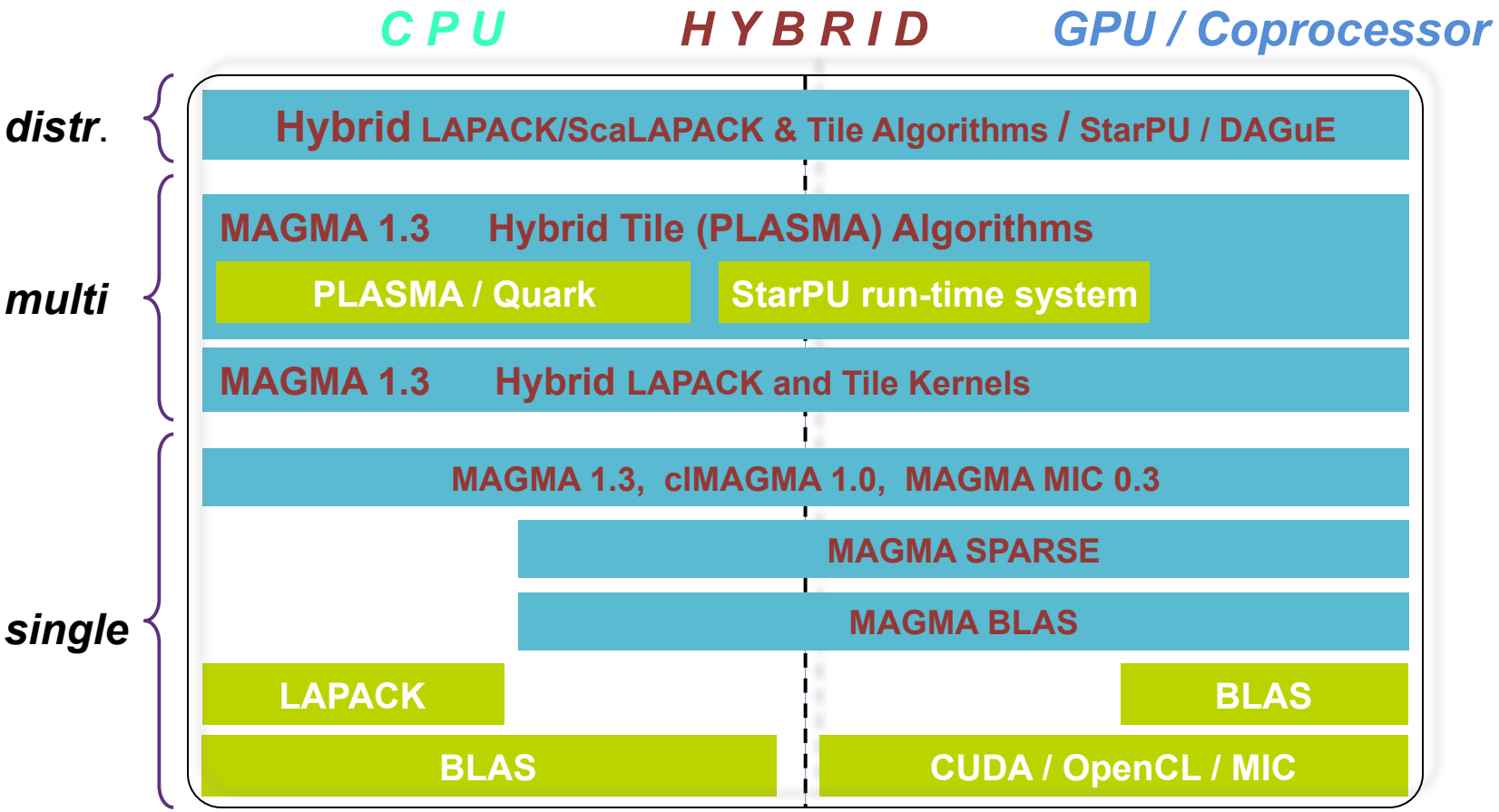
- **MAGMA 1.3** FOR **CUDA**
- **clMAGMA 1.0** FOR **OpenCL**
- **MAGMA MIC 0.3** FOR **Intel Xeon Phi**

