Linear Algebra Software Technologies for Exascale

Stan Tomov[1]

A. Abdelfattah[1], A. Haidar[2], M. Gates[1], I. Yamazaki[3], A. Ayala[1], Jack Dongarra[1]

[1] Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

[2] Nvidia Corporation

[3] Sandia National Laboratory

OLCF User Meeting
Oak Ridge, TN
May 23, 2019

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

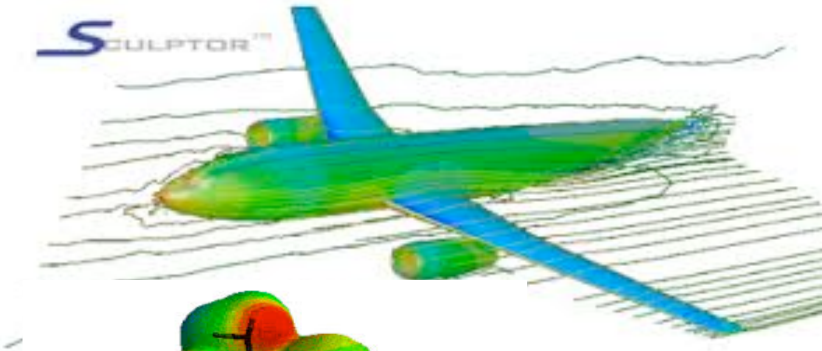
Outline

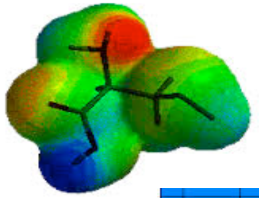
Some linear algebra software technologies for Exascale

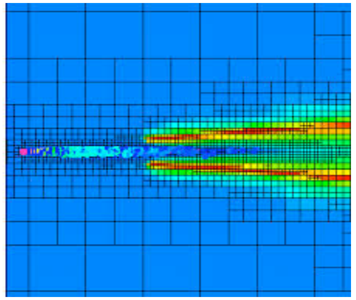
- Mixed-precision solvers using GPU Tensor Cores
- Batched linear algebra for many small problems
- Redesign of LAPACK and ScaLAPACK for new architectures
  - MAGMA and SLATE libraries
- Accelerating memory-bound codes
  - The case of redesigning 3D FFTs for GPU-only execution

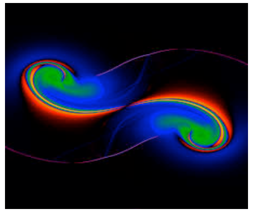
Linear Algebra in Applications

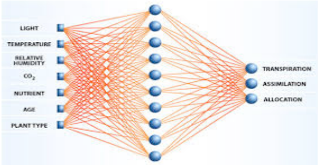
- **Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications:**
  - **Linear systems:** Solve $Ax = b$
    - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
  - **Least squares:** Find x to minimize $|| Ax – b ||$
    - Computational statistics (e.g., linear least squares or ordinary least squares), econometrics, control theory, signal processing, curve fitting, and many more
  - **Eigenproblems:** Solve $Ax = \lambda x$
    - Computational chemistry, quantum mechanics, material science, face recognition, PCA, data-mining, marketing, Google Page Rank, spectral clustering, vibrational analysis, compression, and many more
  - **SVD:** $A = U \Sigma V^*$ ($Au = \sigma v$ and $A^*v = \sigma u$)
    - Information retrieval, web search, signal processing, big data analytics, low rank matrix approximation, total least squares minimization, pseudo-inverse, and many more
  - **Many variations depending on structure of A**
    - A can be symmetric, positive definite, tridiagonal, Hessenberg, banded, sparse with dense blocks, etc.
- **Batched LA on many small DLA problems**
- **FFTs**











ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

UTK/ICL involved in ECP projects providing various high-performance linear algebra functionalities

- SLATE
  - Provides SOA algorithmic and technology innovation in dense linear algebra software
- FFT-ECP
  - Design and implement a sustainable 2D/3D FFT library for Exascale systems
- xSDK
  - Provides interoperability across existing numerical libraries hypre, PETSc, SuperLU, Trilinos, MAGMA, PLASMA and DPLASMA
- CEED
  - Co-Design next-generation discretization software and algorithms that will enable a wide range of FE applications

MAGMA Today

MAGMA – provides highly optimized LA well beyond LAPACK for GPUs;
– research vehicle for LA on new architectures for a number of projects.

for architectures in
{ CPUs + Nvidia GPUs (CUDA),
CPUs + AMD GPUs (OpenCL),
CPUs + Intel Xeon Phis,
manycore (native: GPU or KNL/CPU),
embedded systems, combinations, and
software stack, e.g., since CUDA x}

for precisions in
{ s, d, c, z,
half-precision (FP16),
mixed, … }

for interfaces
{ heterogeneous CPU/GPU, native, … }

- **LAPACK**
- **BLAS**
- **Batched LAPACK**
- **Batched BLAS**
- **Sparse**
- **Tensors**
- **MAGMA-DNN**
- **Templates**
- **…**

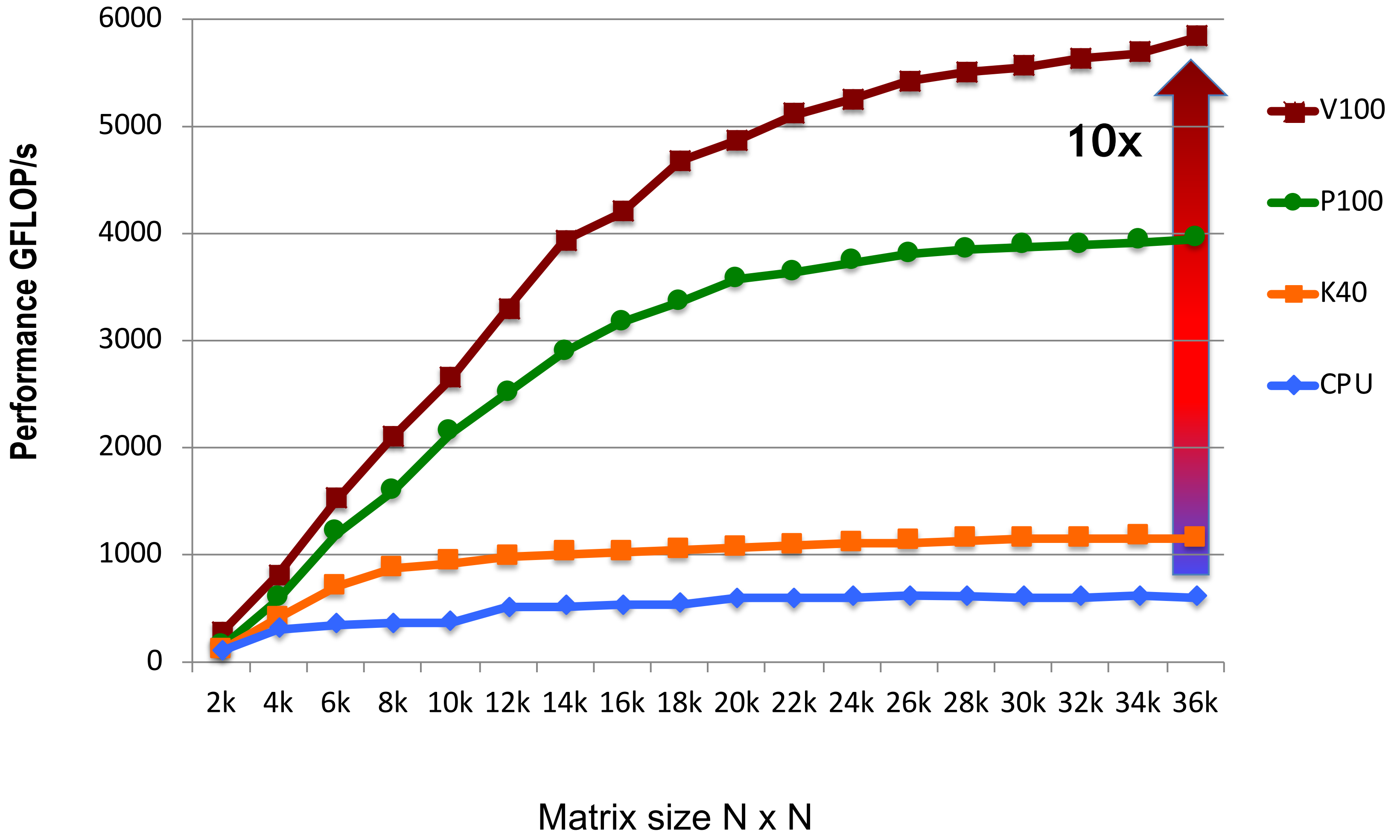
- **Collaboration and support from vendors**
  NVIDIA, Intel, and AMD
- **Two releases per year**
  Latest MAGMA 2.5.1
  Number of downloads per release ~ **4K**
  Highly tuned for latest GPUs and heterogeneous architectures
- **MAGMA Forum**:
  3,248 + 279 (3,527) posts in **869 + 78 (955) topics**, **1,841 + 1057 (2,898) users**
- **MAGMA is incorporated/used in**
  MATLAB (as of the R2010b),
  contributions in **CUBLAS** and **MKL**,
  AMD, **Siemens** (in NX Nastran 9.1), **ArrayFire,**
  ABINIT, **Quantum-Espresso**, **R** (in HiPLAR & CRAN),
  SIMULIA (Abaqus), **MSC Software** (Nastran and Marc),
  Cray (in LibSci for accelerators **libsci_acc**),
  Nano-TCAD (Gordon Bell finalist),
  Numerical Template Toolbox (**Numscale**), and others.
- **MAGMA used in ECP –** CEED, PEEKS, xSDK, ALEXA/TASMANIAN, SLATE & FFT

Why use GPUs in HPC?

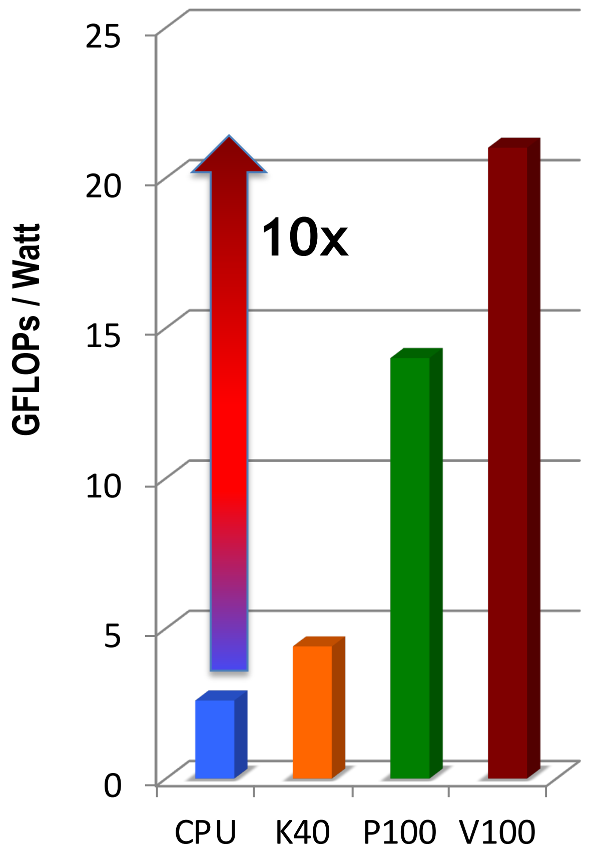
PERFORMANCE & ENERGY EFFICIENCY

MAGMA 2.5.1 LU factorization in double precision arithmetic

CPU Intel Xeon E5-2650 v3 (Haswell) 2x10 cores @ 2.30 GHz
K40 NVIDIA Kepler GPU 15 MP x 192 @ 0.88 GHz
P100 NVIDIA Pascal GPU 56 MP x 64 @ 1.19 GHz
V100 NVIDIA Volta GPU 80 MP x 64 @ 1.38 GHz



Energy efficiency
(under ~ the same power draw)



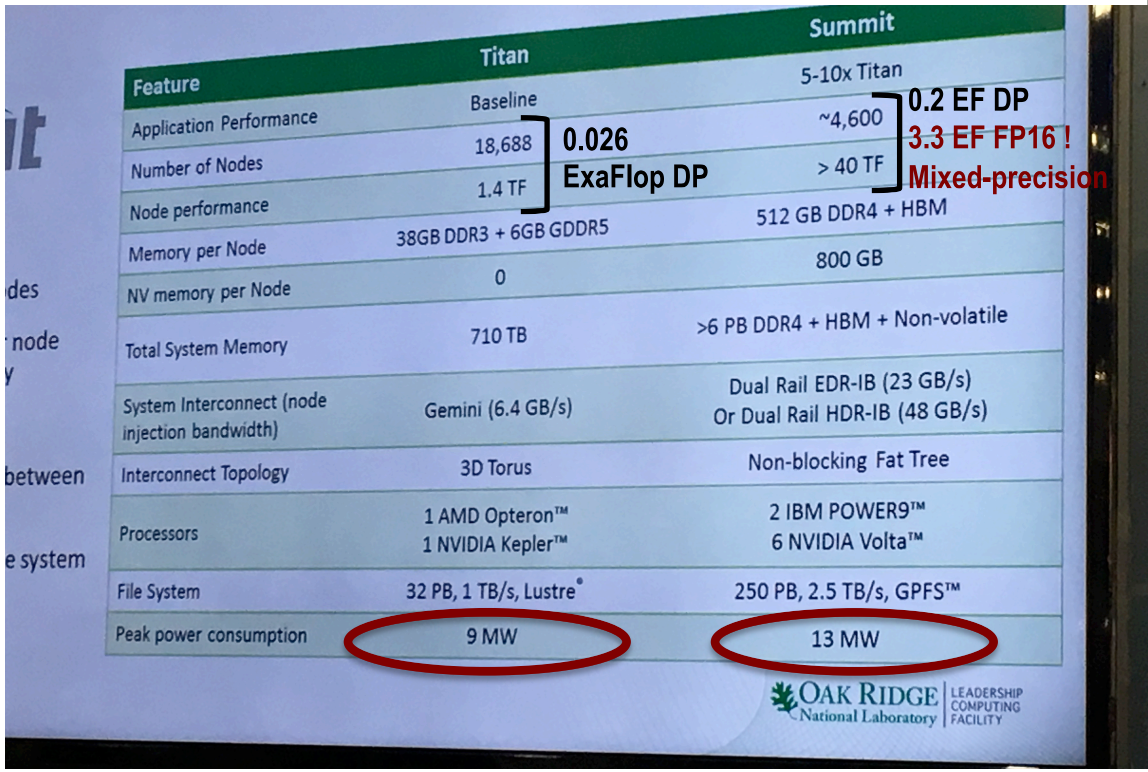
ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Mixed-precision LA

- V100 GPUs have hardware acceleration for FP16 arithmetic
  - Tensor Cores (TC) capable of **125 Tflop/s** in FP16

ORNL Summit Supercomputer

200-Petaflops System Debuts as America's Top Supercomputer for Science

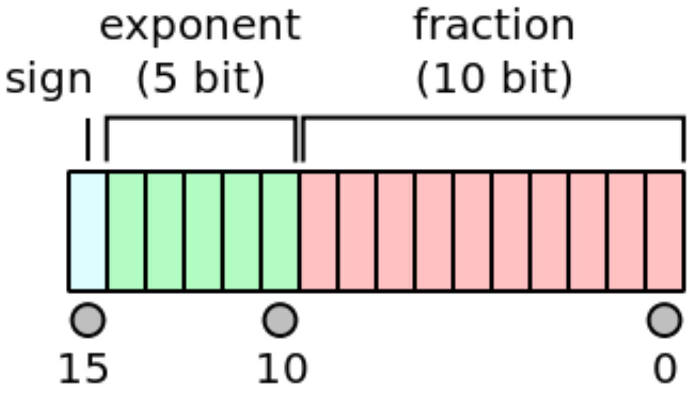


Feature	Titan	Summit
Application Performance	Baseline	5-10x Titan
Number of Nodes	18,688	~4,600
Node performance	1.4 TF	> 40 TF
Memory per Node	38GB DDR3 + 6GB GDDR5	512 GB DDR4 + HBM
NV memory per Node	0	800 GB
Total System Memory	710 TB	>6 PB DDR4 + HBM + Non-volatile
System Interconnect (node injection bandwidth)	Gemini (6.4 GB/s)	Dual Rail EDR-IB (23 GB/s) Or Dual Rail HDR-IB (48 GB/s)
Interconnect Topology	3D Torus	Non-blocking Fat Tree
Processors	1 AMD Opteron™ 1 NVIDIA Kepler™	2 IBM POWER9™ 6 NVIDIA Volta™
File System	32 PB, 1 TB/s, Lustre®	250 PB, 2.5 TB/s, GPFS™
Peak power consumption	9 MW	13 MW

https://www.ornl.gov/news/ornl-launches-summit-supercomputer

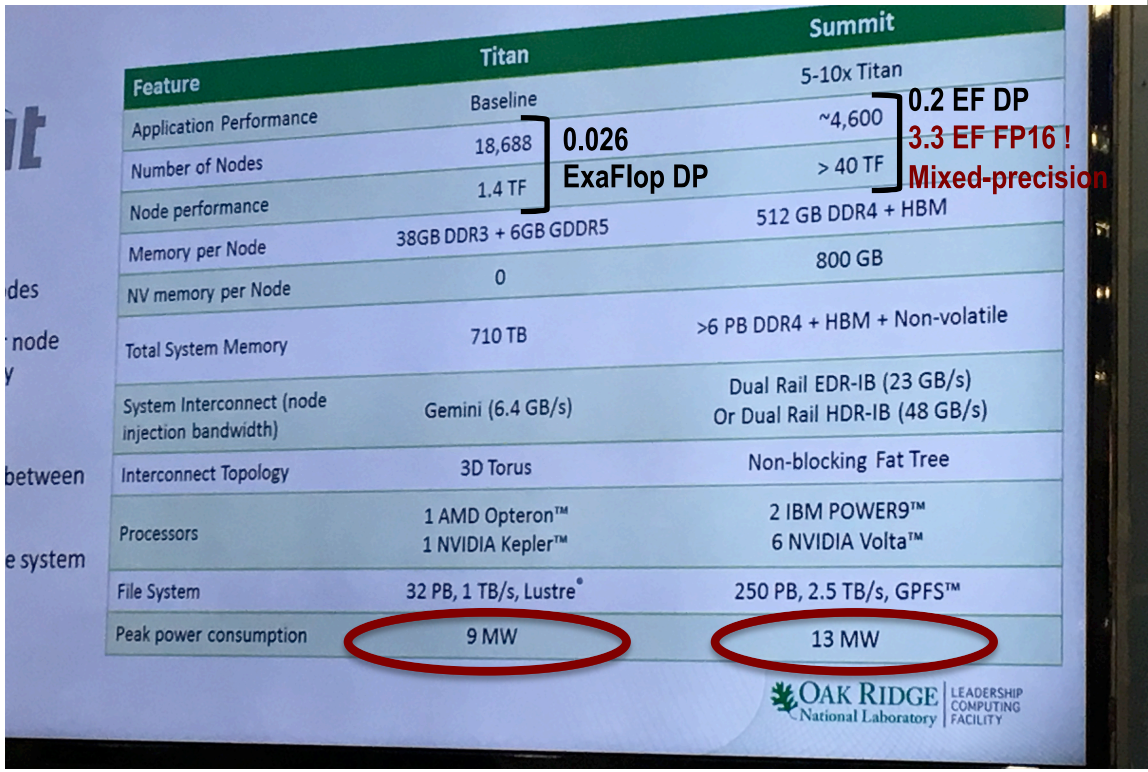
Mixed-precision LA

- V100 GPUs have hardware acceleration for FP16 arithmetic
  - Tensor Cores (TC) capable of **125 Tflop/s** in FP16
- Can we use it for scientific computing?
  - Max representable value is 65504
  - About 3.3 decimal digits of precision



ORNL Summit Supercomputer

200-Petaflops System Debuts as America's Top Supercomputer for Science



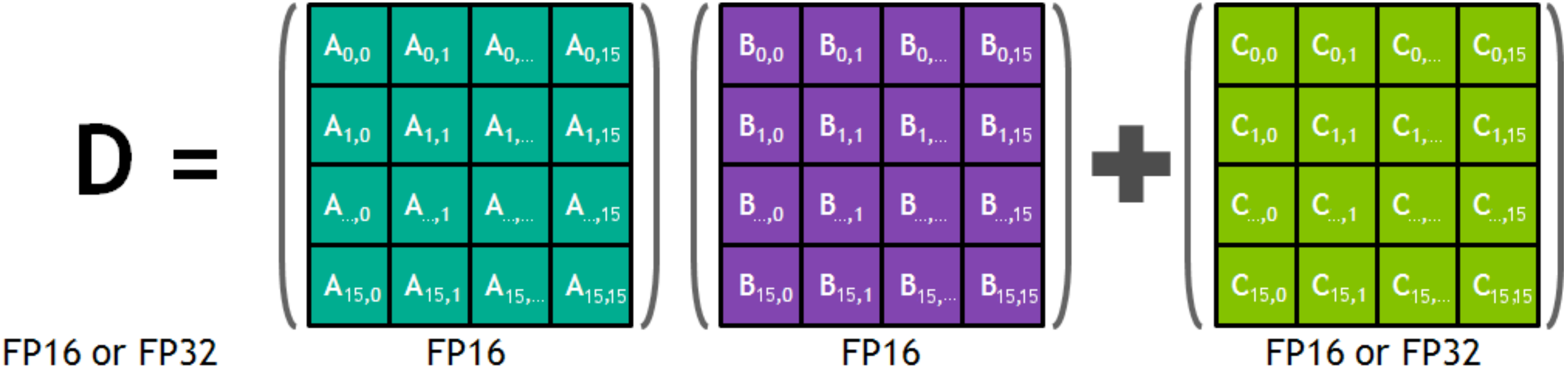
Feature	Titan	Summit
Application Performance	Baseline	5-10x Titan
Number of Nodes	18,688	~4,600
Node performance	1.4 TF	> 40 TF
Memory per Node	38GB DDR3 + 6GB GDDR5	512 GB DDR4 + HBM
NV memory per Node	0	800 GB
Total System Memory	710 TB	>6 PB DDR4 + HBM + Non-volatile
System Interconnect (node injection bandwidth)	Gemini (6.4 GB/s)	Dual Rail EDR-IB (23 GB/s) Or Dual Rail HDR-IB (48 GB/s)
Interconnect Topology	3D Torus	Non-blocking Fat Tree
Processors	1 AMD Opteron™ 1 NVIDIA Kepler™	2 IBM POWER9™ 6 NVIDIA Volta™
File System	32 PB, 1 TB/s, Lustre®	250 PB, 2.5 TB/s, GPFS™
Peak power consumption	9 MW	13 MW