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

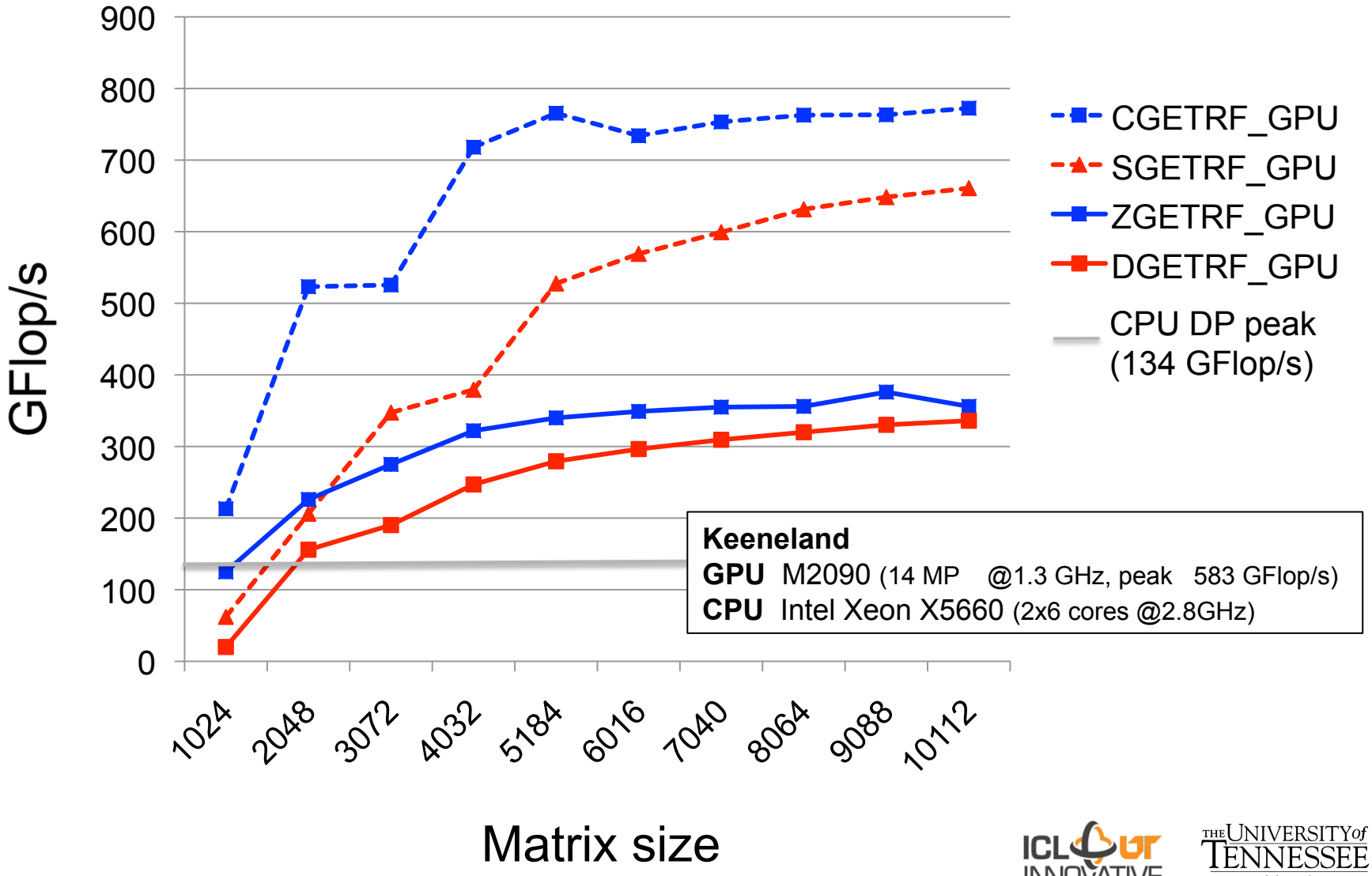
MAGMA Software Stack



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

Multiple precision support

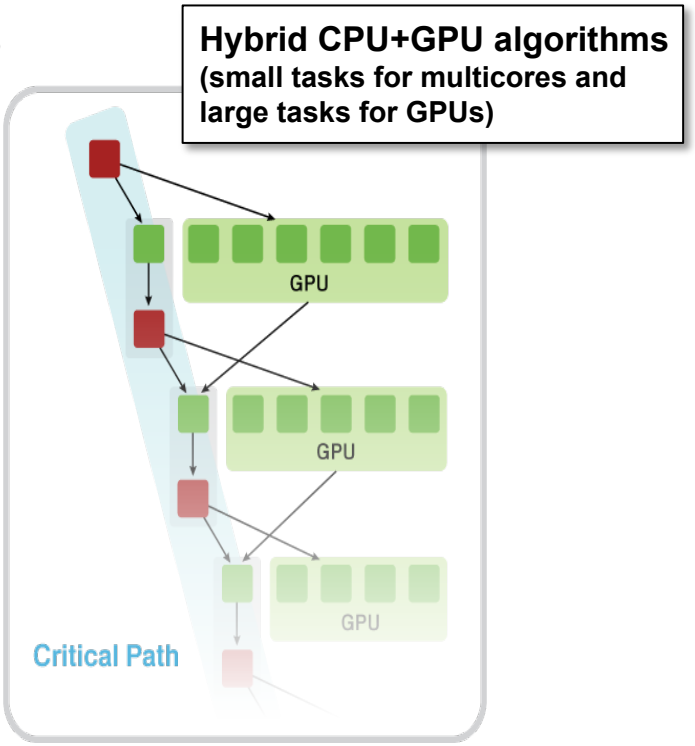
Performance of the LU factorization in various precisions



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

A Hybrid Algorithm Example

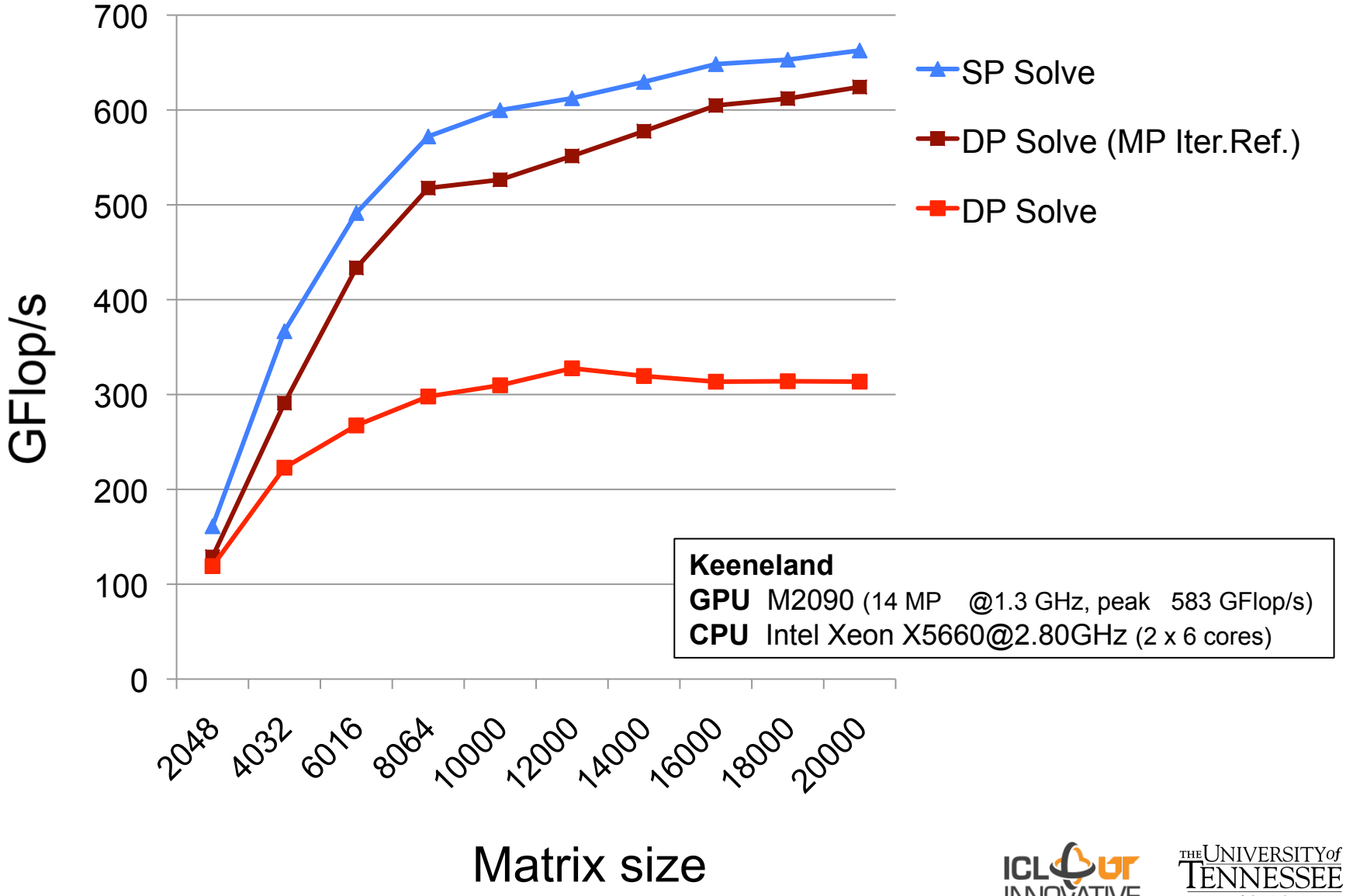
- Left-looking hybrid Cholesky factorization in MAGMA

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

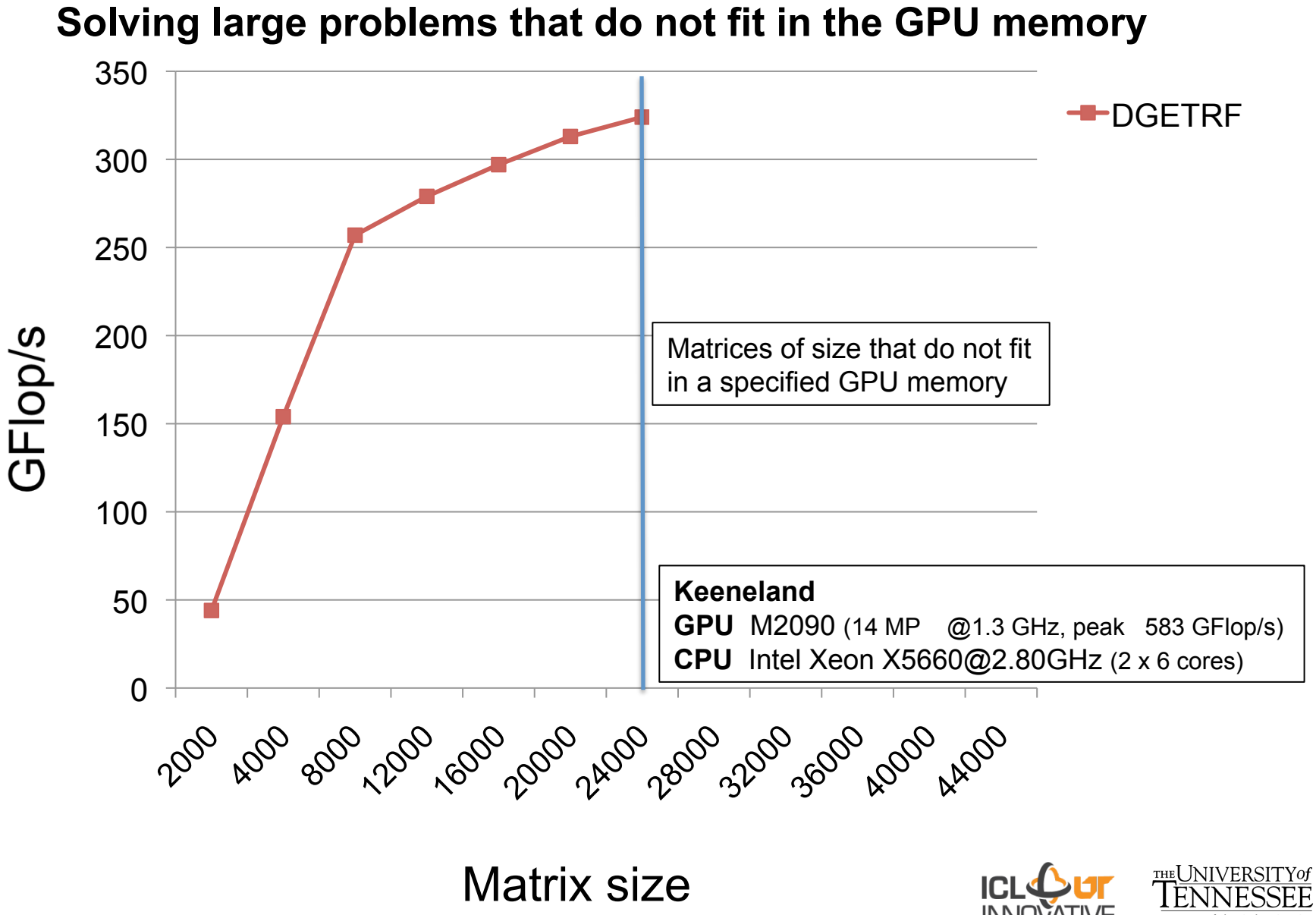
- The difference with LAPACK – the 4 additional lines in red
- Line 8 (done on CPU) is overlapped with work on the GPU (from line 6)