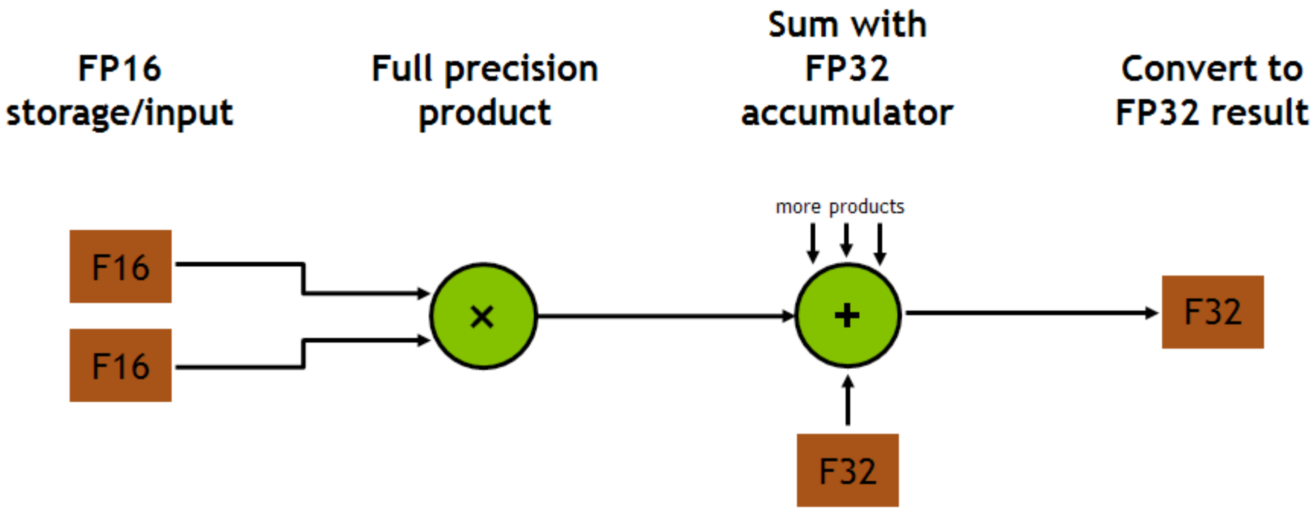
https://www.ornl.gov/news/ornl-launches-summit-supercomputer

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

Mixed-precision LA

Each TC performs **64 floating point FMA mixed-precision** operations **per clock**





Tensor Core Accelerated IRS solving linear system Ax = b

solving linear system Ax = b
LU factorization

- LU factorization is used to solve a linear system Ax=b

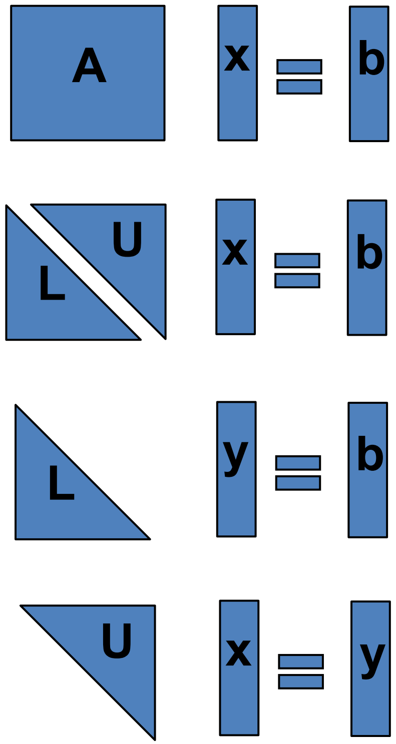
$A\ x = b$

$LUx = b$

$Ly\quad = b$

then

$Ux\quad = y$

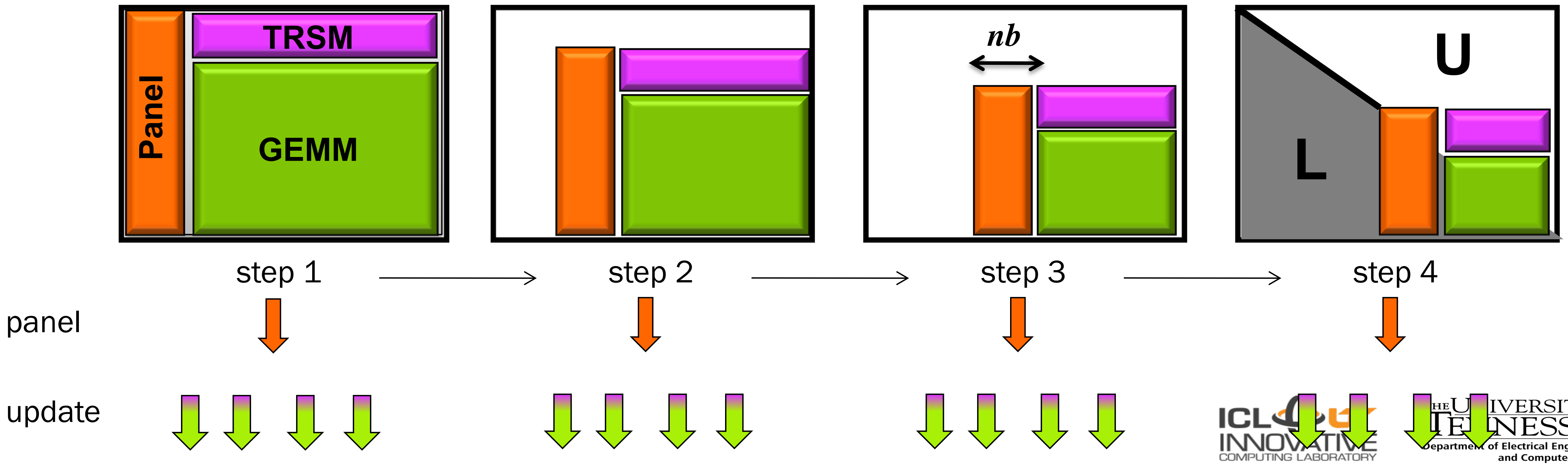


Tensor Core Accelerated IRS solving linear system Ax = b

For s = 0, nb, .. N

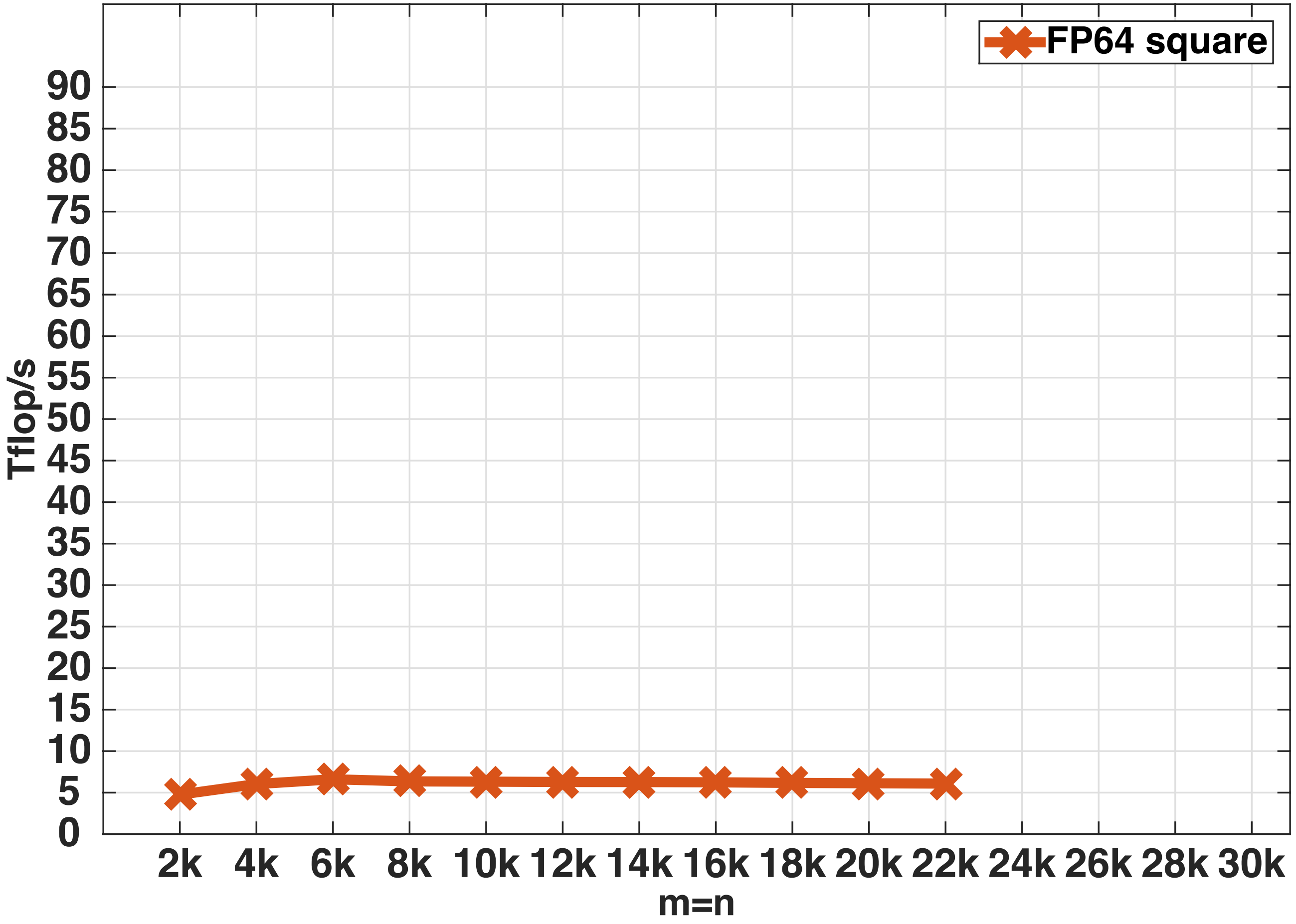
1. panel factorize
2. update trailing matrix

LU factorization requires $O(n^3)$
most of the operations are spent in GEMM



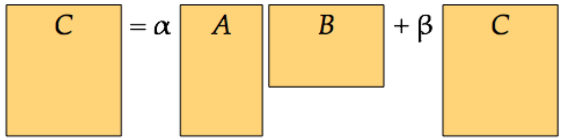
Tensor Core Accelerated IRS Motivation

Study of the Matrix Matrix multiplication kernel on Nvidia V100



- dgemm achieve about 6.4 Tflop/s

Matrix matrix multiplication GEMM

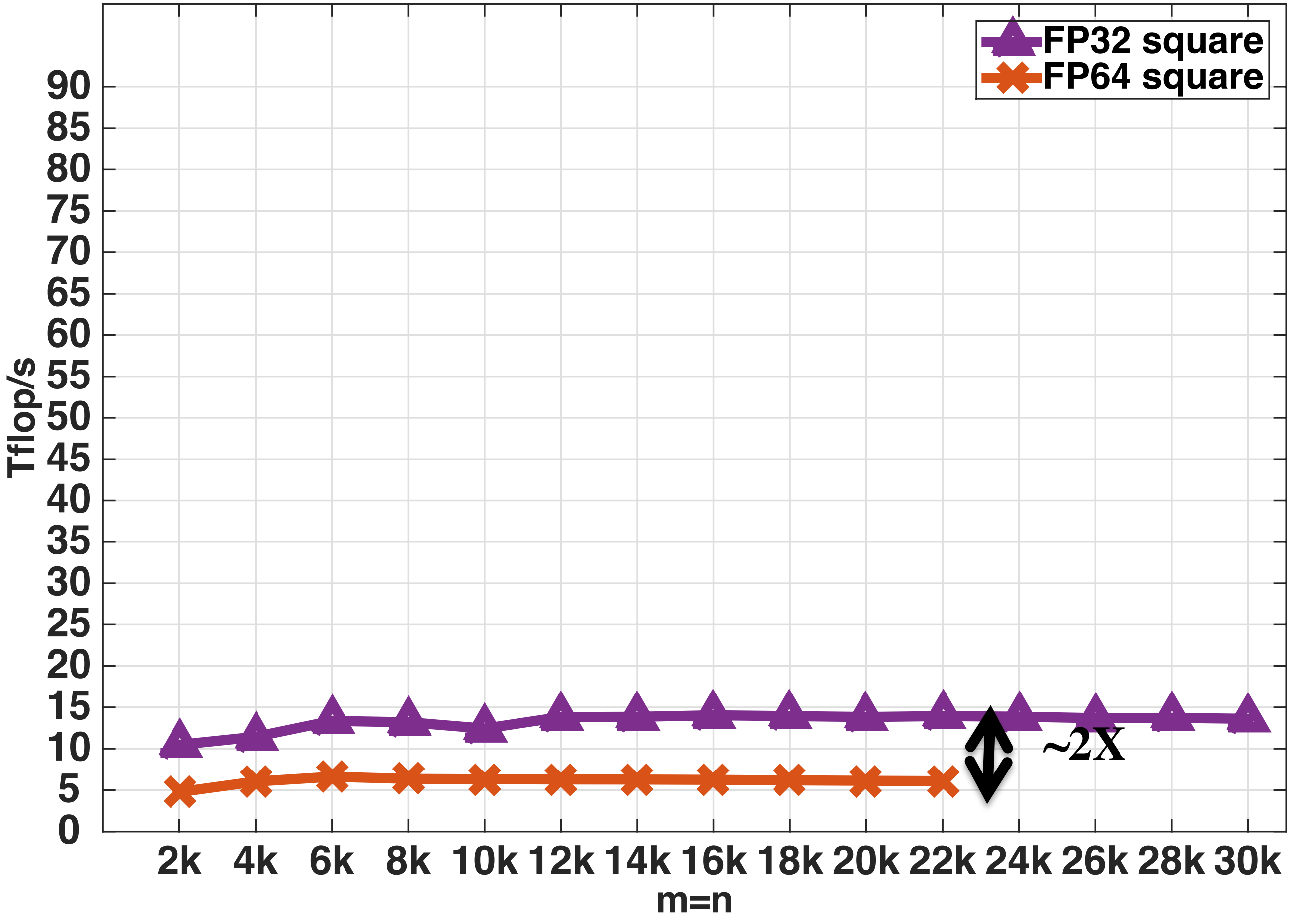


ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

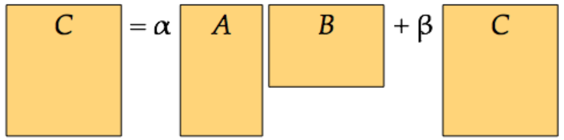
Tensor Core Accelerated IRS Motivation

Study of the Matrix Matrix multiplication kernel on Nvidia V100



- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s

Matrix matrix multiplication GEMM

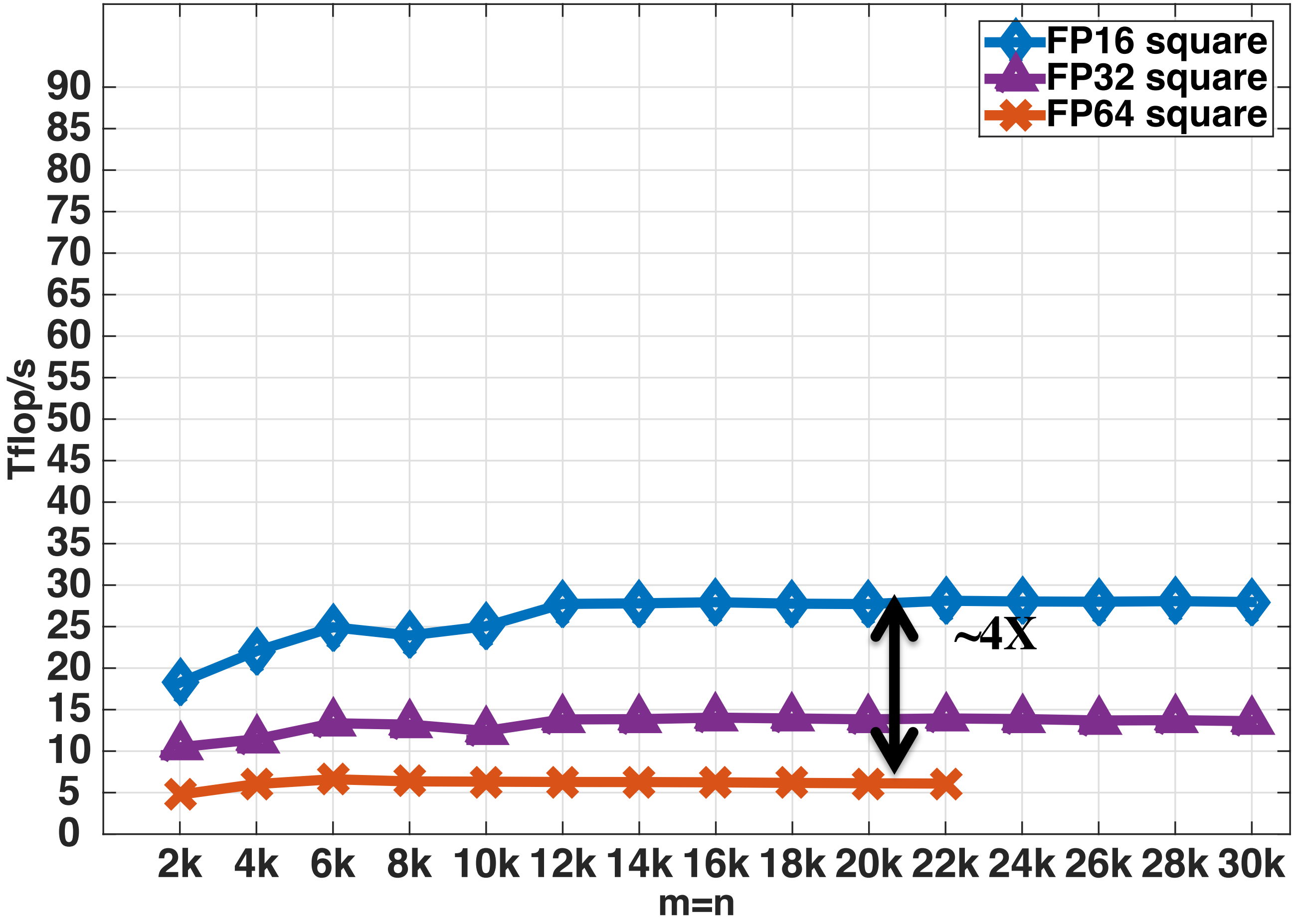


ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

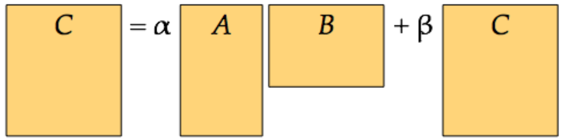
Tensor Core Accelerated IRS Motivation

Study of the Matrix Matrix multiplication kernel on Nvidia V100



- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
- hgemm achieve about 27 Tflop/s

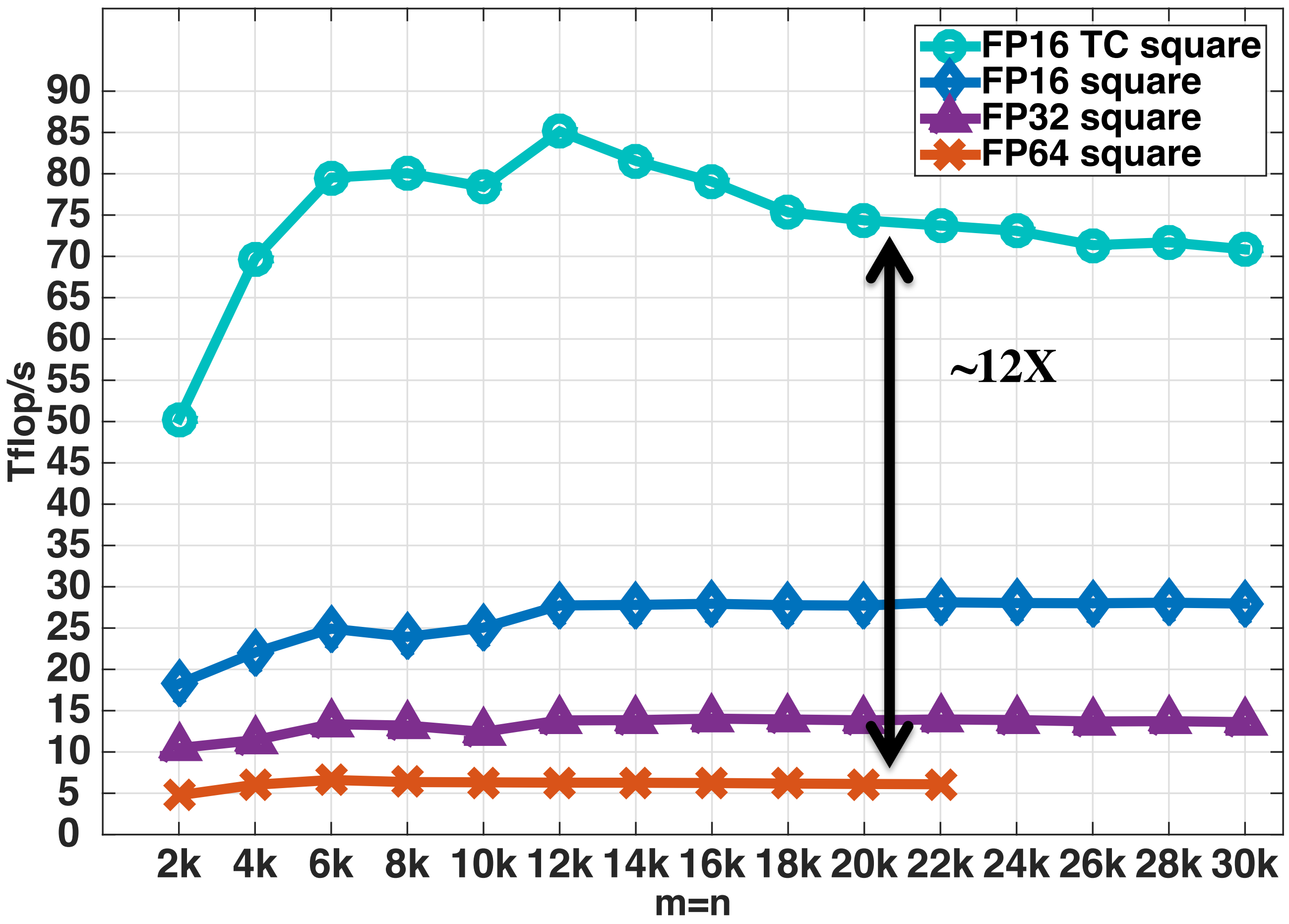
Matrix matrix multiplication GEMM



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

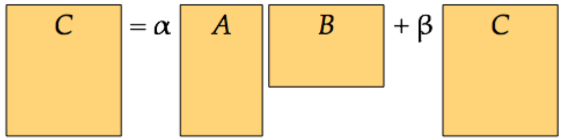
Tensor Core Accelerated IRS Motivation

Study of the Matrix Matrix multiplication kernel on Nvidia V100



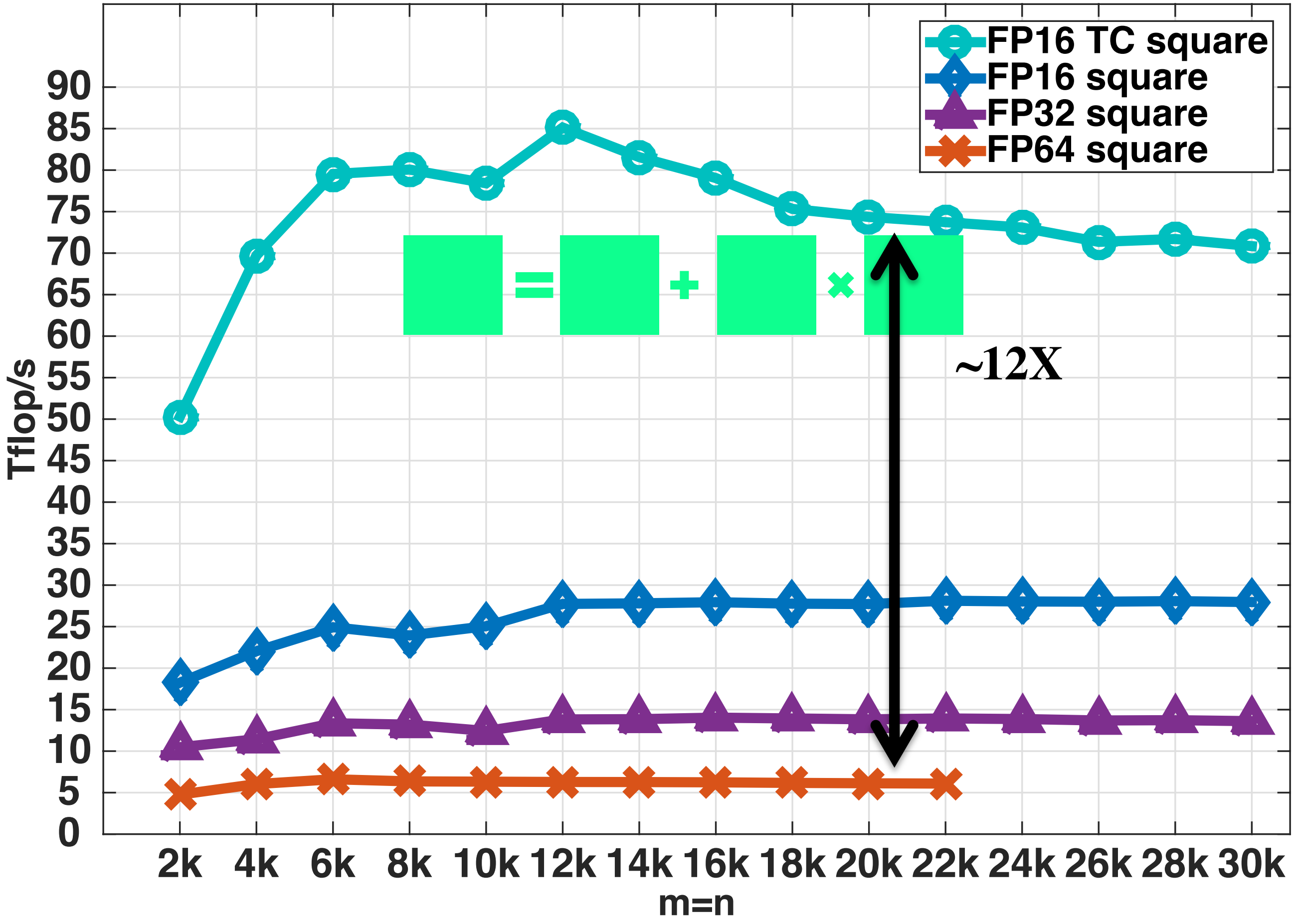
- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
- hgemm achieve about 27 Tflop/s
- Tensor cores gemm reach about 85 Tflop/s

Matrix matrix multiplication GEMM



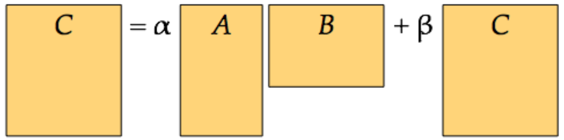
Tensor Core Accelerated IRS Motivation

Study of the Matrix Matrix multiplication kernel on Nvidia V100



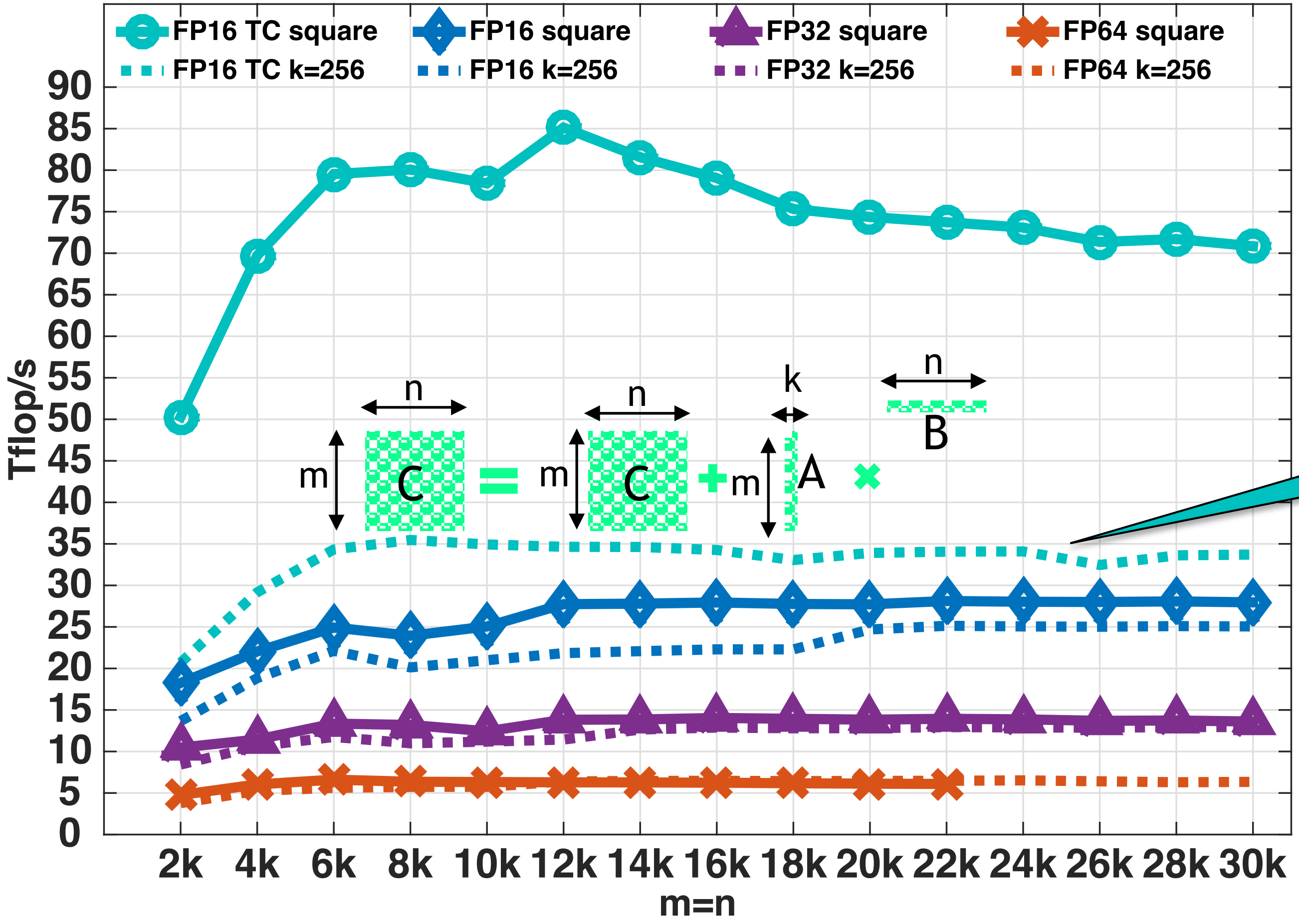
- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
- hgemm achieve about 27 Tflop/s
- Tensor cores gemm reach about 85 Tflop/s

Matrix matrix multiplication GEMM



Tensor Core Accelerated IRS Motivation

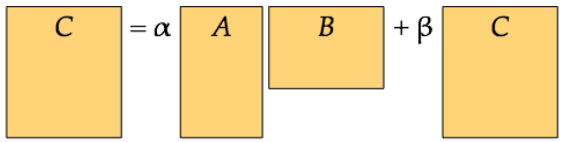
Study of the Matrix Matrix multiplication kernel on Nvidia V100



- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
- hgemm achieve about 27 Tflop/s
- Tensor cores gemm reach about 85 Tflop/s

Rank-k GEMM needed by LU does not perform as well as square but still OK

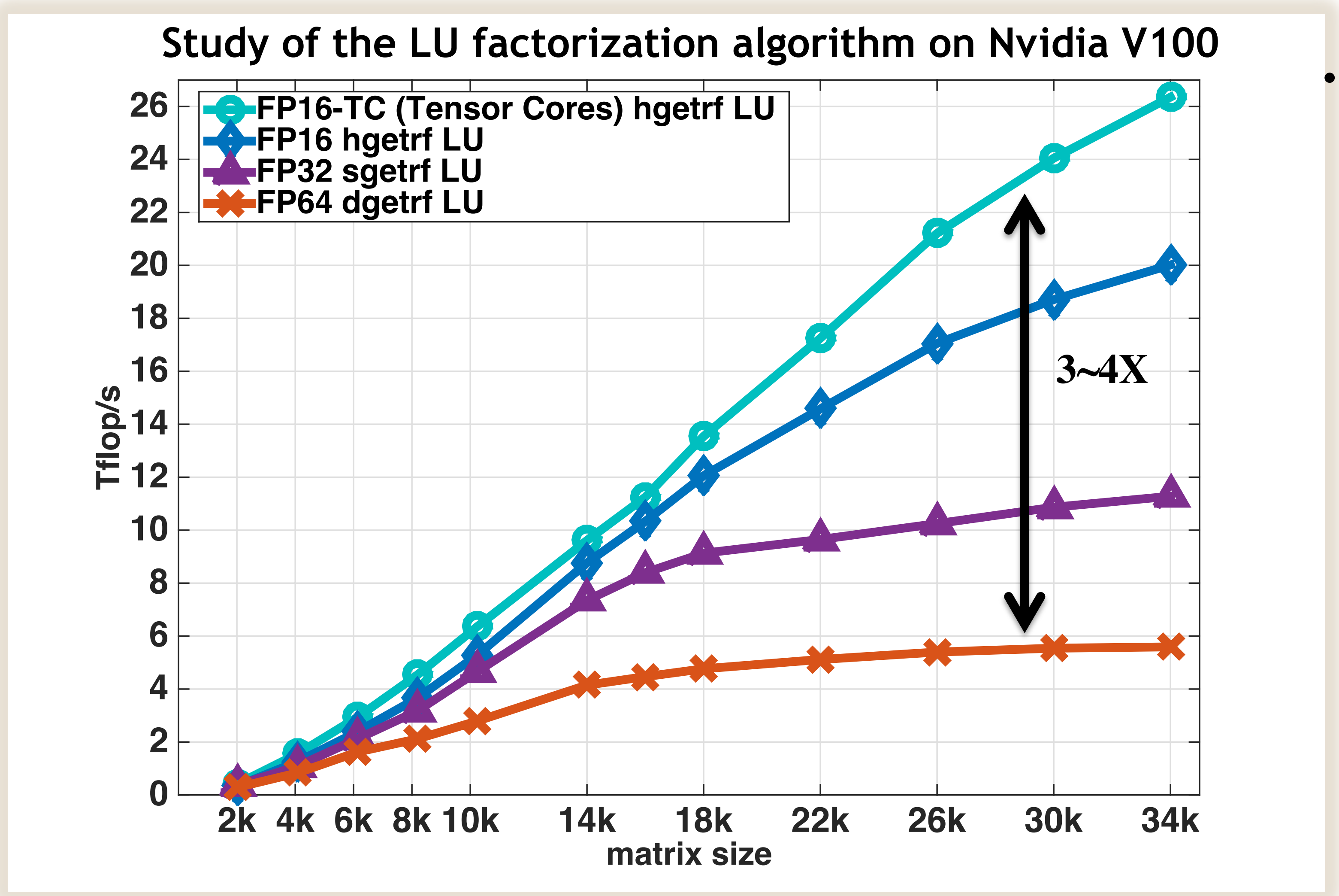
Matrix matrix multiplication GEMM



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Leveraging Half Precision in HPC on V100
Motivation



- LU factorization is used to solve a linear system Ax=b

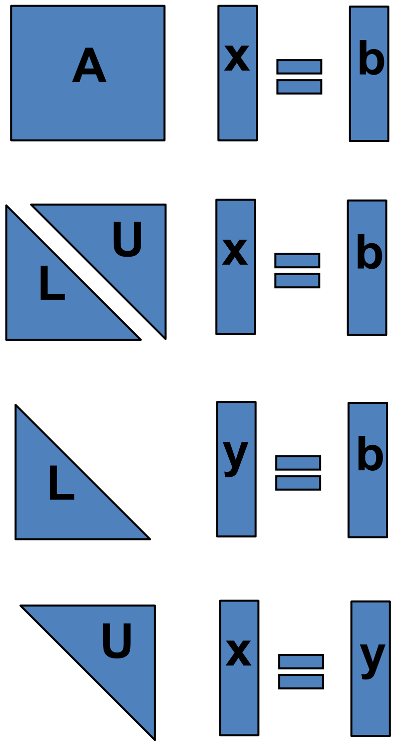
A x = b

LUx = b

Ly = b

then

Ux = y



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Tensor Core Accelerated IRS solving linear system Ax = b

Use Mixed Precision algorithms

Idea: use lower precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine the solution in order to achieve the FP64 arithmetic

- Achieve higher performance → faster time to solution
- Reduce power consumption by decreasing the execution time → Energy Savings !!!

Tensor Core Accelerated IRS solving linear system Ax = b

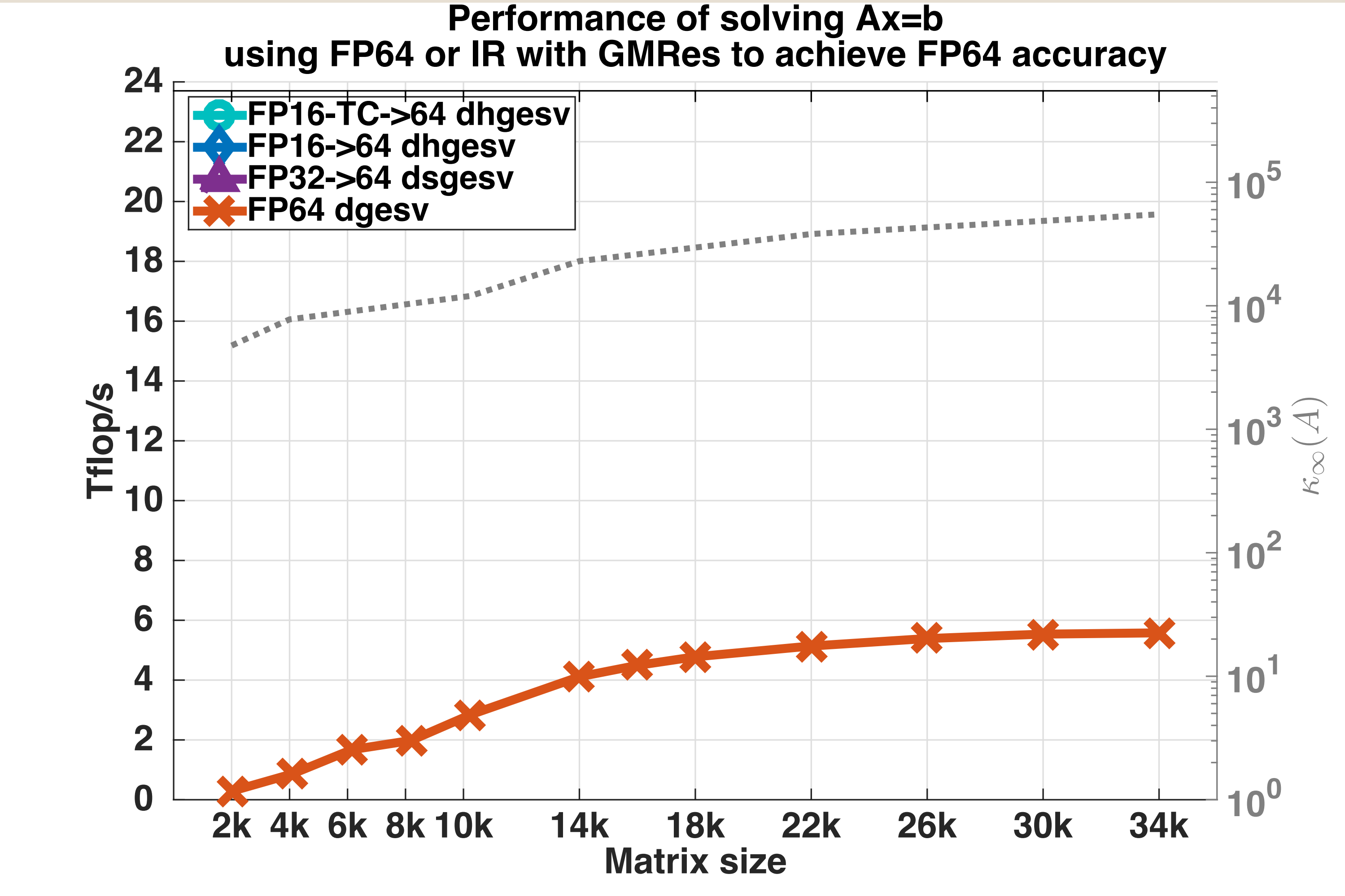
Idea: use lower precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine the solution in order to achieve the FP64 arithmetic

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

L U = lu(A)		lower precision	$O(n^3)$
x = U\(L\b)		lower precision	$O(n^2)$
r = b – Ax		FP64 precision	$O(n^2)$
WHILE || r || not small enough			
1. find a correction "z" to adjust x that satisfy Az=r solving Az=r could be done by either:			
➢ z = U\(L\r)	Classical Iterative Refinement	lower precision	$O(n^2)$
➢ GMRes preconditioned by the LU to solve Az=r	Iterative Refinement using GMRes	lower precision	$O(n^2)$
2. x = x + z		FP64 precision	$O(n^1)$
3. r = b – Ax		FP64 precision	$O(n^2)$
END			

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- E. Carson and N. J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior

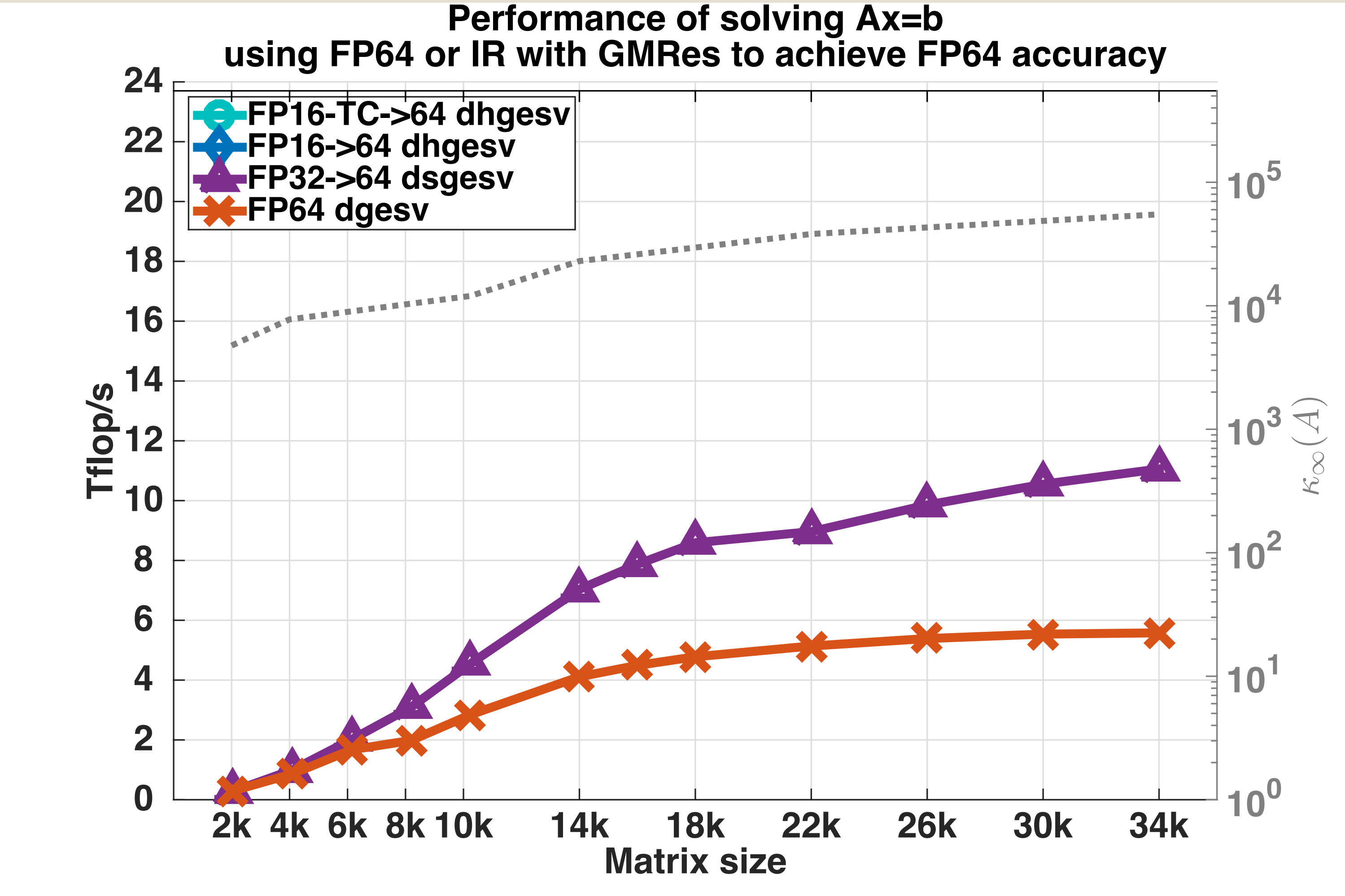


Flops = $2n^3/(3 \text{ time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU

Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior

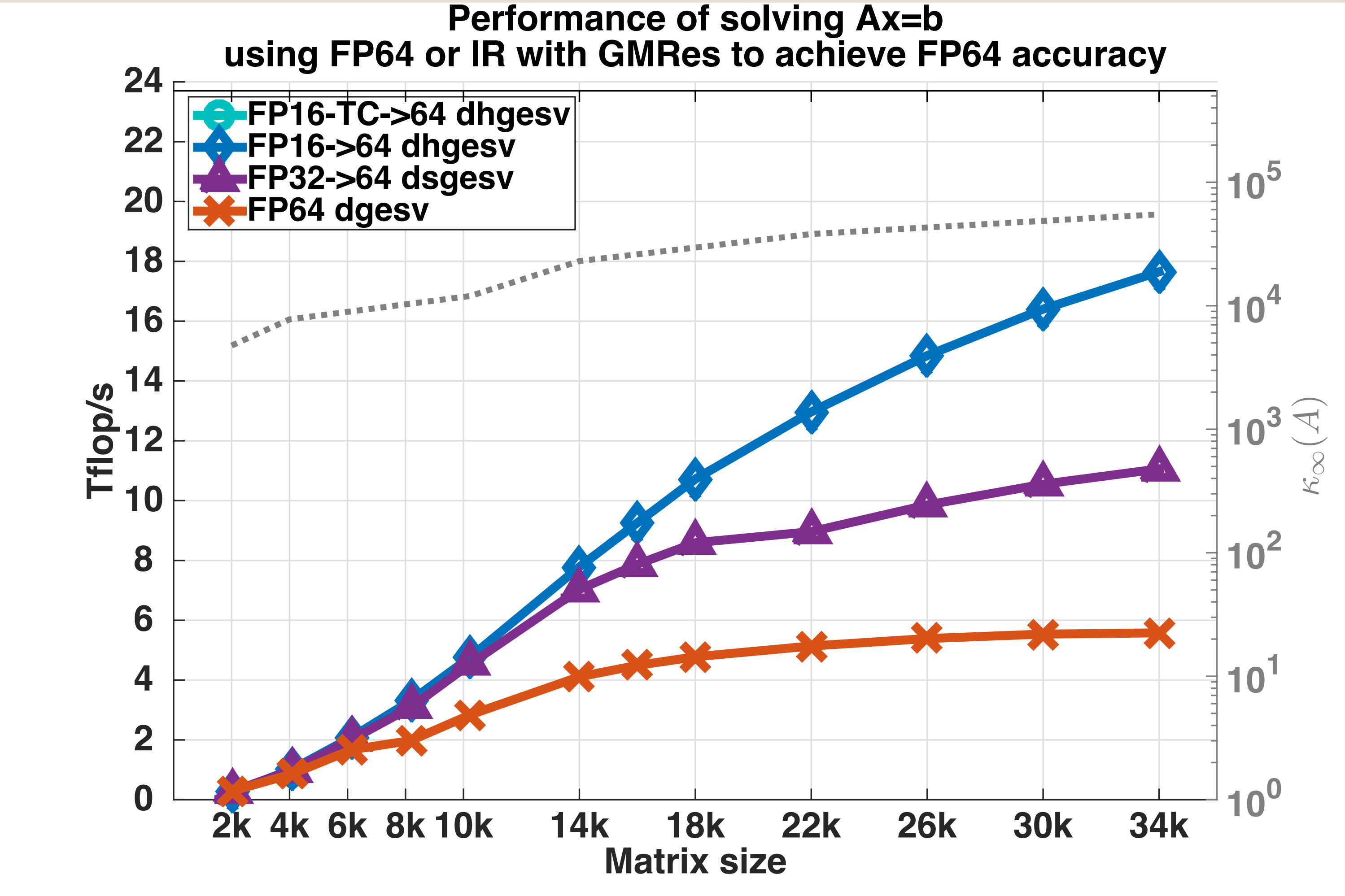


Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy

Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior



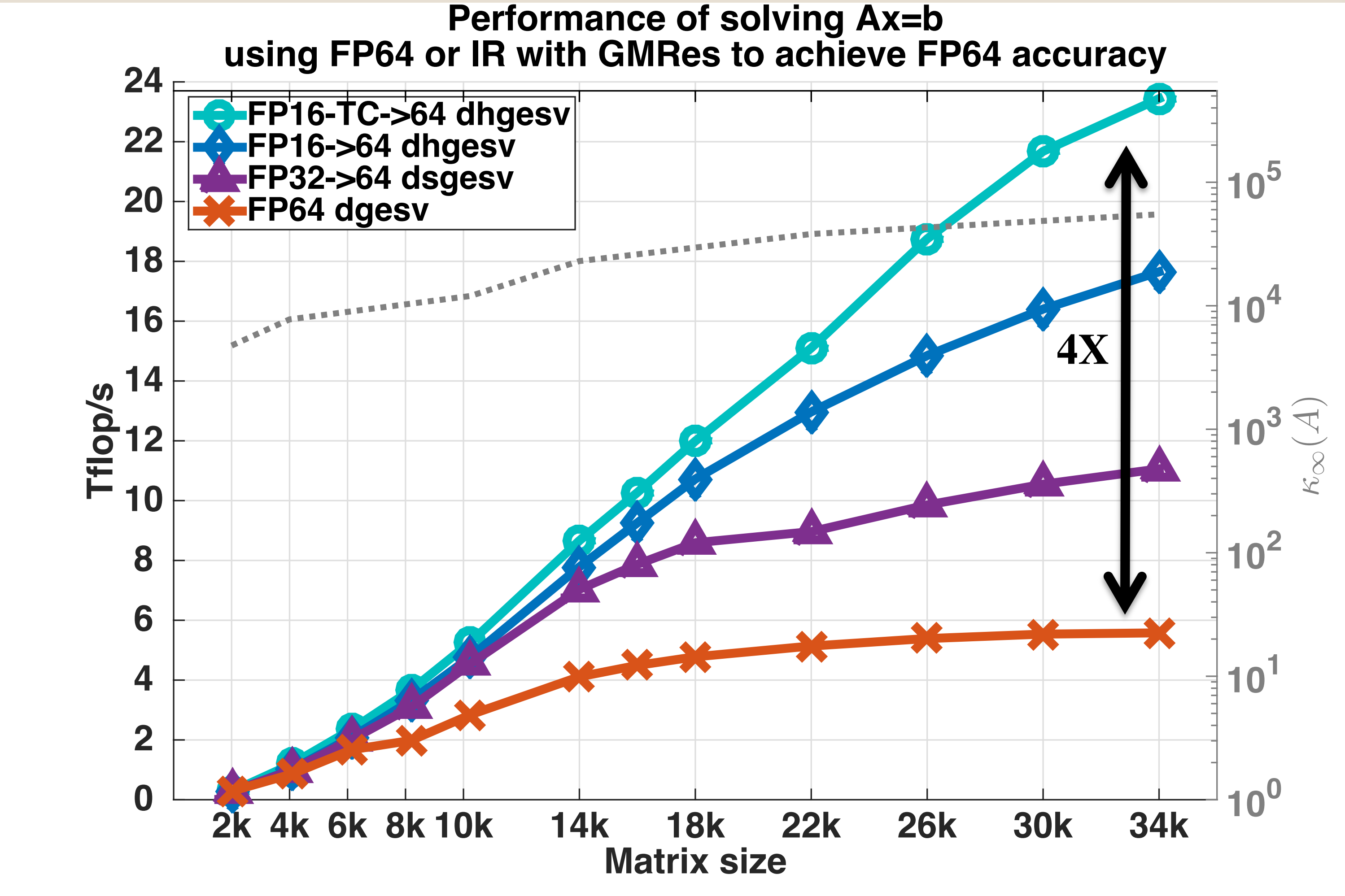
Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 LU and iterative refinement to achieve FP64 accuracy

Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

TENNESSEE
Department of Electrical Engineering and Computer Science

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior



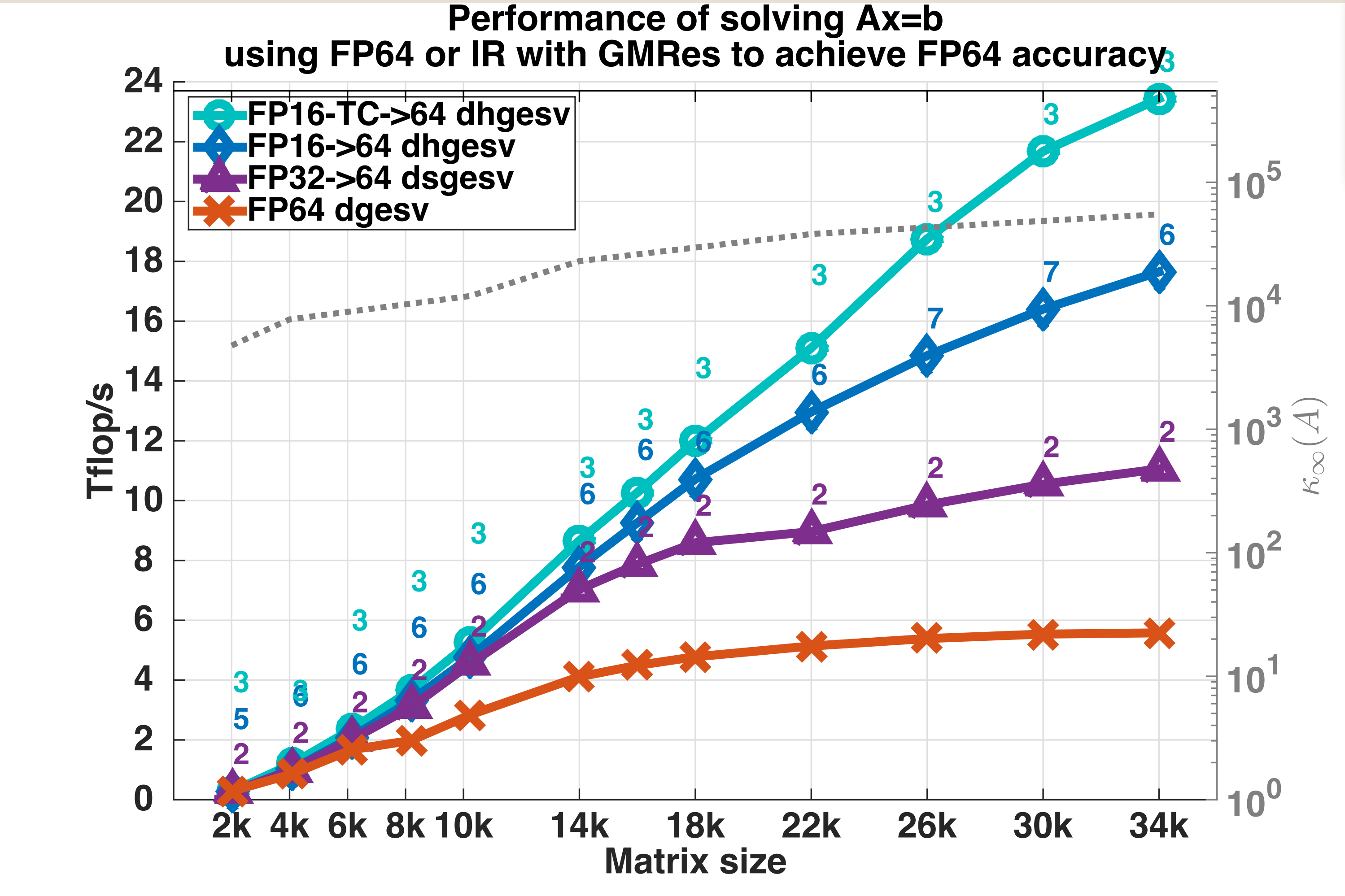
Flops = $2n^3/(3\ time)$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy

Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior



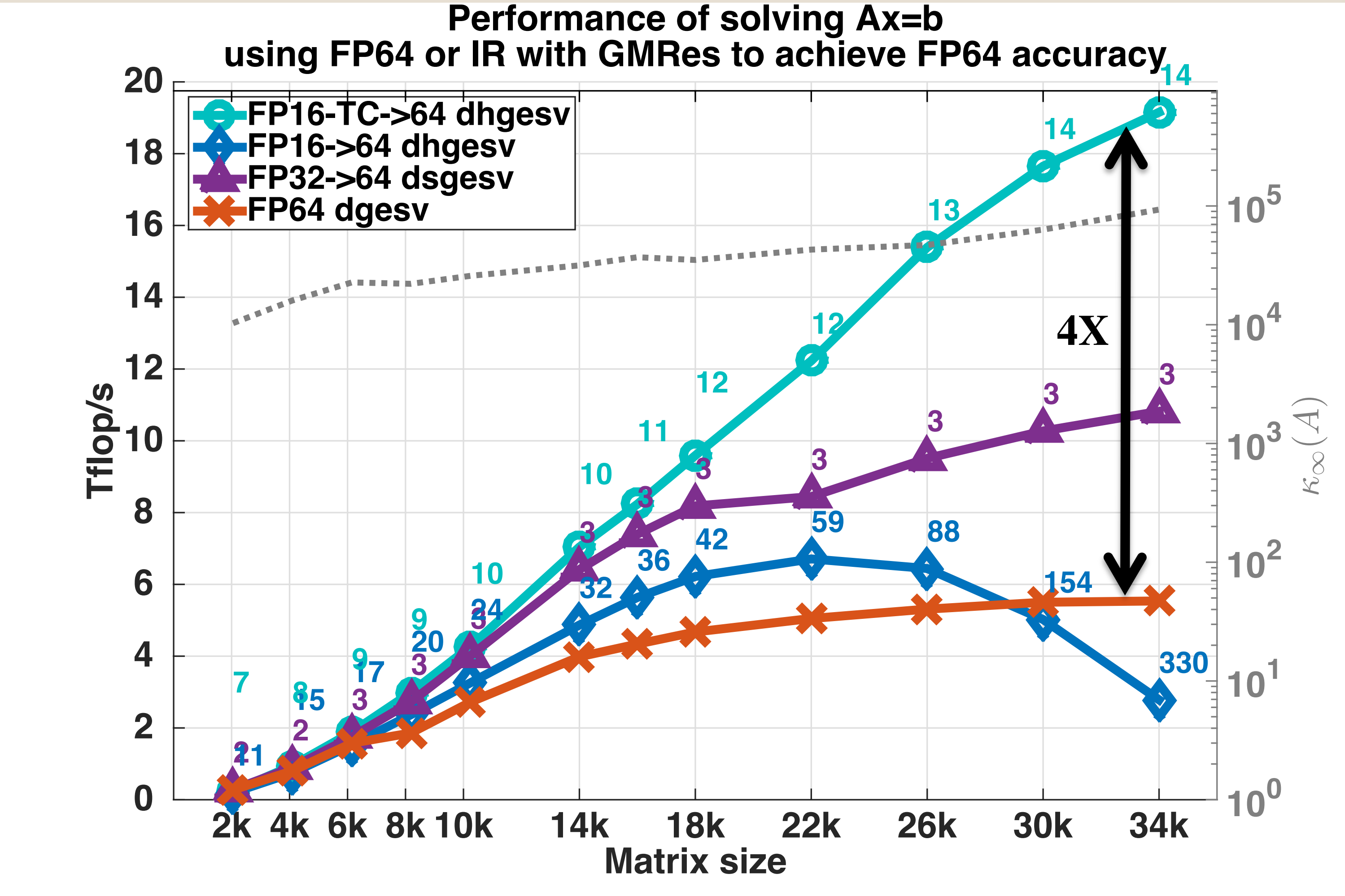
Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy

Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior



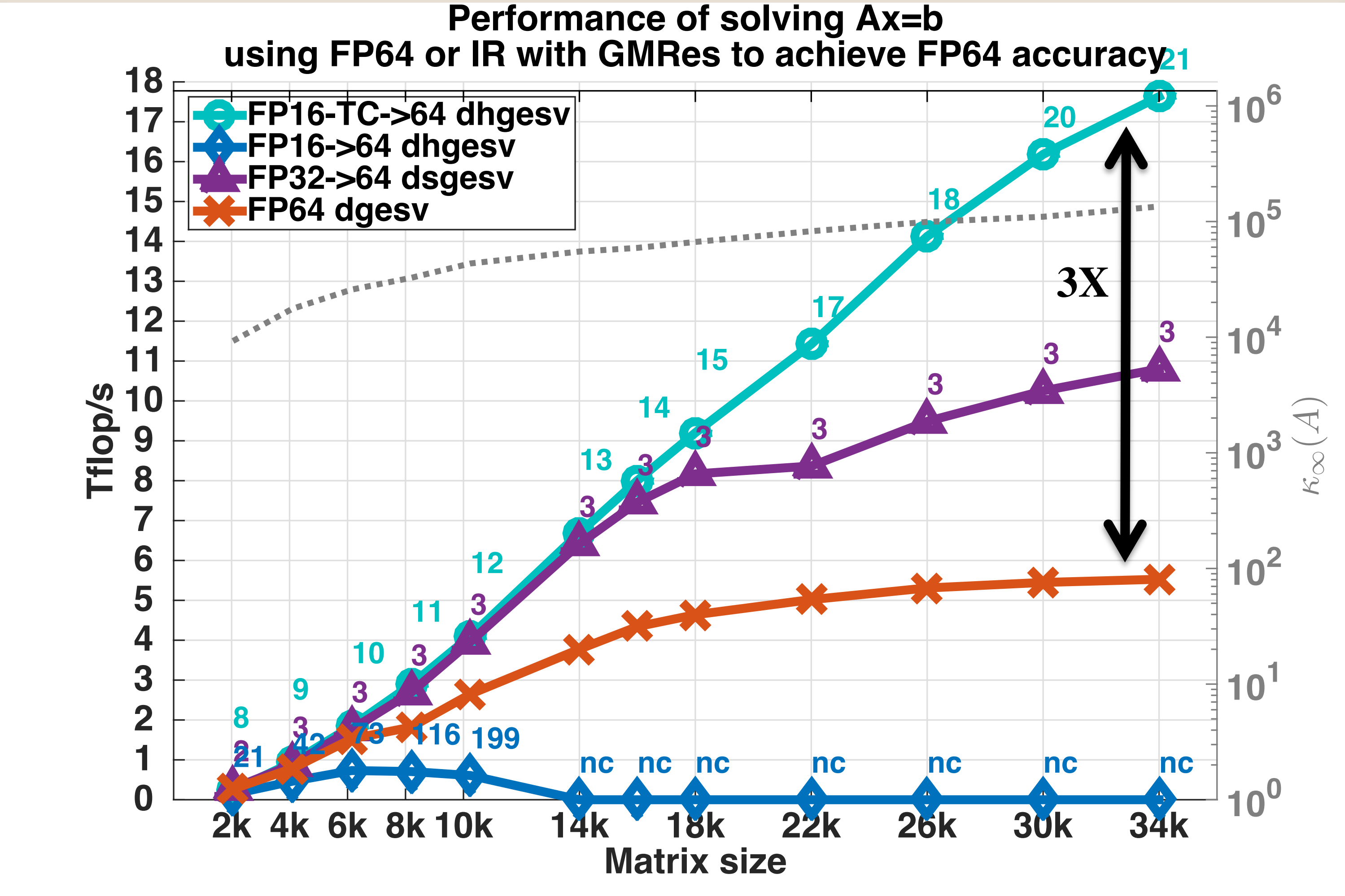
Flops = $2n^3/(3 \text{ time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy

Problem generated with an clustered distribution of the singular values $\sigma = [1, \cdots, 1, \frac{1}{cond}];$

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

Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior



Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU
- solving Ax = b using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 LU and iterative refinement to achieve FP64 accuracy
- solving Ax = b using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy

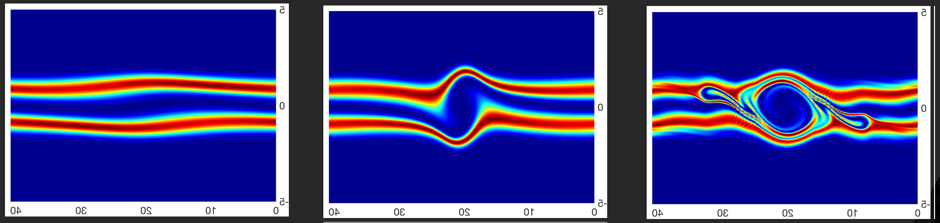
Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$.