Mixed precision iterative refinement

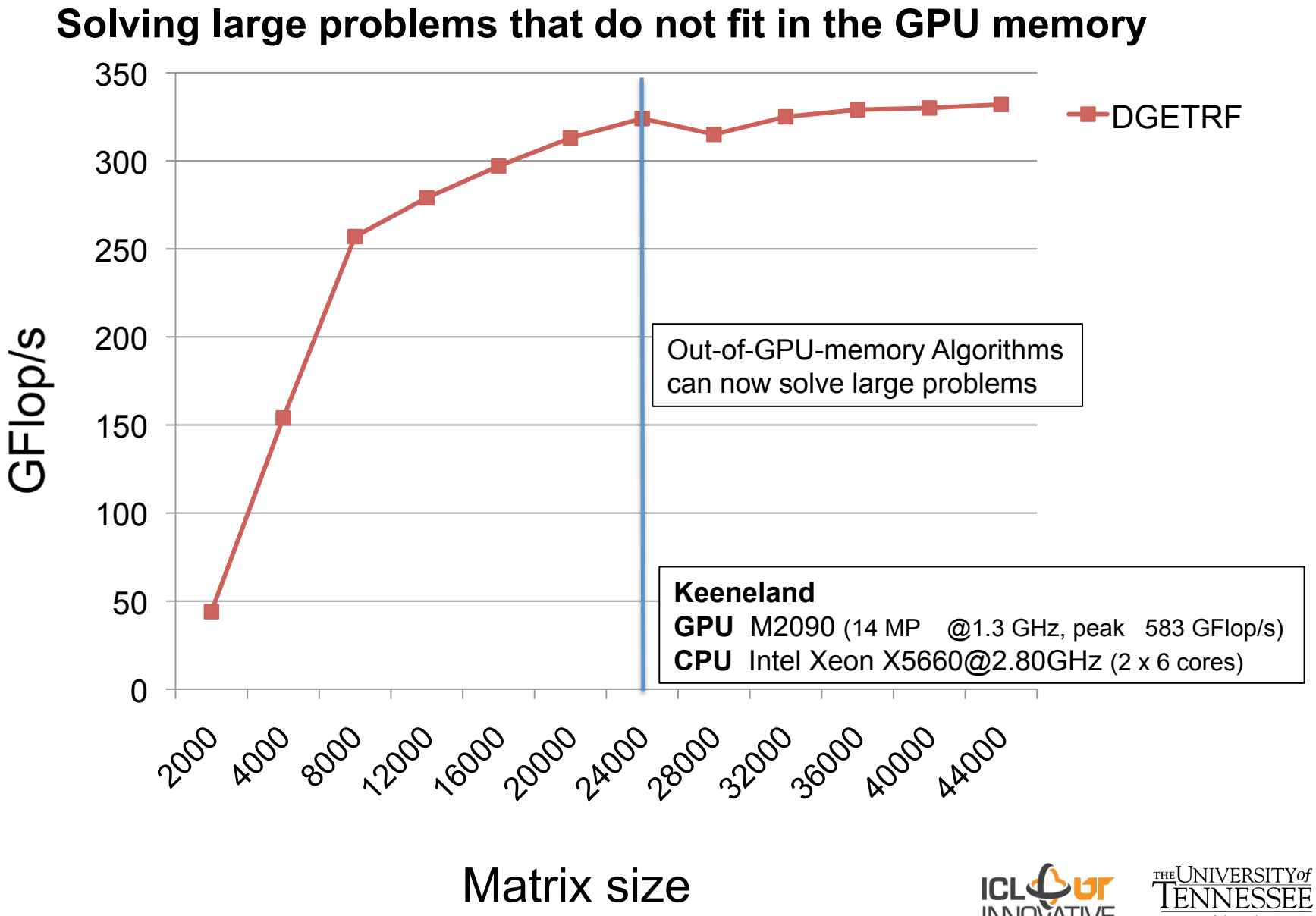
Solving general dense linear systems using mixed precision iterative refinement



Out of GPU Memory Algorithms

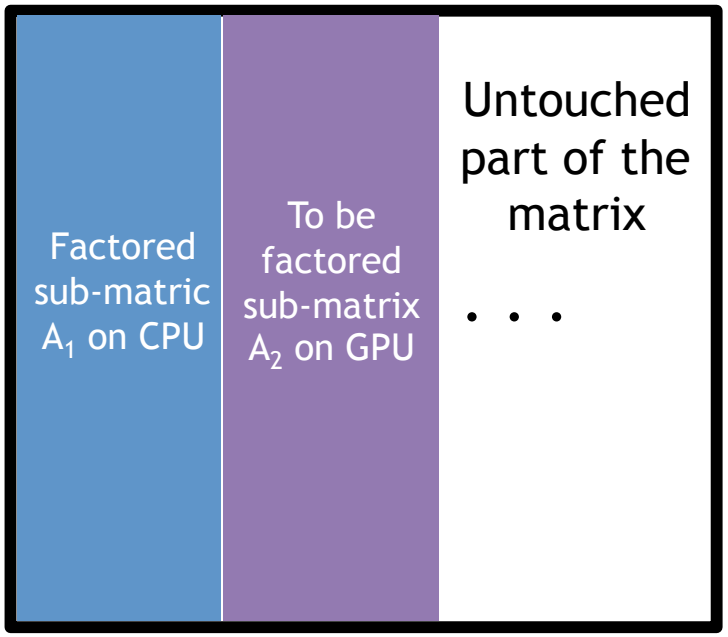


Out of GPU Memory Algorithms



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



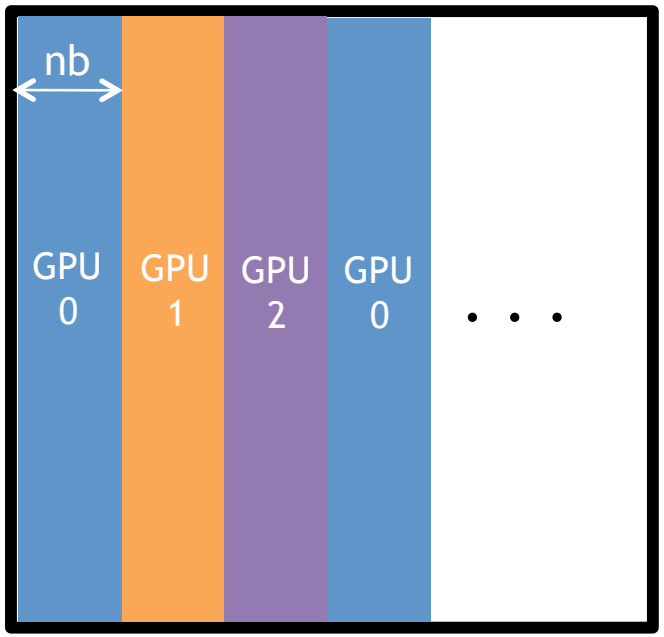
1) Copy A_2 to the GPU
2) Update A_2 using A_1 (a panel of A_1 at a time)
3) Factor the updated A_2 using existing hybrid code
4) Copy factored A_2 to the CPU

Trivially extended to multiGPUs:

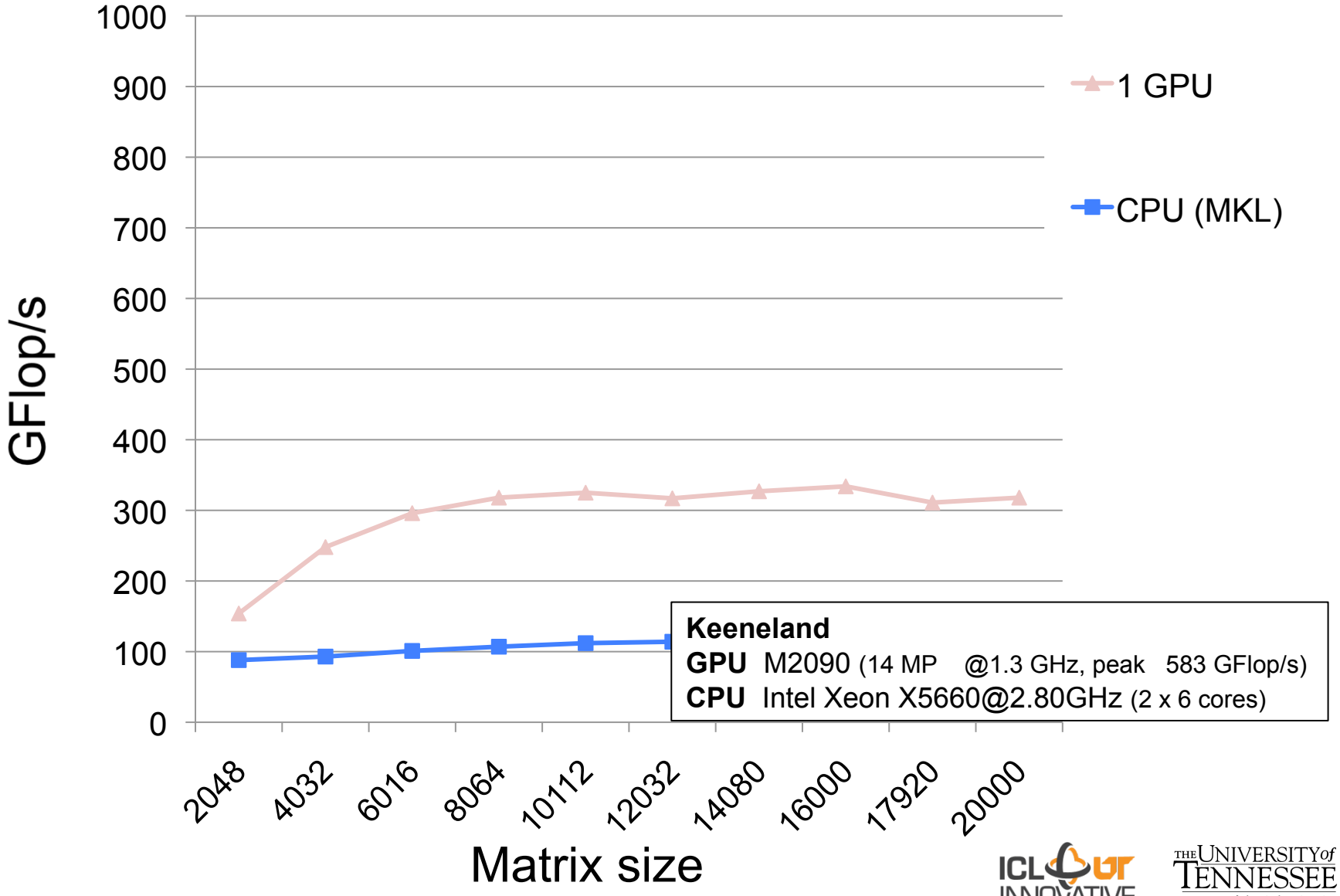
A_2 is "larger" with 1-D block cyclic distribution, again reusing existing algorithms