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

ADVANCING FUSION DISCOVERIES

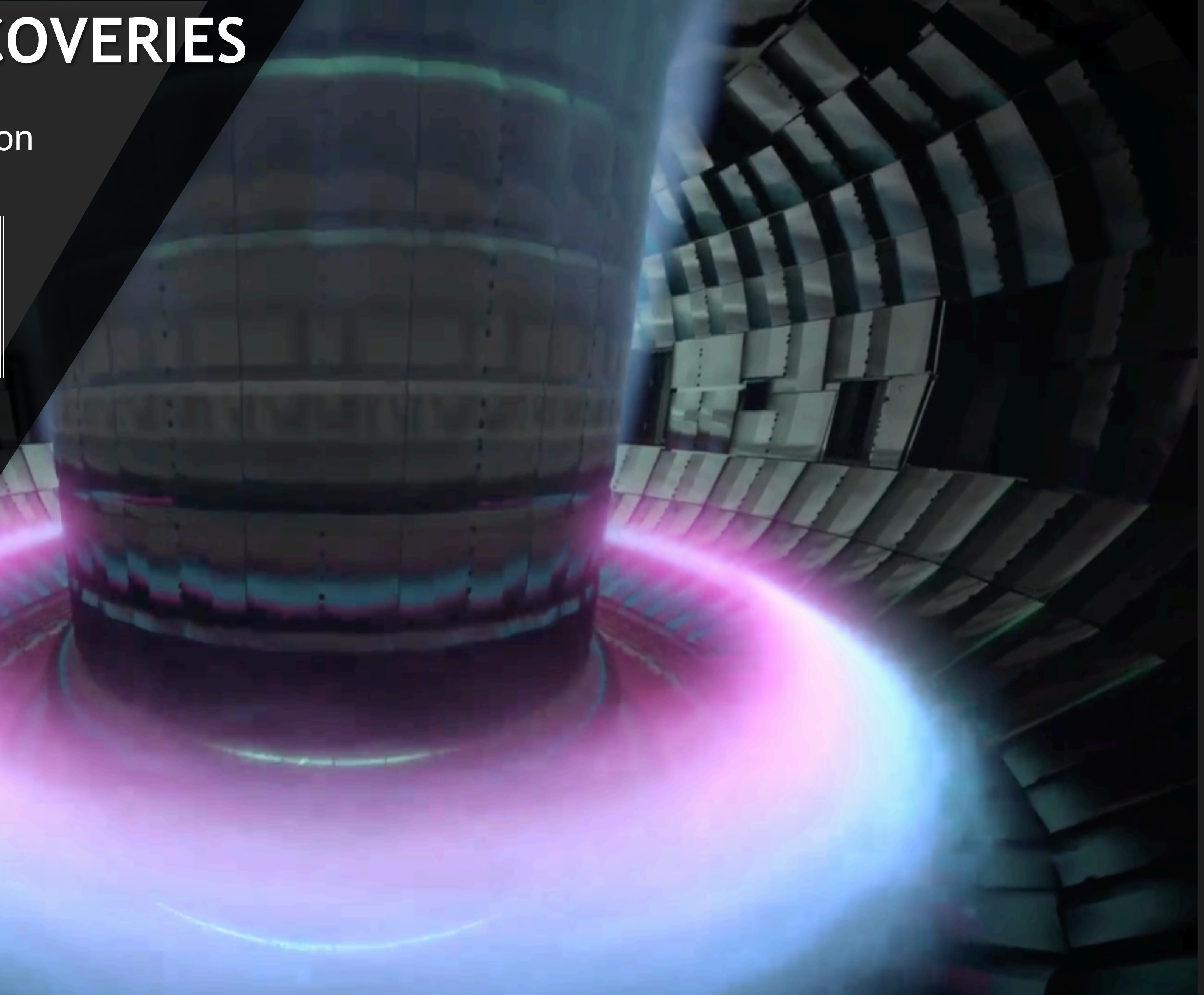
ASGarD: Adaptive Sparse Grid Discretization

Two stream instability study



Scientists believe fusion is the future of energy but maintaining plasma reactions is challenging and disruptions can result in damage to the tokamak. Researchers at ORNL are simulating instabilities in the plasma to provide physicists a better understanding of what happens
inside the reactor.

With NVIDIA Tensor Cores the simulations run 3.5X faster
than previous methods so the team can simulate significantly longer physical times and help advance our understanding of how to sustain the plasma and generate energy

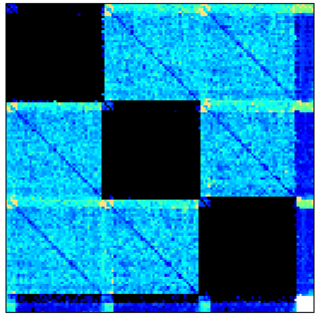


Joint work with NVIDIA, ORNL & UTK: Azzam Haidar, David Green, Ed Azevedo, Wael Elwasif, Graham Lopez, Tyler McDaniel, Lin Mu, Stan Tomov, Jack Dongarra

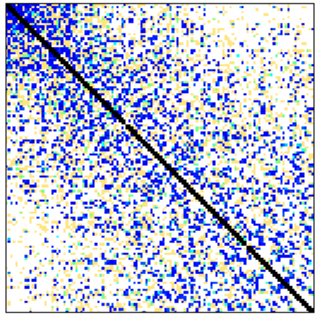
Tensor Core Accelerated IRS solving linear system Ax = b Performance Behavior

PERFORMANCE FOR REAL-LIFE MATRICES FROM THE SUITESPARSE COLLECTION AND FROM DENSE MATRIX ARISING FROM RADAR DESIGN

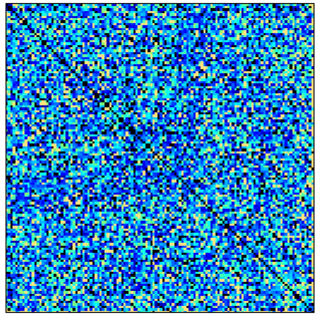
				FP64	FP32 → FP64			FP16 → FP64			FP16-TC → FP64		
name	Description	size	$\kappa_\infty(A)$	dgesv	dsgesv			dhgesv			dhgesv-TC		
				time(s)	# iter	time (s)	speedup	# iter	time (s)	speedup	# iter	time (s)	speedup
em192	radar design	26896	10^6	5.70	3	3.11	**1.8328**	40	5.21	**1.0940**	10	2.05	**2.7805**
appu	NASA app benchmark	14000	10^4	0.43	2	0.27	**1.5926**	7	0.24	**1.7917**	4	0.19	**2.2632**
ns3Da	3D Navier Stokes	20414	$7.6\ 10^3$	1.12	2	0.69	**1.6232**	6	0.54	**2.0741**	4	0.43	**2.6047**
nd6k	ND problem set	18000	$3.5\ 10^2$	0.81	2	0.45	**1.8000**	5	0.36	**2.2500**	3	0.30	**2.7000**
nd12k	ND problem set	36000	$4.3\ 10^2$	5.36	2	2.75	**1.9491**	5	1.86	**2.8817**	3	1.31	**4.0916**
Poisson	2D Poisson problem	32000	$2.1\ 10^6$	3.81	2	2.15	**1.7721**	59	2.04	**1.8676**	10	1.13	**3.3717**
Vlasov	2D Vlasov problem	22000	$8.3\ 10^3$	1.65	2	0.95	**1.7368**	4	0.67	**2.4627**	3	0.48	**3.4375**



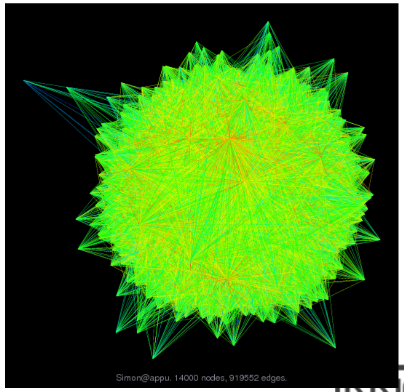
nd12k
1,679,599 nnz

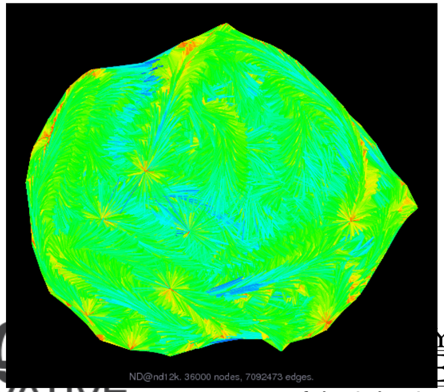


nd12k
14,220,946 nnz



appu
1,853,104 nnz



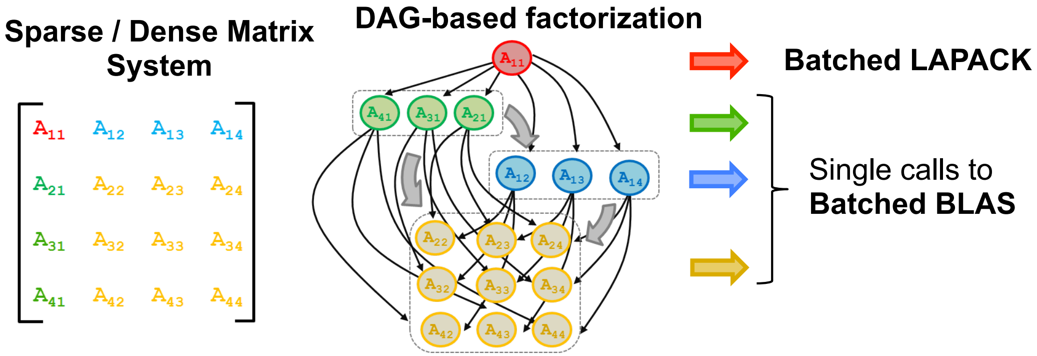


Batched LA:
Many applications need LA on many small matrices

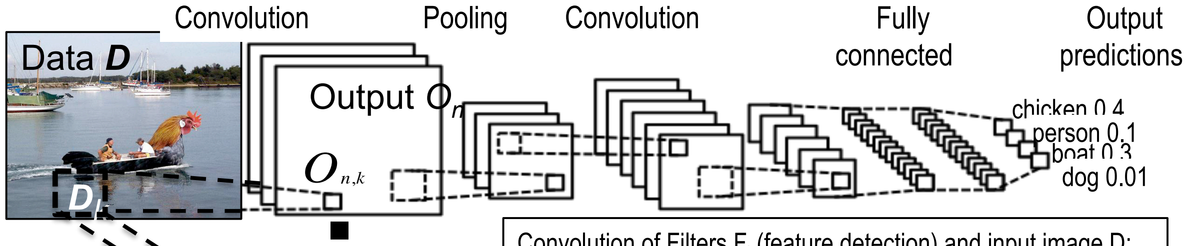
Data Analytics and associated with it Linear Algebra on small LA problems are needed in many applications:

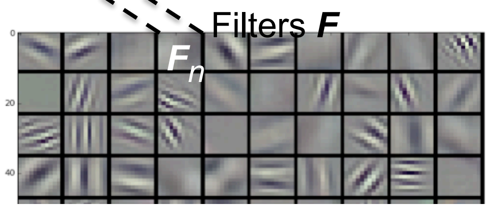
- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, etc.

Sparse/Dense solvers & preconditioners



Machine learning





Convolution of Filters F_i (feature detection) and input image D:

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

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

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

Applications using high-order FEM

- Matrix-free basis evaluation needs efficient tensor contractions,

$$C_{i1,i2,i3} = \sum_k A_{k,i1} B_{k,i2,i3}$$

- **Within ECP CEED Project,** designed MAGMA batched methods to split the computation in many small high-intensity GEMMs, grouped together (batched) for efficient execution:

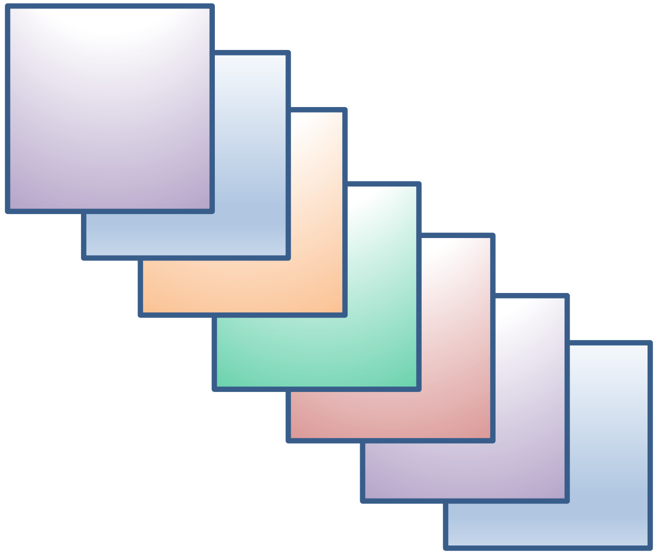
Batch_{ $C_{i3} = A^T B_{i3}$, for range of i3 }

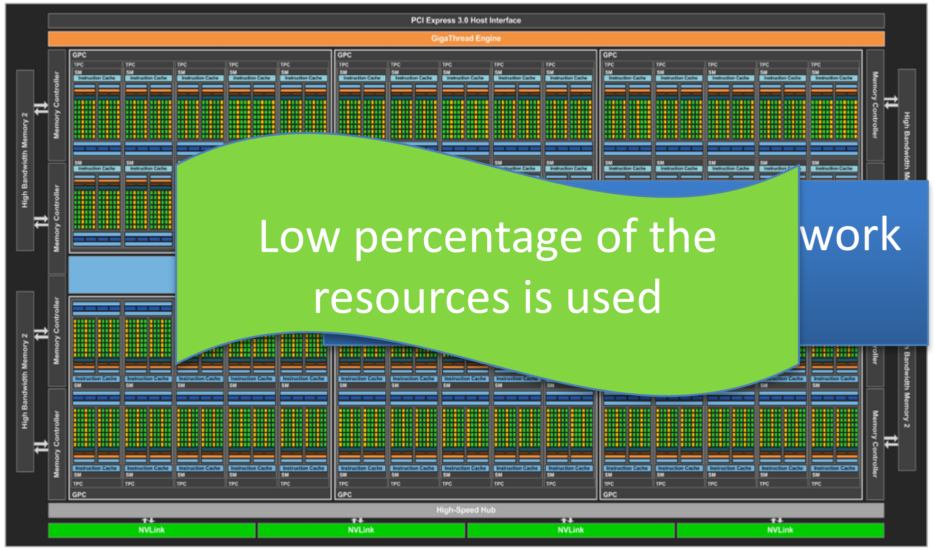
MAGMA Batched Computations

1. Non-batched computation

- **loop over the matrices one by one** and compute using multithread (note that, since matrices are of small sizes there is not enough work for all the cores). So we expect low performance as well as threads contention might also affect the performance

```
for (i=0; i<batchcount; i++)
   dgemm(…)
```





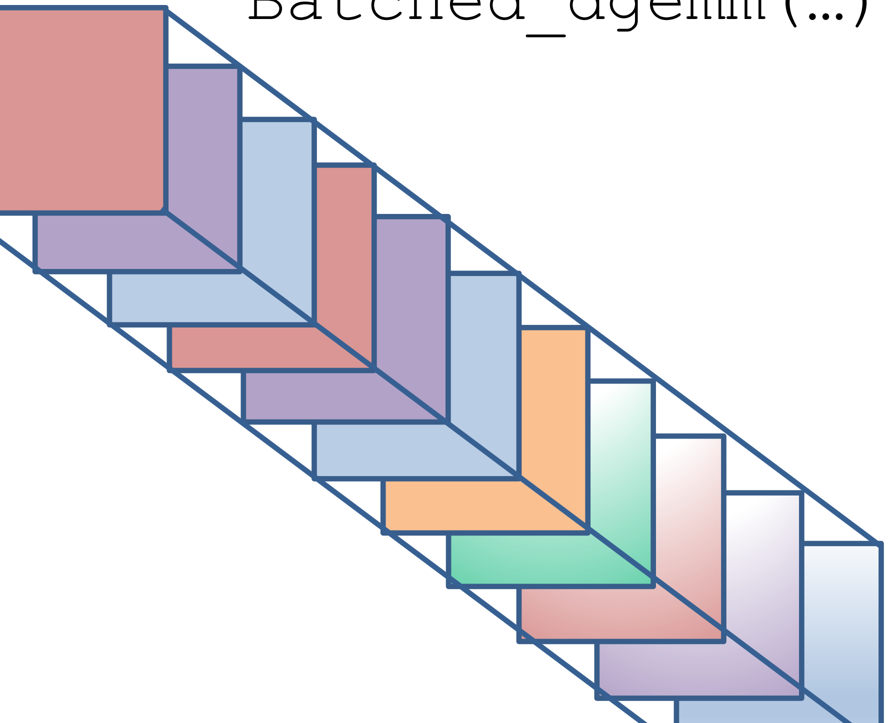
MAGMA Batched Computations

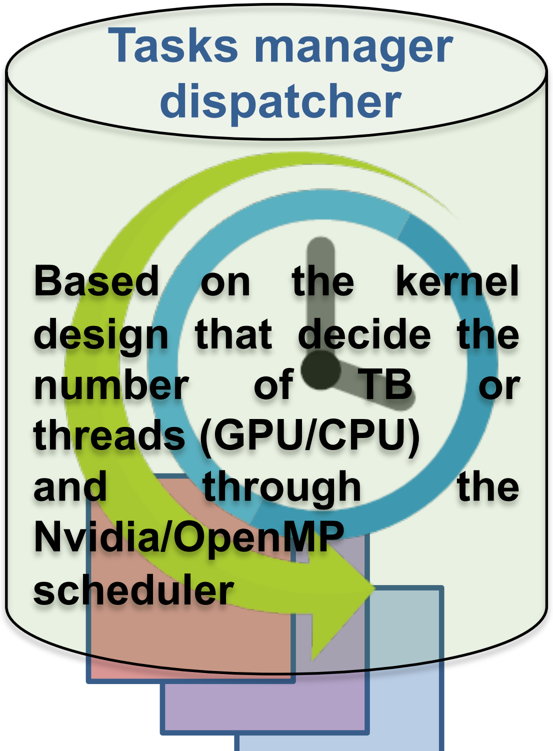
1. Batched computation

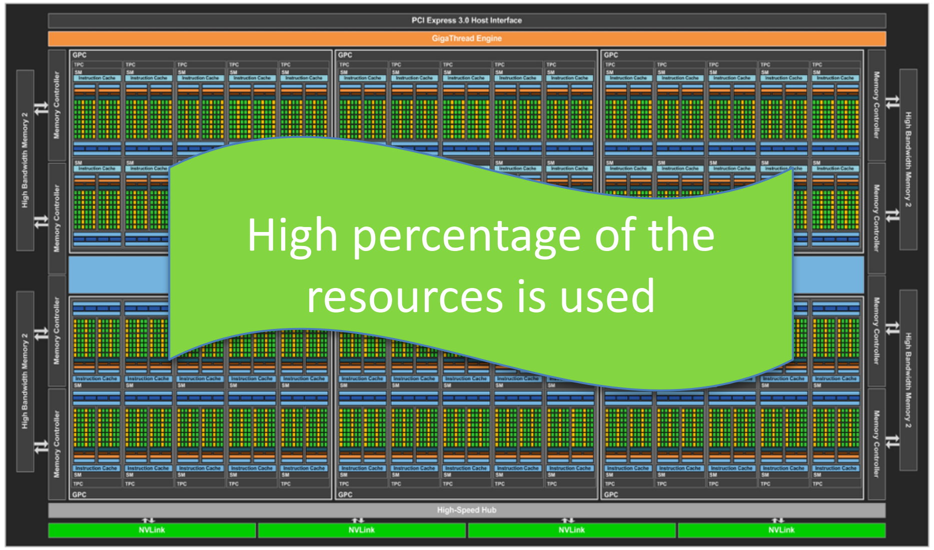
Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently

- For very small matrices, assign a matrix/core (CPU) or per TB for GPU
- For medium size a matrix go to a team of cores (CPU) or many TB's (GPU)
- For large size switch to multithreads classical 1 matrix per round.