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



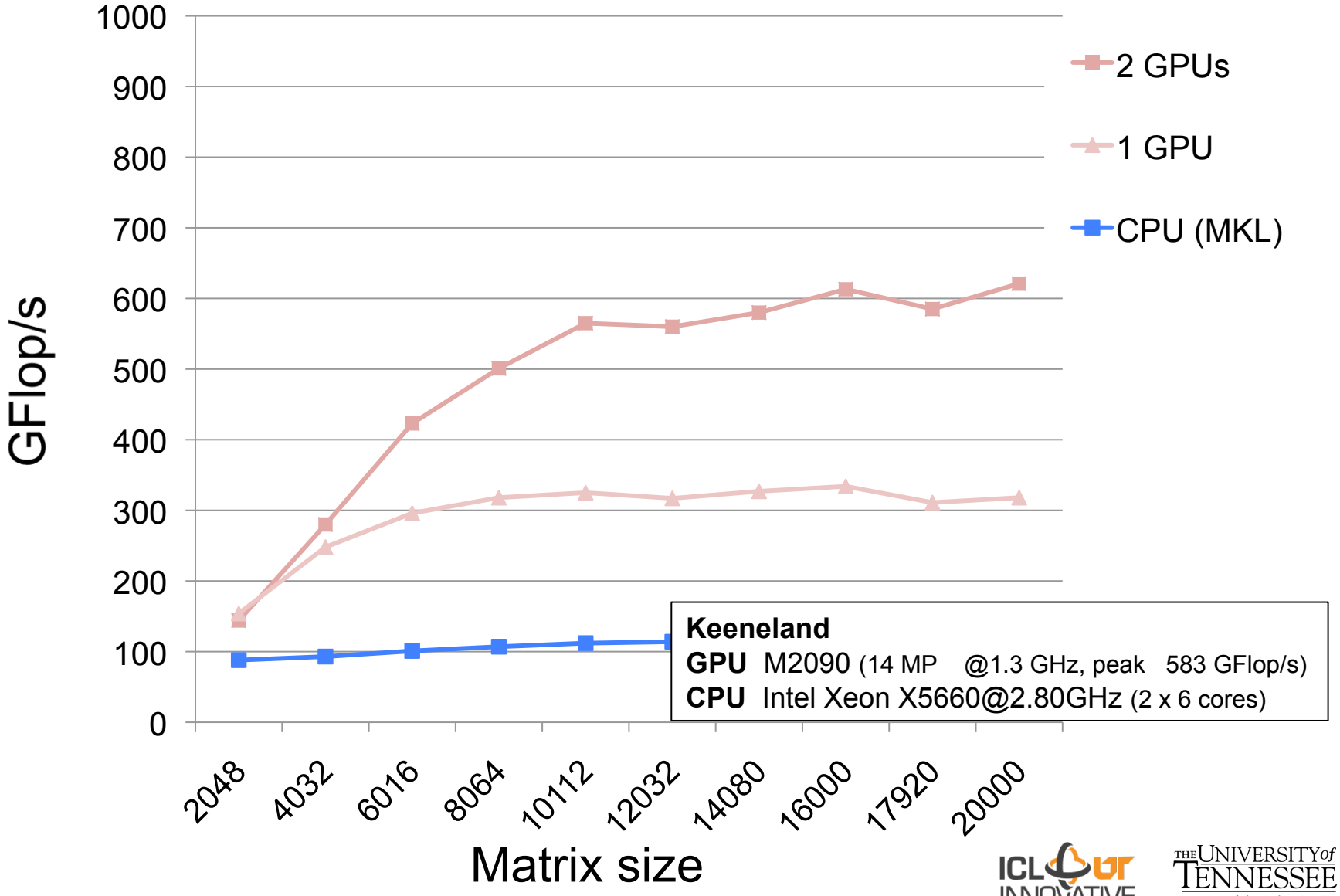
LU Factorization on multiGPUs in DP



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

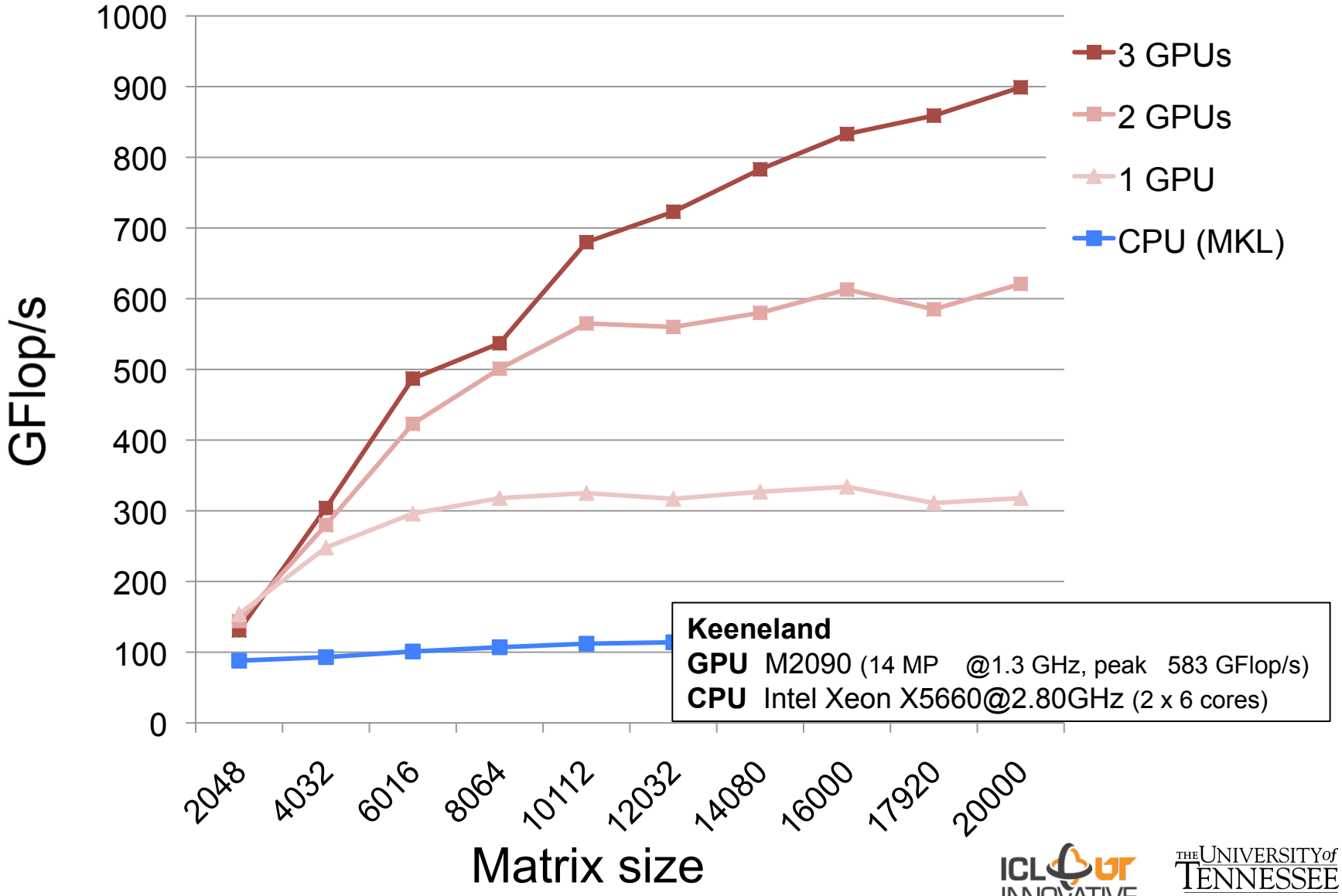
LU Factorization on multiGPUs in DP



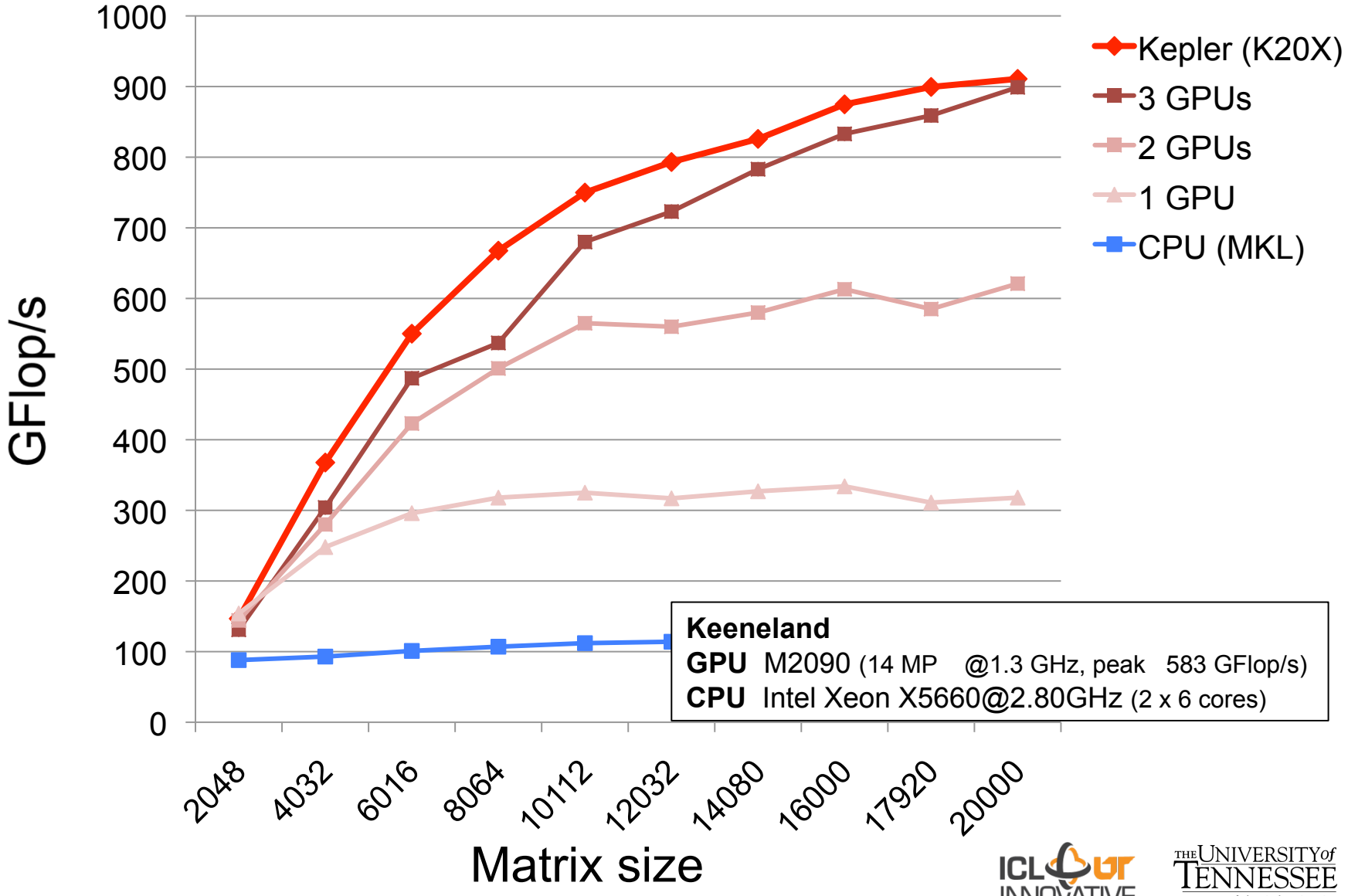
ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