`Batched_dgemm(…)`







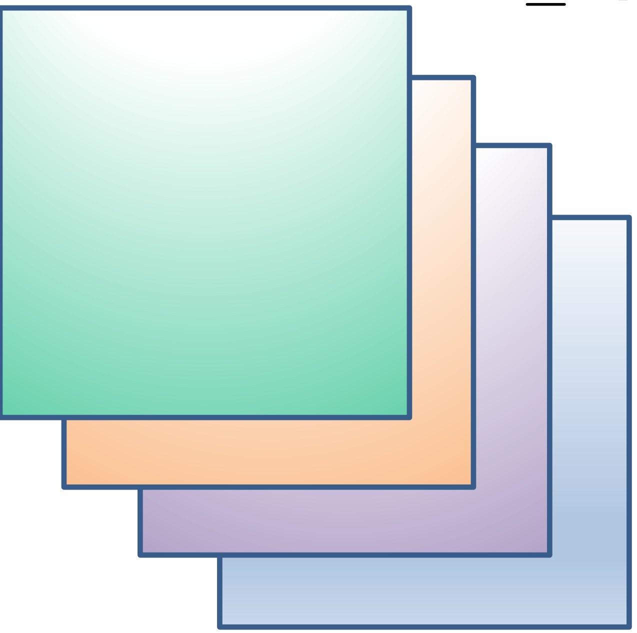
MAGMA Batched Computations

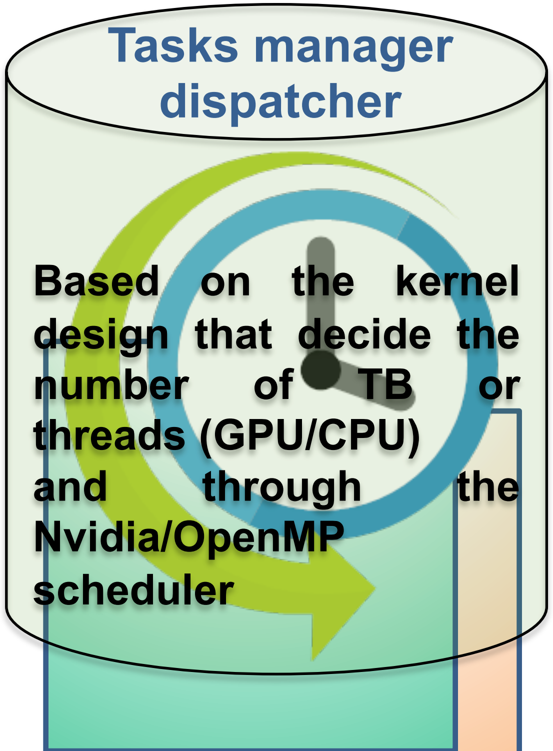
1. Batched computation

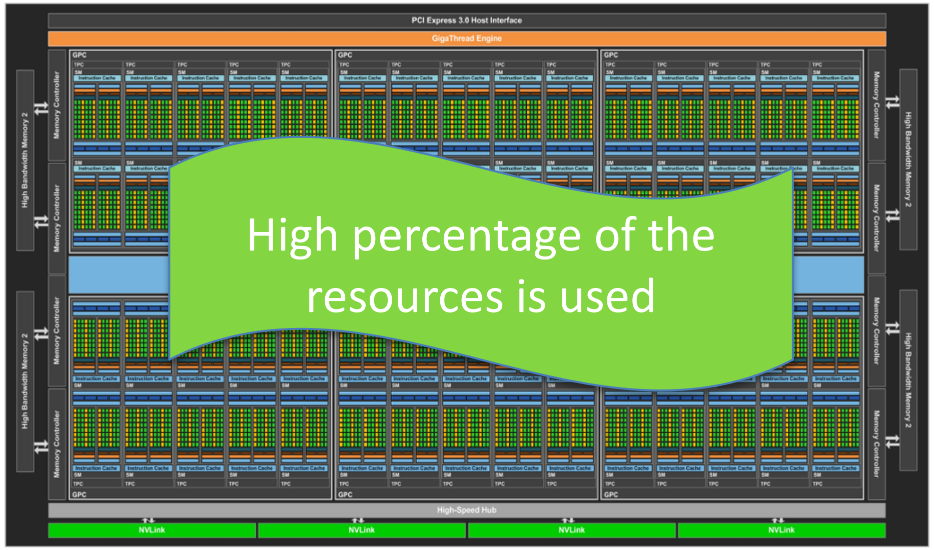
Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently

- For very small matrices, assign a matrix/core (CPU) or per TB for GPU
- For medium size a matrix go to a team of cores (CPU) or many TB's (GPU)
- For large size switch to multithreads classical 1 matrix per round.

```
Batched_dgemm(…)
```







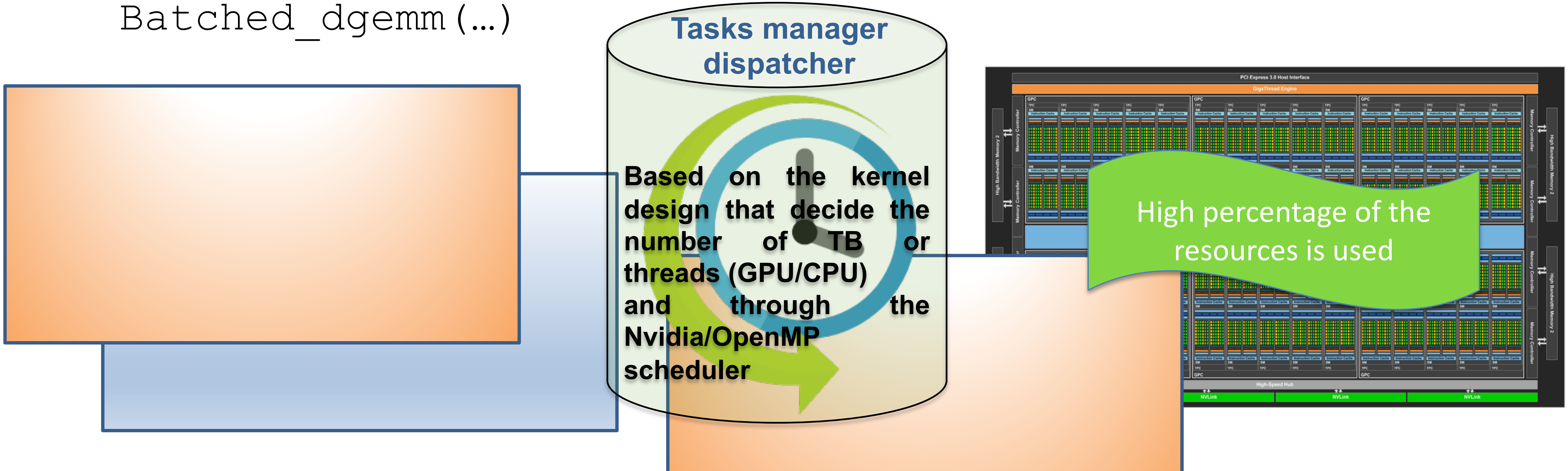
MAGMA Batched Computations

1. Batched computation

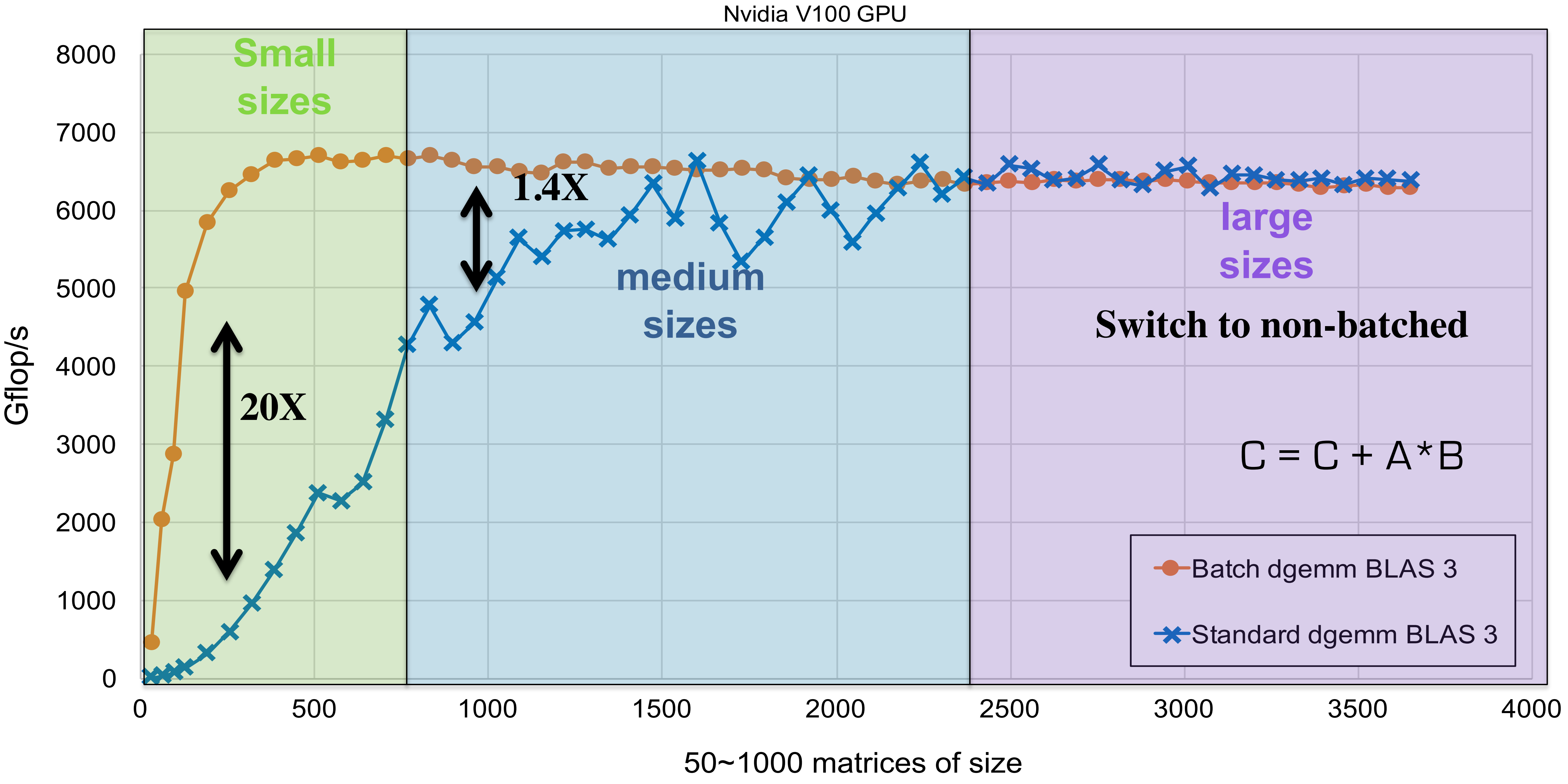
Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently

- For very small matrices, assign a matrix/core (CPU) or per TB for GPU
- For medium size a matrix go to a team of cores (CPU) or many TB's (GPU)
- For large size switch to multithreads classical 1 matrix per round.

`Batched_dgemm(…)`

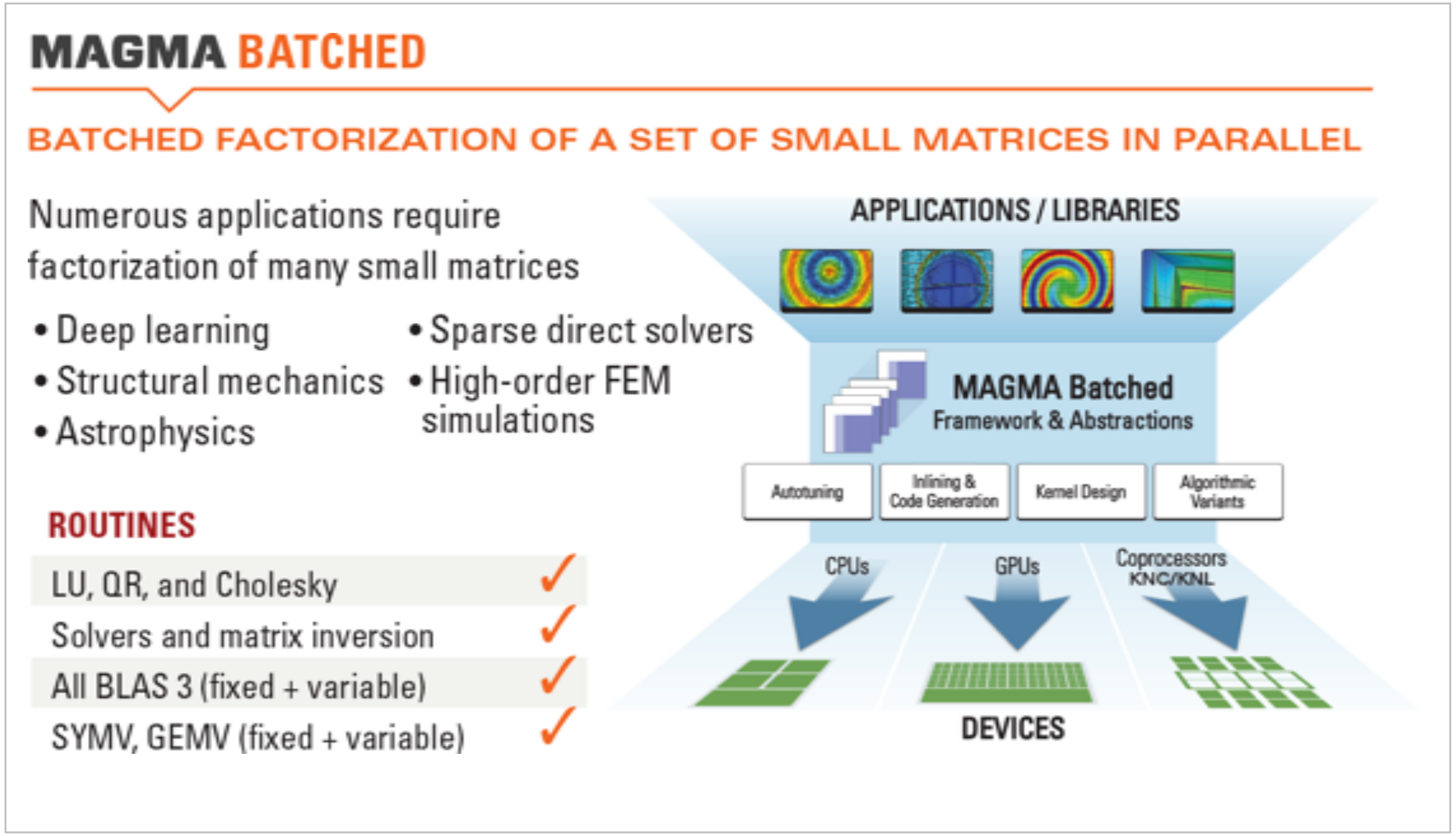


MAGMA Batched Computations



How to implement fast batched DLA?

MAGMA 2.5.1 provides the most extended set of Batched BLAS and LAPACK functionalities to date



http://icl.cs.utk.edu/magma https://bitbucket.org/icl/magma

CEED

ECP Co-Design Project
Center for Efficient Exascale Discretiations (CEED)

MAGMA – main technologies used/developed:

- **Batched BLAS standardization**
- **Batched GEMMs**
- **Tensor contractions through fusing Batched DEMMs**

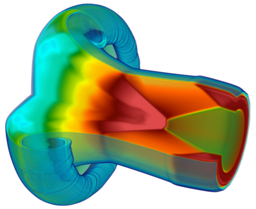
$$\texttt{batch<e=0..nelems>}\{\ B_e^T D_e.*(B_e A_e {B_e}^T) B_e\ \}$$

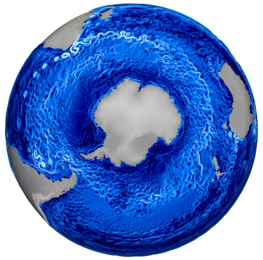
vs.

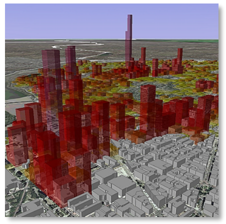
$$\begin{aligned}
&\texttt{batch<e=0..nelems>}\ \{\ C_e = A_e {B_e}^T\ \};\\
&\texttt{batch<e=0..nelems>}\ \{\ C_e = B_e C_e\ \};\\
&\texttt{batch<e=0..nelems>}\ \{\ C_e = D_e.*C_e\ \};\\
&\texttt{batch<e=0..nelems>}\ \{\ C_e = C_e B_e\ \};\\
&\texttt{batch<e=0..nelems>}\ \{\ C_e = B_e^T C_e\ \};
\end{aligned}$$

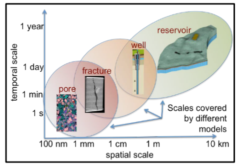
- **Auto-generation of kernels and tuning**
- **MAGMA Templates**
  - **to easily port CPU code to GPUs**

CEED target applications

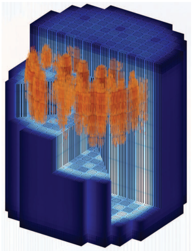

Compressible flow (MARBL)


Climate (E3SM)

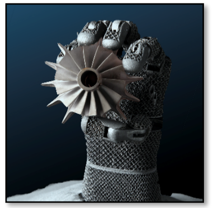

Urban systems (Urban)

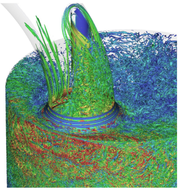


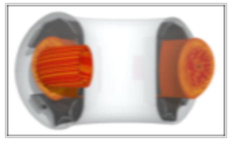
Subsurface (GEOS)


Modular Nuclear Reactors (ExaSMR)

Wind Energy (ExaWind)

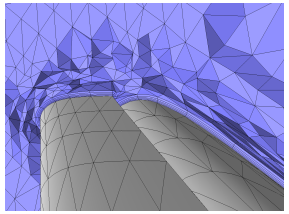

Additive Manufacturing (ExaAM)

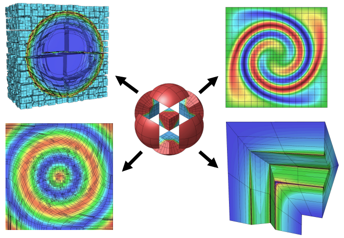

Combustion (Nek5000)

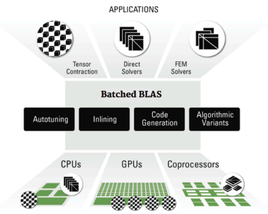
Magnetic Fusion (WDMApp)


We are interested in working with other applications!

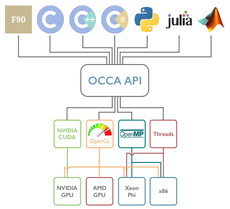
High-Order Software Ecosystem


High-order Meshes


Unstructured AMR



Tensor contractions

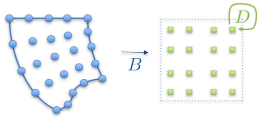


Performance portability

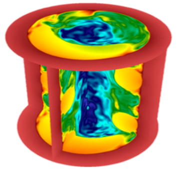
PETSc
hypre
high performance preconditioners

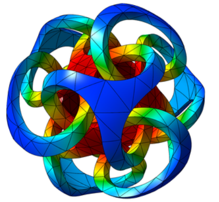
Scalable matrix-free solvers

$A = P^T G^T B^T D B G P$



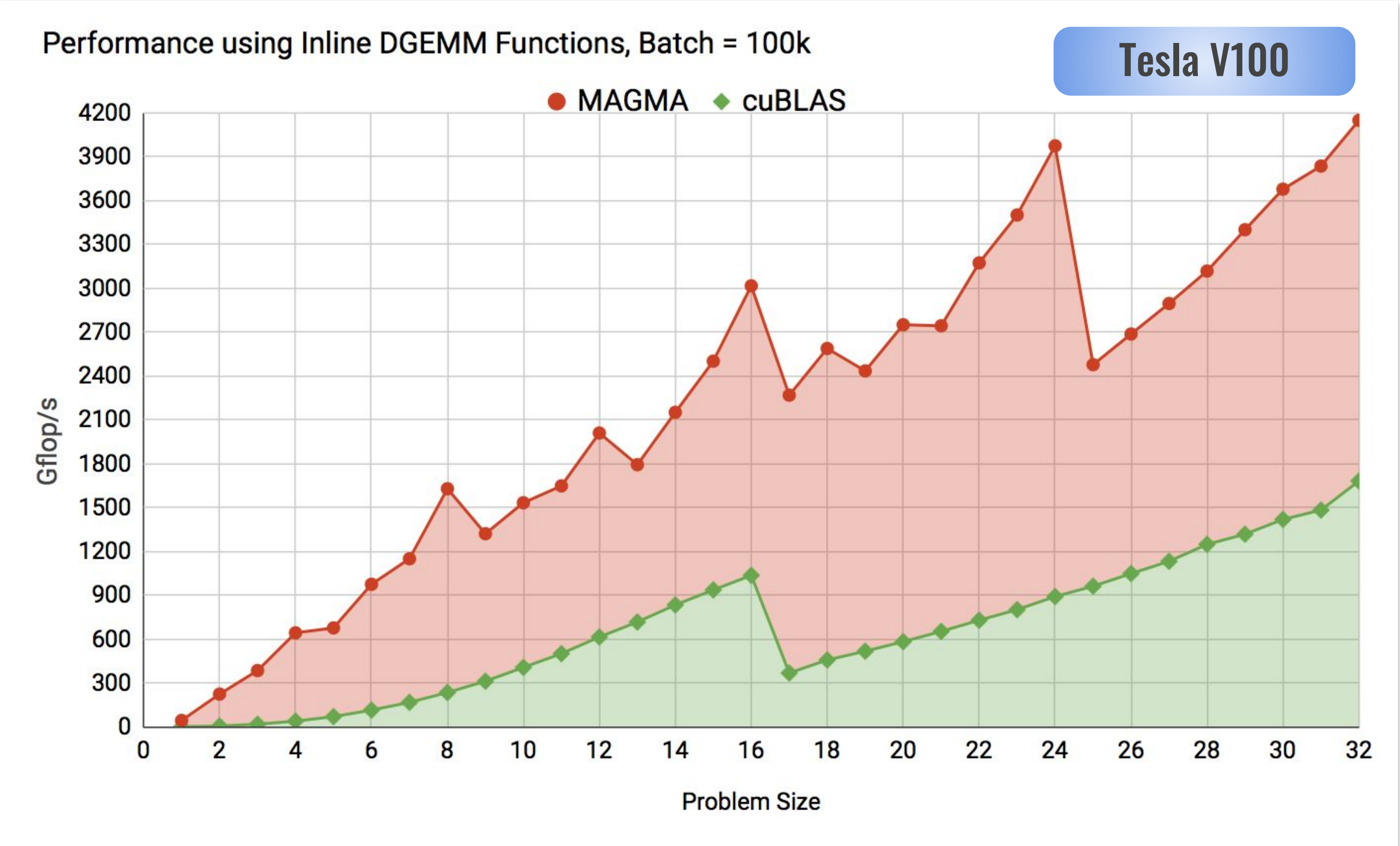
High-Order Operator Format


General Interpolation


High-Order Visualization

Batched computing technology

- Fused kernels



THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

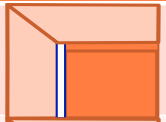
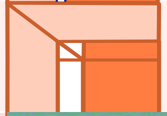
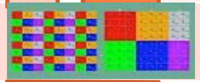
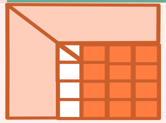
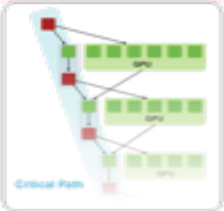
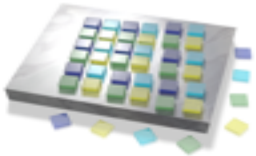
Redesign LAPACK and ScaLAPACK for new architectures: the MAGMA and SLATE libraries

- **Make the most up-to-date algorithms and highly-tuned numerical kernels available as building blocks for production codes on emerging architectures**

- **Develop data abstractions and APIs to ease interoperability and integration, e.g., through familiar Sca/LAPACK interfaces, wherever possible**

- **Implement self-contained novel linear algebra algorithms that can replace the currently used libraries in production codes**

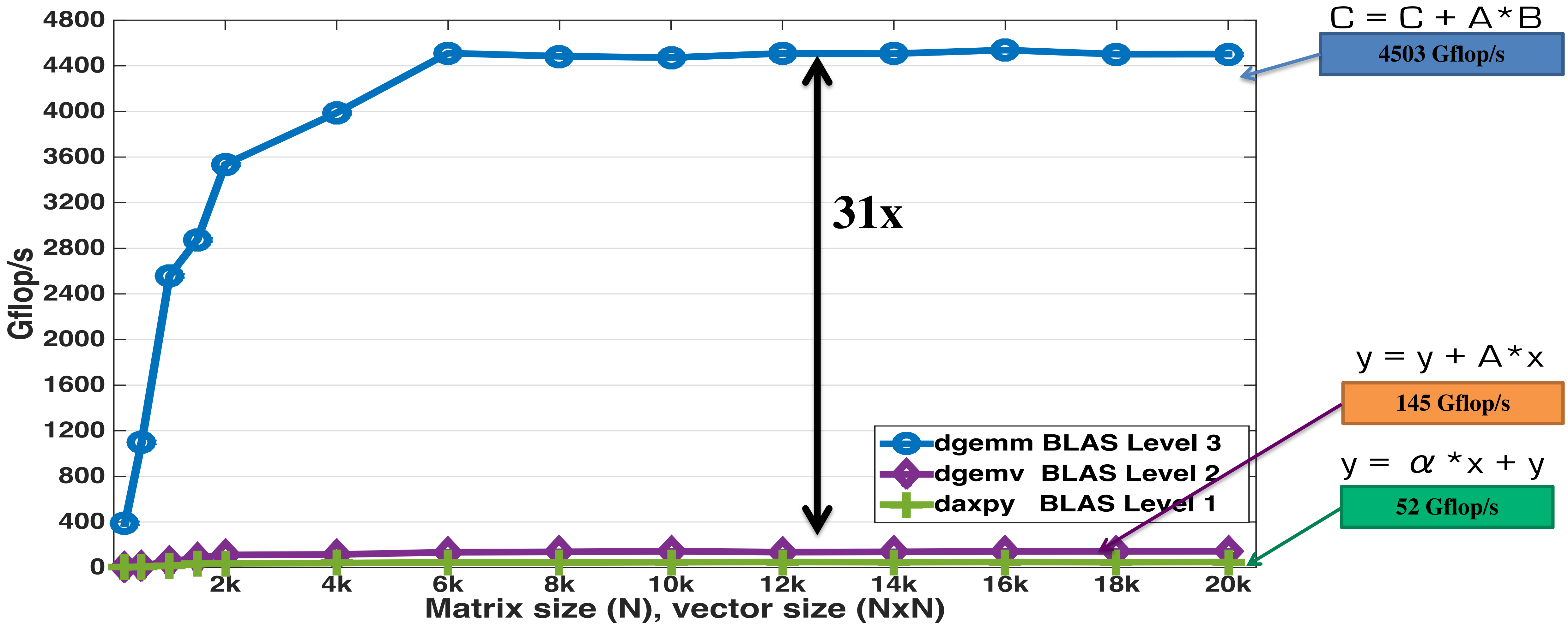
Use of BLAS for portability

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Level 1 **BLAS**
LAPACK (80's) (Blocking, cache friendly)		Level 3 **BLAS**
ScaLAPACK (90's) (Distributed Memory)		**PBLAS**
PLASMA (00's) New Algorithms (many-core friendly)		**BLAS** on tiles + DAG scheduling
MAGMA Hybrid Algorithms (heterogeneity friendly)		**BLAS** tasking + (CPU / GPU / Xeon Phi) hybrid scheduling
SLATE Tile Algorithms (distributed heterogeneous)		**Batched BLAS** tasking + hybrid scheduling

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

Level 1, 2 and 3 BLAS

Nvidia **P100**, 1.19 GHz, Peak DP = 4700 Gflop/s



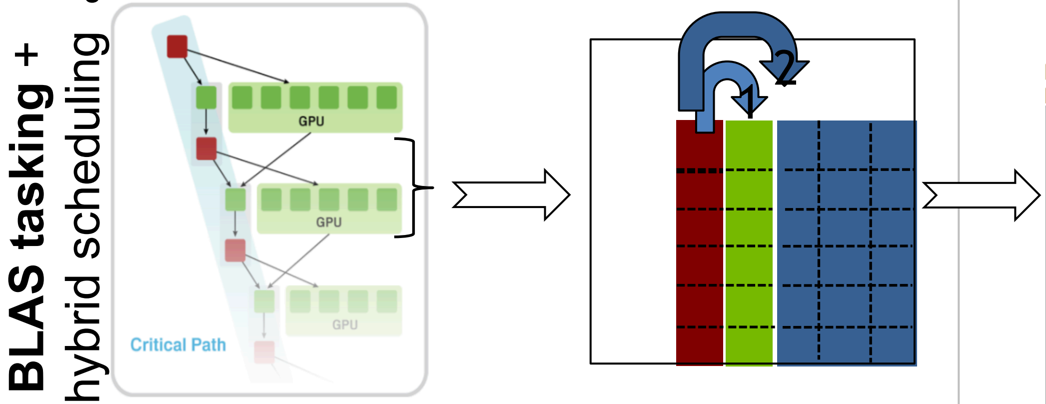
Nvidia P100
The theoretical peak double precision is 4700 Gflop/s
CUDA version 8.0

ICL INNOVATIVE COMPUTING LABORATORY

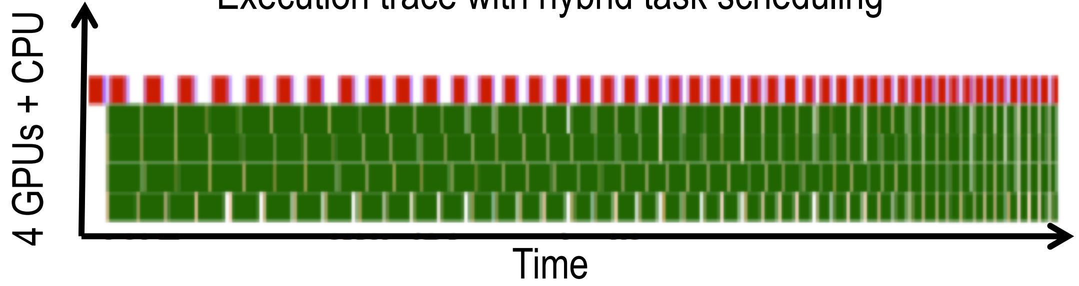
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Programming model: BLAS + scheduling

MAGMA
hybrid scheduling



Execution trace with hybrid task scheduling



MAGMA Dynamic

Left-looking hybrid Cholesky

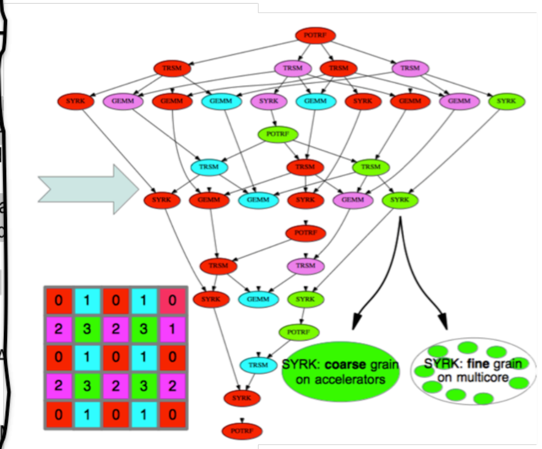
From sequential
LAPACK

```
for( j=0, j<n; j+=nb) {
  jb = min(nb, n-j);
  zherk( "Upper", "Conju
        &jb, &j, &one,

  if (j+jb < n)
    zgemm( "Conjugate
          dA(0,j), &ld

  zpotrf2( "Upper", &jb
  if (info != 0)
    *info += j;

  if (j+jb) < n) {

    ztrsm( "Left", "Uppe
          &jb, &n
  }
}
```

to parallel hybrid
MAGMA

```
for( j=0, j<n; j+=nb) {
  jb = min(nb, n-j);
  magma_zherk( MagmaUpper, MagmaConjTra
               jb, j, one, dA(0,j), ldda, one, dA
  magma_zgetmatrix_async( jb, jb, dA(j,j), ldd
  if (j+jb < n)
    magma_zgemm( MagmaConjTrans, Magma
                 dA(0,j), ldda, dA(0,j+jb), ldd
  magma_event_sync( event );
  zpotrf( MagmaUpperStr, &jb, work, &jb, info);
  if (info != 0)
    *info += j;
  magma_zsetmatrix_async(jb, jb, work, jb, dA
  If (j+jb) < n) {
    magma_event_sync( event );
    magma_ztrsm( MagmaLeft, MagmaUpper,
                 jb, n-j-jb, one, dA(j,j), ldda,
  }
}
```

MAGMA runtime environment



[A. Haidar, A. YarKhan, C. Cao, P. Luszczek, S. Tomov, and J. Dongarra, **"Flexible Linear Algebra Development and Scheduling with Cholesky Factorization"**, 17th IEEE International Conference on High Performance Computing and Communications, New York, August 2015.]

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

Programming model: BLAS + scheduling

- It follows asynchronous/dynamic execution phases of the panel and the update

- It hide the memory bound behavior of the panel factorization

For s = 0, nb, .. N
- **panel factorize**

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

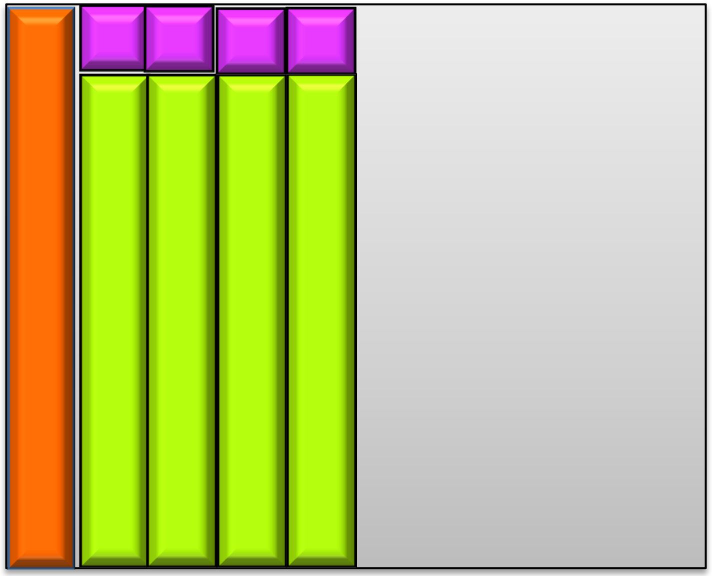
Programming model: BLAS + scheduling

- It follows asynchronous/dynamic execution phases of the panel and the update

- It hide the memory bound behavior of the panel factorization

For s = 0, nb, .. N

- panel factorize
- update next panel
- update remaining blocks



ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

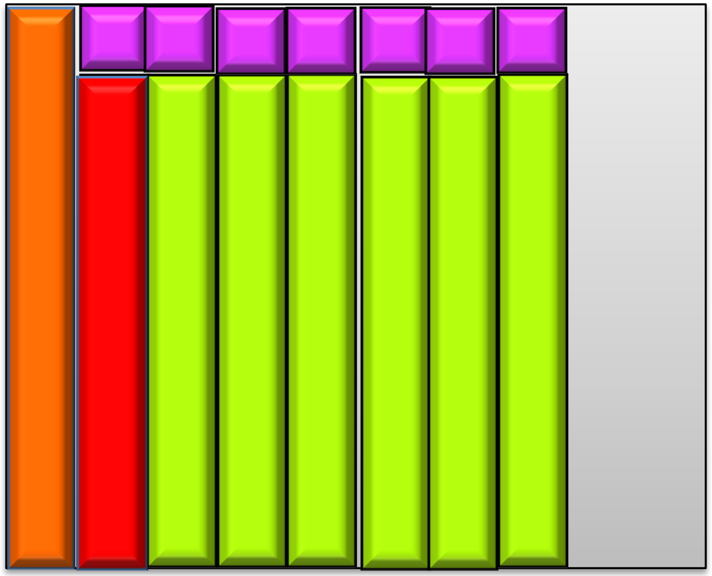
Programming model: BLAS + scheduling

- It follows asynchronous/dynamic execution phases of the panel and the update

- It hide the memory bound behavior of the panel factorization

For s = 0, nb, .. N

- **panel factorize**
- **update next panel**
- **update remaining blocks**



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

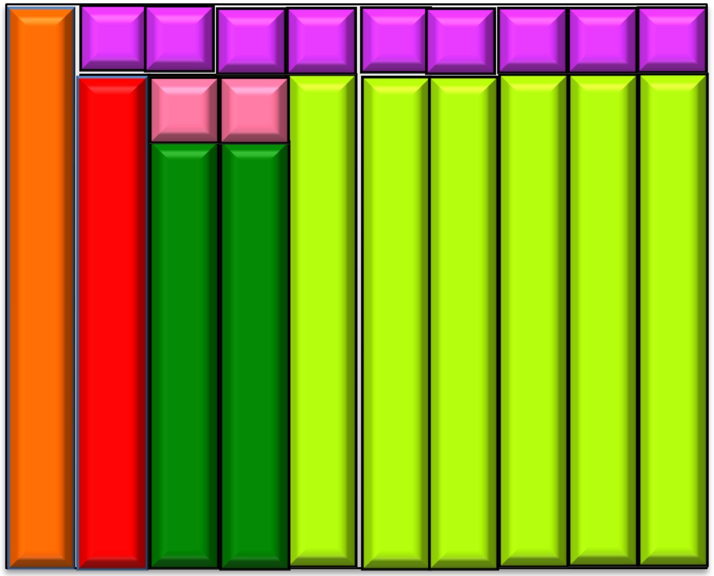
Programming model: BLAS + scheduling

- It follows asynchronous/dynamic execution phases of the panel and the update

- It hide the memory bound behavior of the panel factorization

For s = 0, nb, .. N

- **panel factorize**
- **update next panel**
- **update remaining blocks**

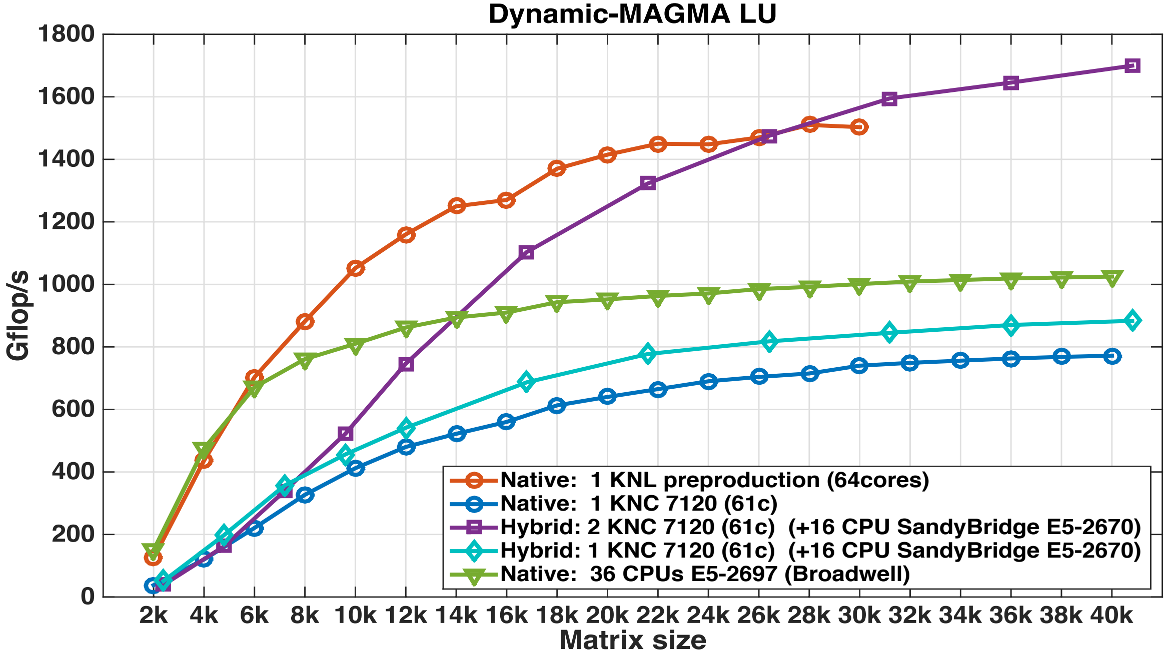


ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Polymorphic approach

- Hardware is generalized
  - Defined as virtual Master and Devices
  - Device can be group of cores, GPU, KNL, etc.
  - Similar for Master

- The same algorithm can run on this generalized hardware
  - User specified Master and Devices
  - Support different hardware configurations, including combinations, through one code