LU Factorization on multiGPUs in DP



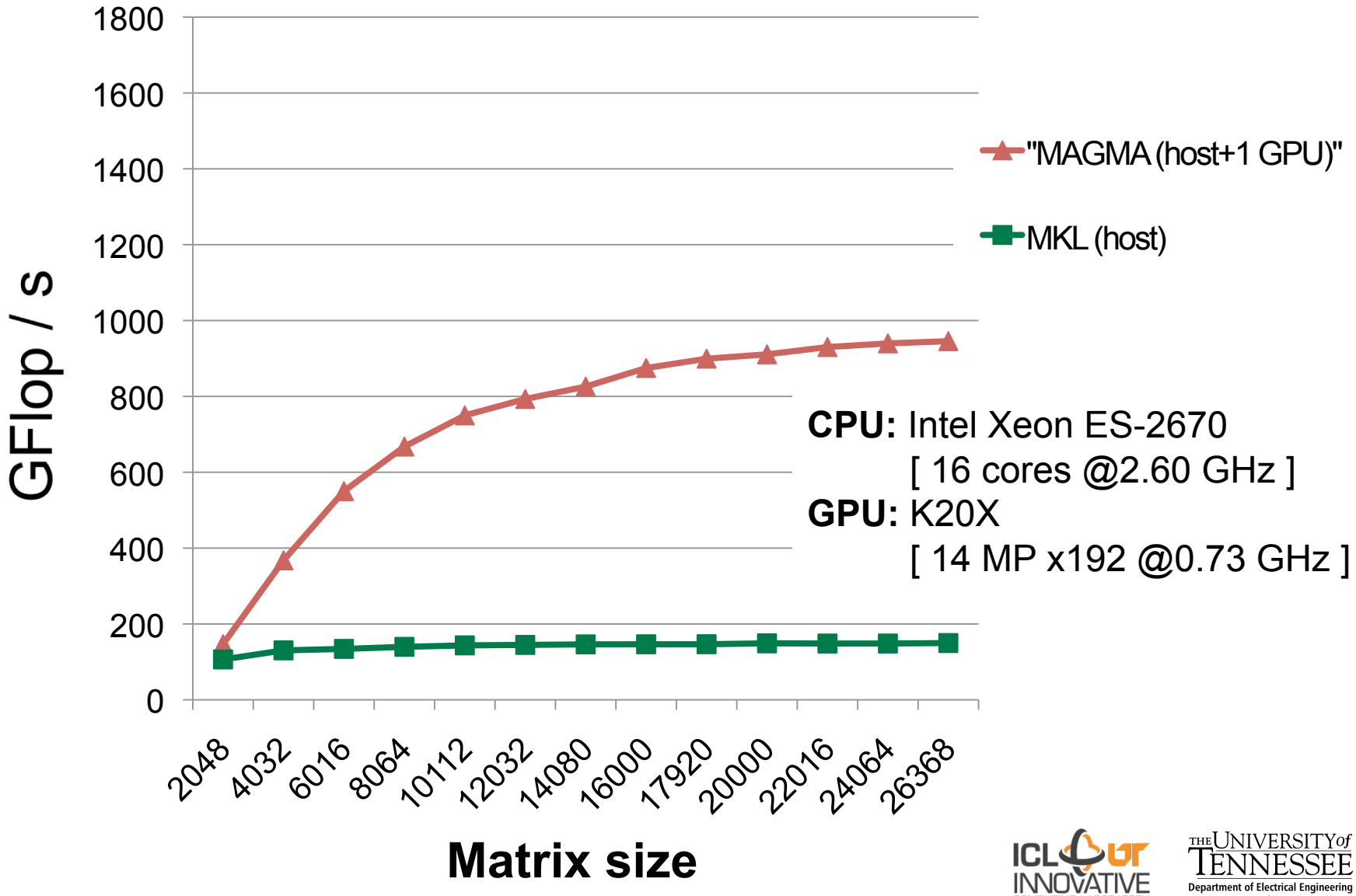
LU Factorization on Kepler in DP



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

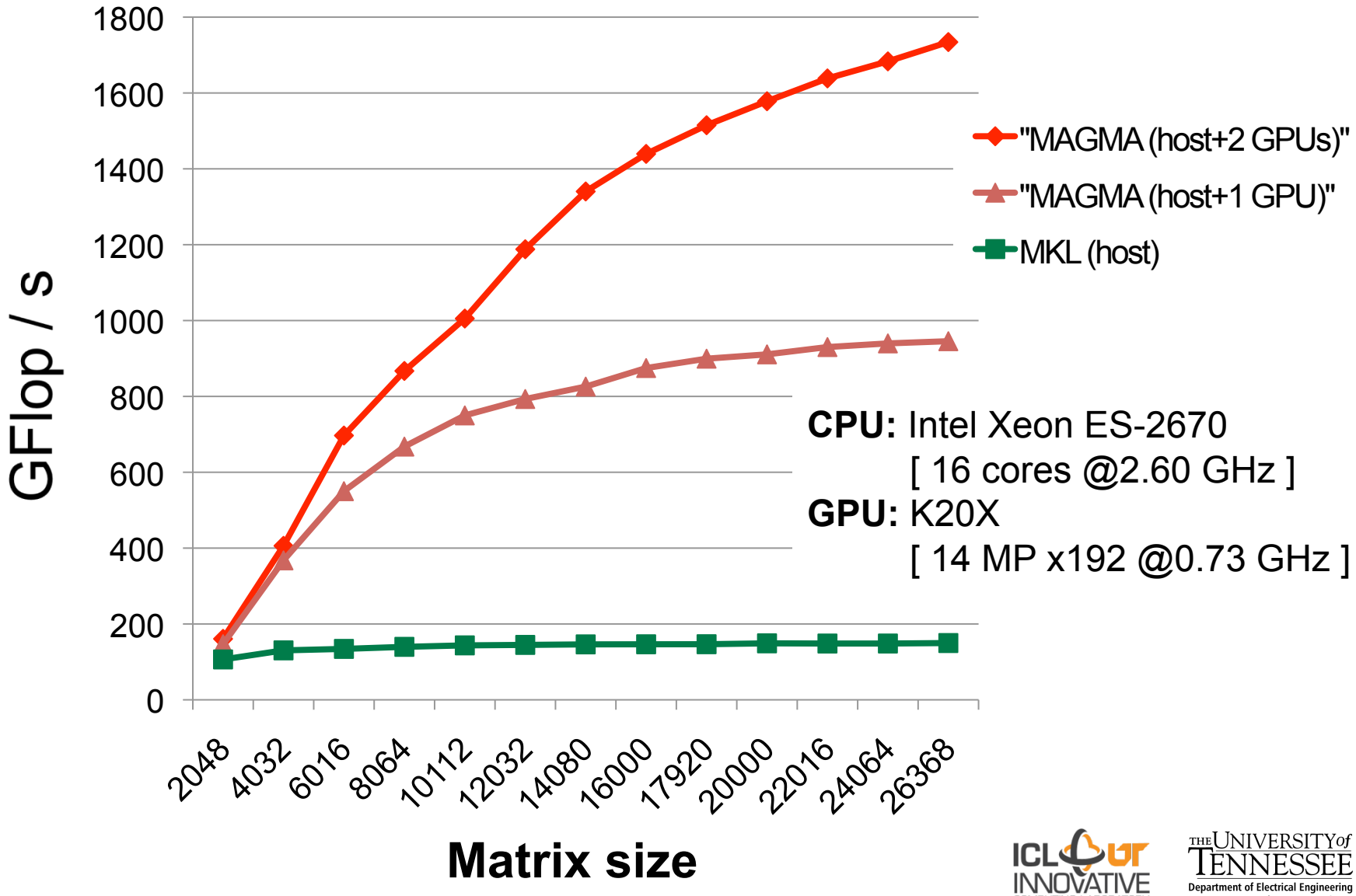
LU Scalability on Kepler in DP



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

LU Scalability on Kepler in DP



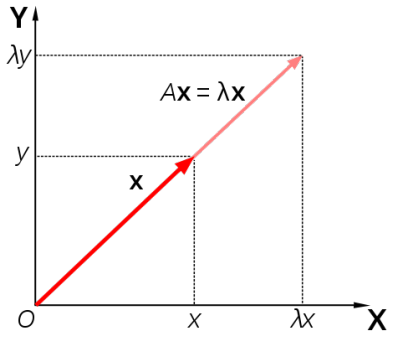
ICL UT INNOVATIVE COMPUTING LABORATORY

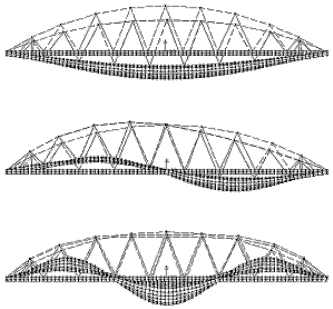
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

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

    . . .





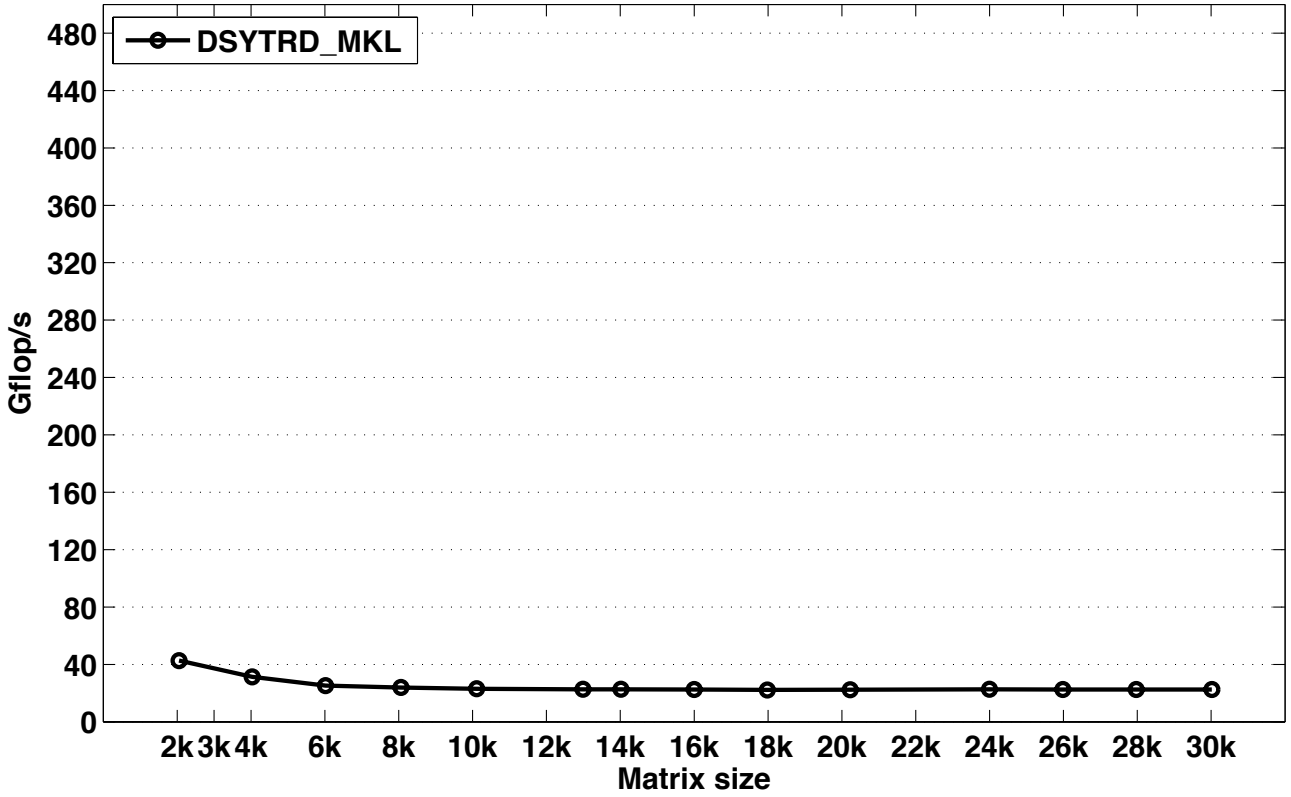
- **Need to solve it fast**

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.

Toward fast Eigensolver



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

Higher is faster

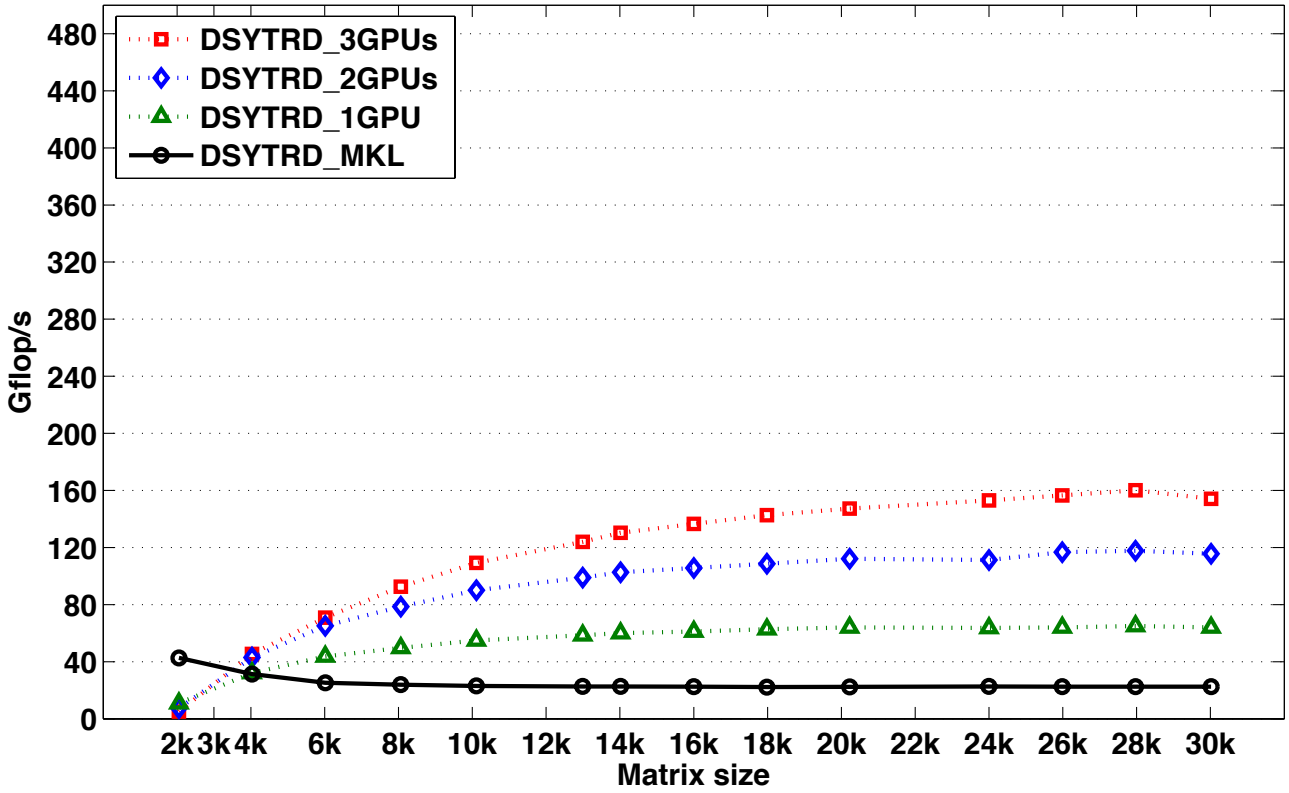
Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

- 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

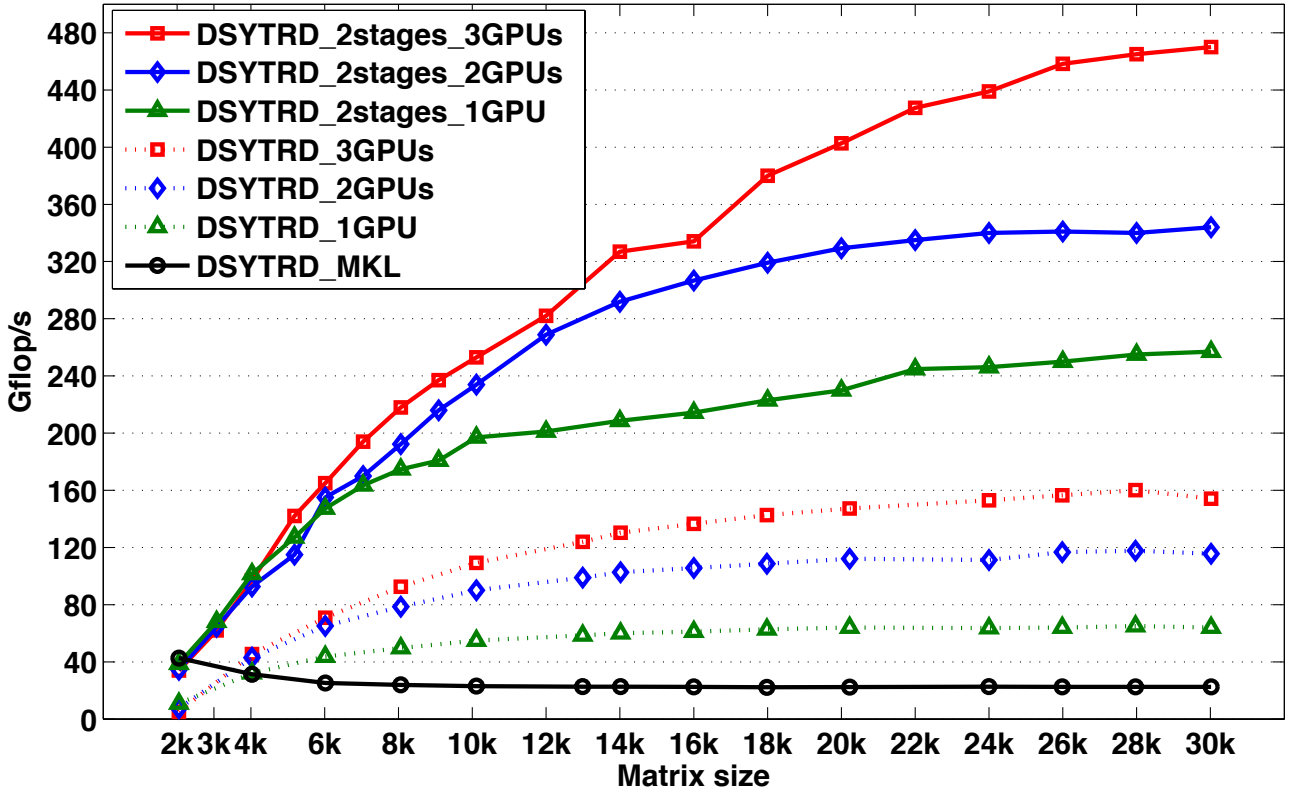
Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

- 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

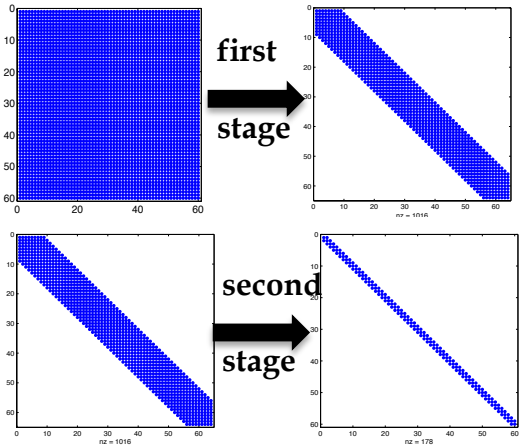


flops formula: $n^3/3*time$
Higher is faster

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.



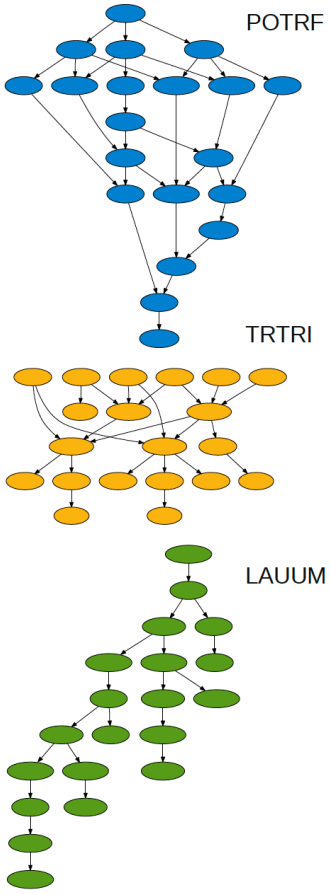
A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks*, ICL Technical report, 03/2012.

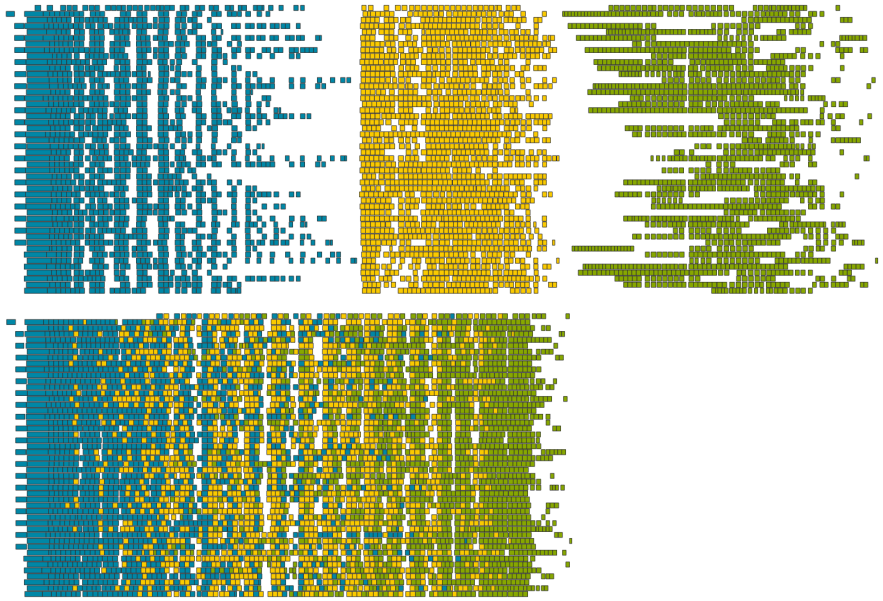
ICL UT INNOVATIVE COMPUTING LABORATORY

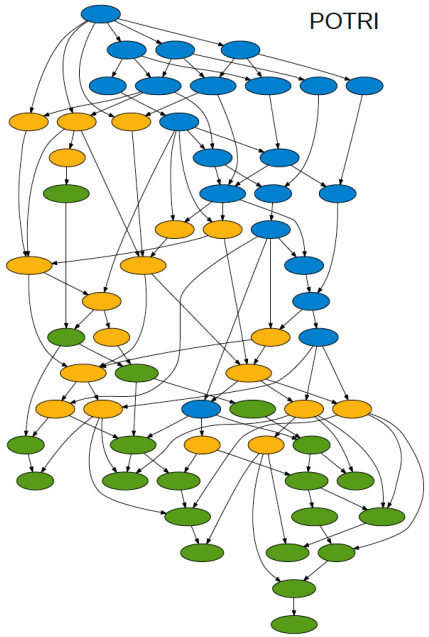
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

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

Use of **StarPU** runtime system to exploit heterogeneous architectures (multicore + GPUs) or **Quark** if only targetting multicore.

Matrices Over Runtime Systems @ Exascale (MORSE)

- Mission statement: "Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems”
- Runtime challenges due to the ever growing hardware complexity
- Algorithmic challenges to exploit the hardware capabilities at most
- Integrated into MAGMA software stack
  - MAGMA 1.3 includes Level 3 BLAS, linear and least squares solvers

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