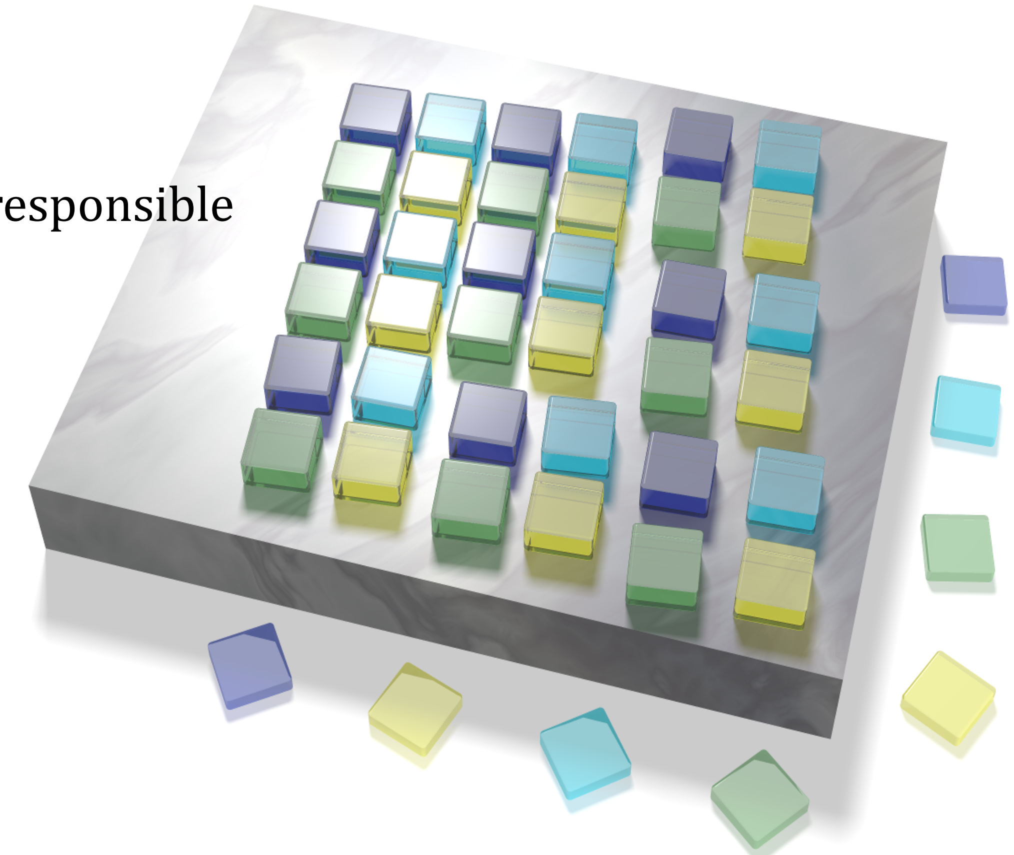


Performance of LU across 5 hardware configurations

SLATE
Software for Linear Algebra Targeting Exascale

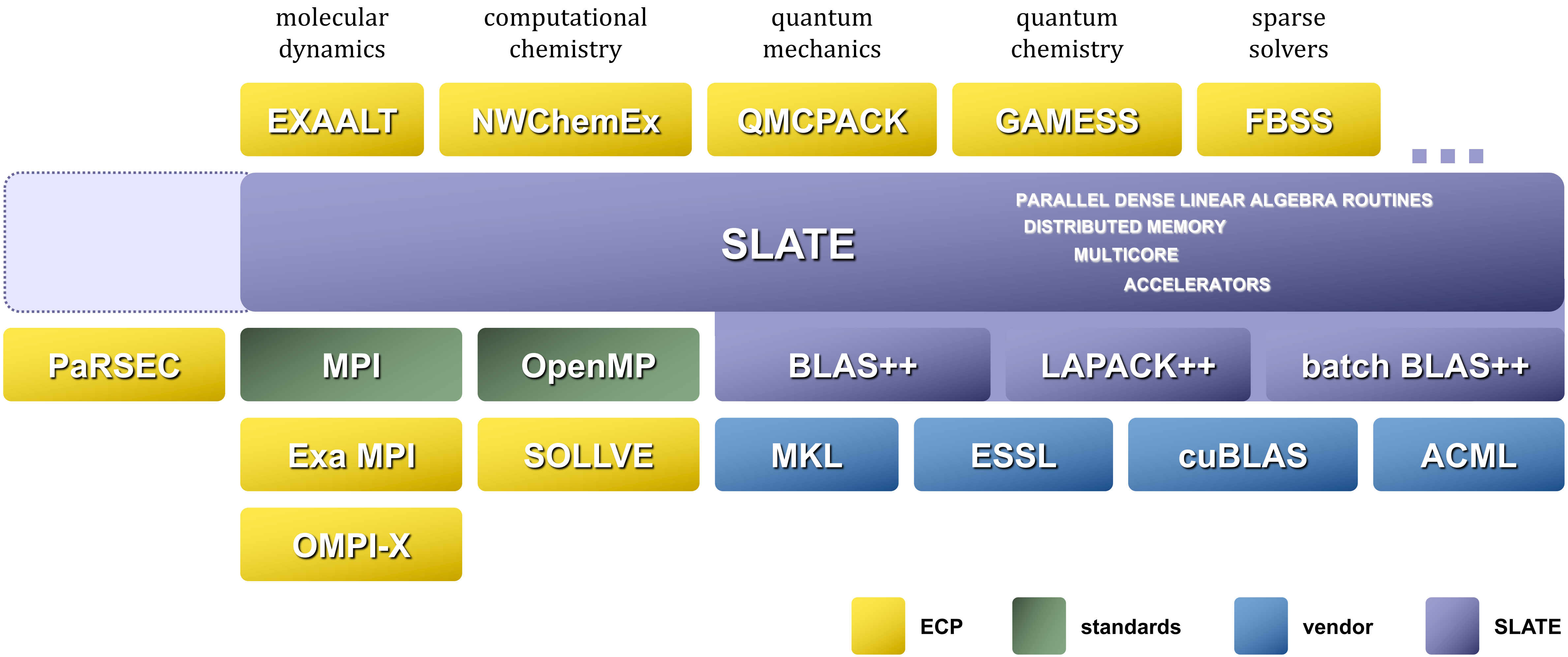
Jakub Kurzak, Mark Gates, Asim YarKhan, Ali Charara, Ichitaro Yamazaki, Jamie Finney, Jack Dongarra

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.



INNOVATIVE COMPUTING LABORATORY
Department of Electrical Engineering and Computer Science

SLATE Stack



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

SLATE Working Notes

http://www.icl.utk.edu/publications/series/swans

- **Designing SLATE: Software for Linear Algebra Targeting Exascale**

http://www.icl.utk.edu/publications/swan-003

- **C++ API for BLAS and LAPACK**

http://www.icl.utk.edu/publications/swan-002

https://bitbucket.org/icl/blaspp

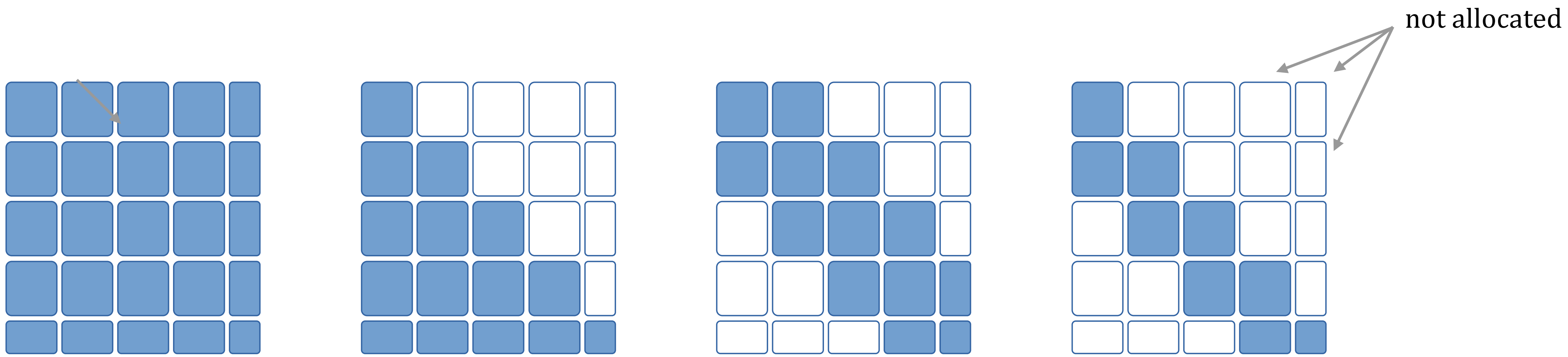
https://bitbucket.org/icl/lapackpp

- **Roadmap for the Development of a Linear Algebra Library for Exascale Computing:**

SLATE: Software for Linear Algebra Targeting Exascale

http://www.icl.utk.edu/publications/swan-001

SLATE Matrix




```
std::map<std::tuple<int64_t, int64_t, int>, Tile<FloatType>*> *tiles_;
```

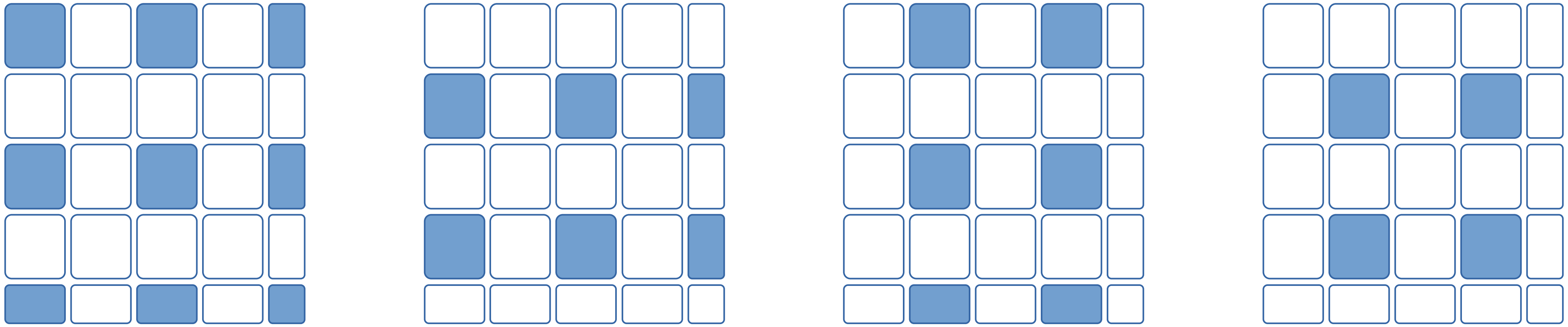
- collection of tiles
- **individually allocated**
- only allocate what is needed
- accommodates: symmetric, triangular, band, …

While in the PLASMA library the matrix is also stored in tiles, the tiles are laid out contiguously in memory.

In contrast, in SLATE, the tiles are individually allocated, with no correlation of their locations in the matrix to their addresses in memory.

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

SLATE Distributed Matrix

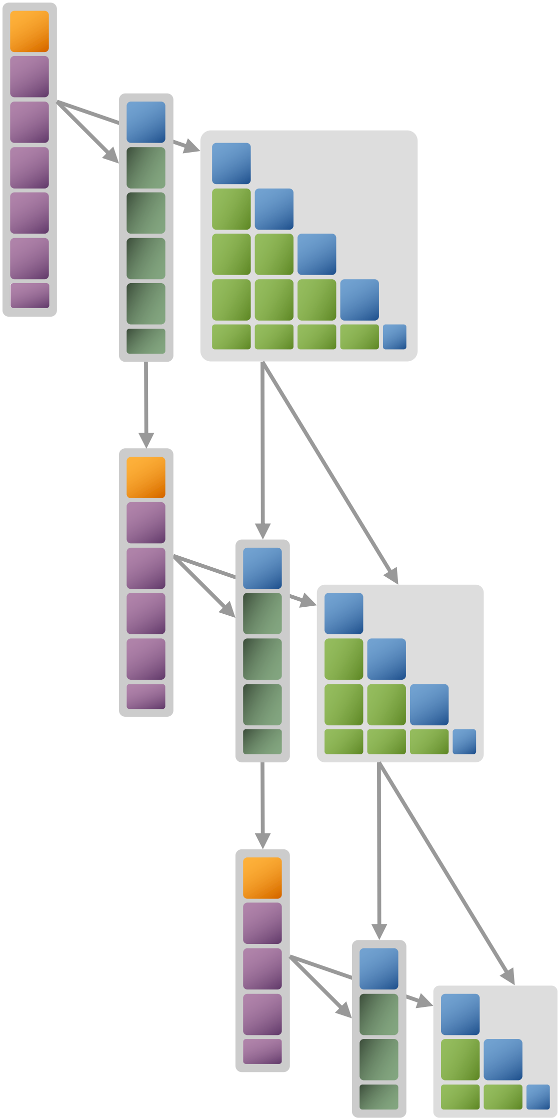


```
std::map<std::tuple<int64_t, int64_t, int>, Tile<FloatType>*> *tiles_;
```

- distributed matrix
- global indexing of tiles
- only allocate the local part
- any distribution is possible (2D block cyclic by default)

The same structure, used for single node representation, naturally supports distributed memory representation.

GEMM Scheduling



- nested parallelism
- top level: #pragma omp task depend
- bottom level:
  - #pragma omp task
  - batch GEMM

Accelerating memory-bound codes: the case of redesigning 3D FFTs for GPU-only execution

- FFTs needed in molecular dynamics, spectrum estimation, fast convolution and correlation, signal modulation, and wireless multimedia applications (although highly needed, FFT has not been actively developed and issues with licenses, etc.)

Main objectives

Design and implement a fast and robust 2-D and 3-D FFT library that targets large-scale heterogeneous systems with multi-core processors and hardware accelerators. Furthermore, FFT-ECP must be co-designed with other ECP application developers

Built from established but ad hoc software tools that have traditionally been part of application code:

- Collect existing FFT capabilities in ECP applications (LAMMPS/fftMPI and HACC/SWFFT)
- Assess gaps and make available as a sustainable FFT math library for ECP applications.

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

Accelerating memory-bound codes: the case of redesigning 3D FFTs for GPU-only execution

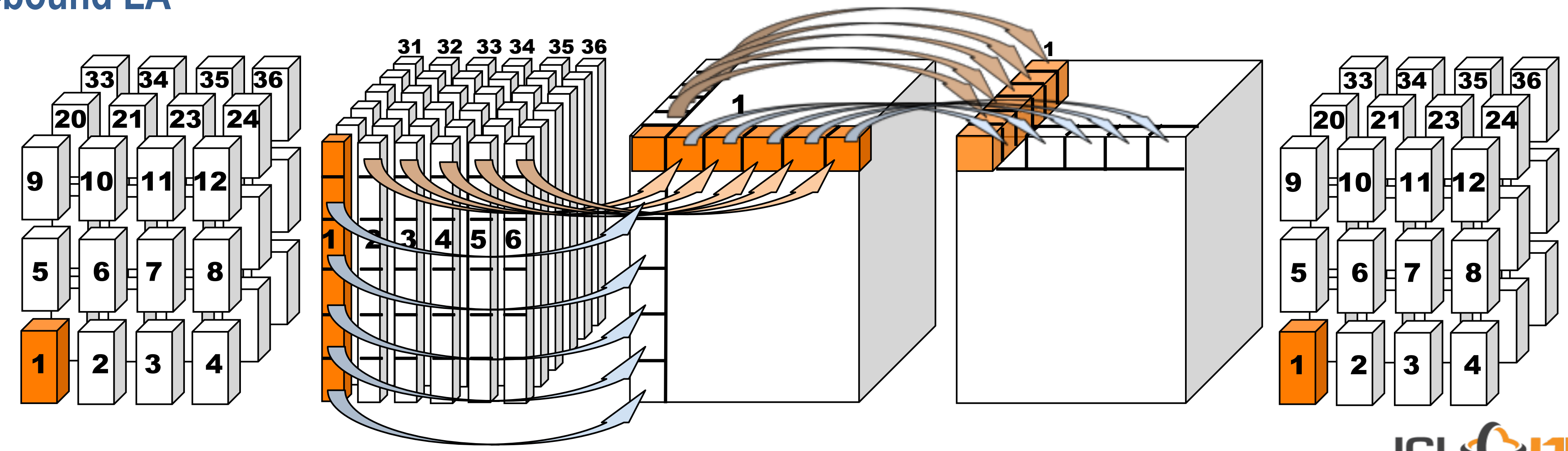
- FFTs needed in molecular dynamics, spectrum estimation, fast convolution and correlation, signal modulation, and wireless multimedia applications (although highly needed, FFT has not been actively developed and issues with licenses, etc.)

Main objectives

Design and implement a fast and robust 2-D and 3-D FFT library that targets large-scale heterogeneous systems with multi-core processors and hardware accelerators. Furthermore, FFT-ECP must be co-designed with other ECP application developers
Built from established but ad hoc software tools that have traditionally been part of application code:

- Collect existing FFT capabilities in ECP applications (LAMMPS/fftMPI and HACC/SWFFT)
- Assess gaps and make available as a sustainable FFT math library for ECP applications.

Memory-bound LA



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

FFT-ECP

- Need to scale on systems like Summit
  - Use CUDA-aware MPI (data stays on GPU memory all the time)
  - To achieve full bandwidth from a node
    - 2 x 2 x 12.5 GB/s = 50 GB/s ?

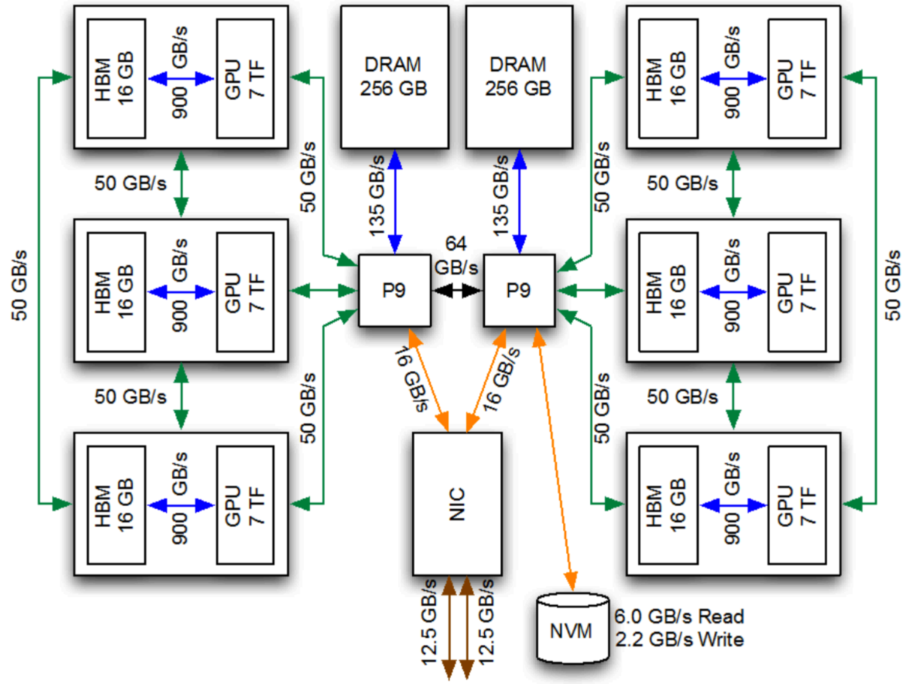
UNDER THE HOOD

Summit has fat nodes!

Many connections

Many devices

Many stacks



TF	42 TF (6x7 TF)
HBM	96 GB (6x16 GB)
DRAM	512 GB (2x16x16 GB)
NET	25 GB/s (2x12.5 GB/s)
MMsg/s	83

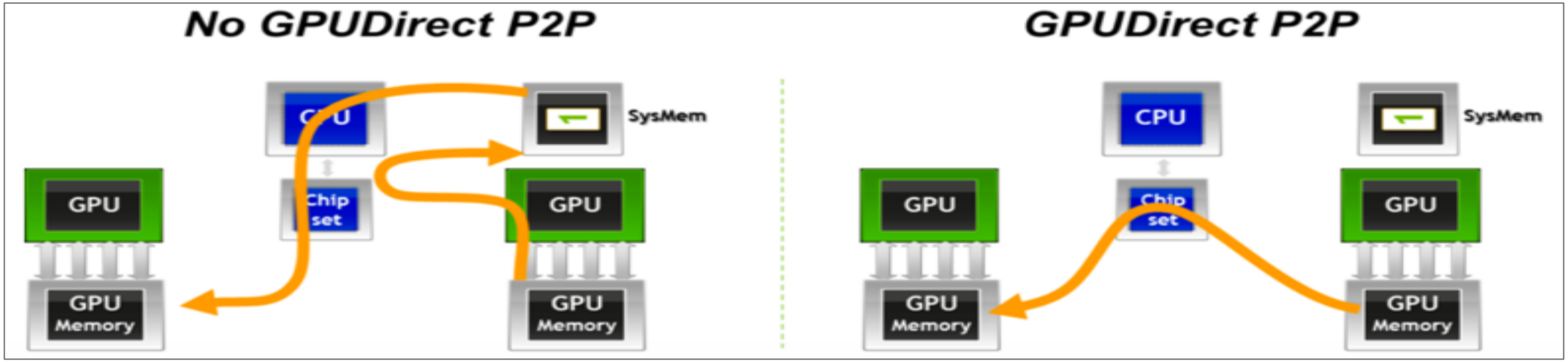
HBM/DRAM Bus (aggregate B/W)
NVLINK
X-Bus (SMP)
PCIe Gen4
EDR IB

HBM & DRAM speeds are aggregate (Read+Write).
All other speeds (X-Bus, NVLink, PCIe, IB) are bi-directional.

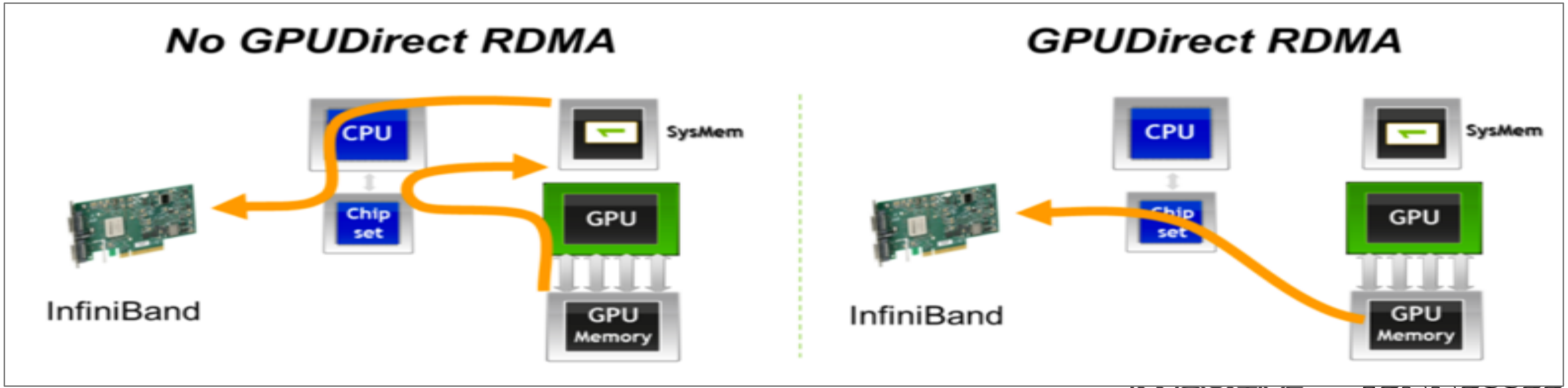
FFT-ECP – Communications is a main bottleneck

GPUDirect technologies in CUDA-aware MPI for fast communications:

P2P between GPUs on a node

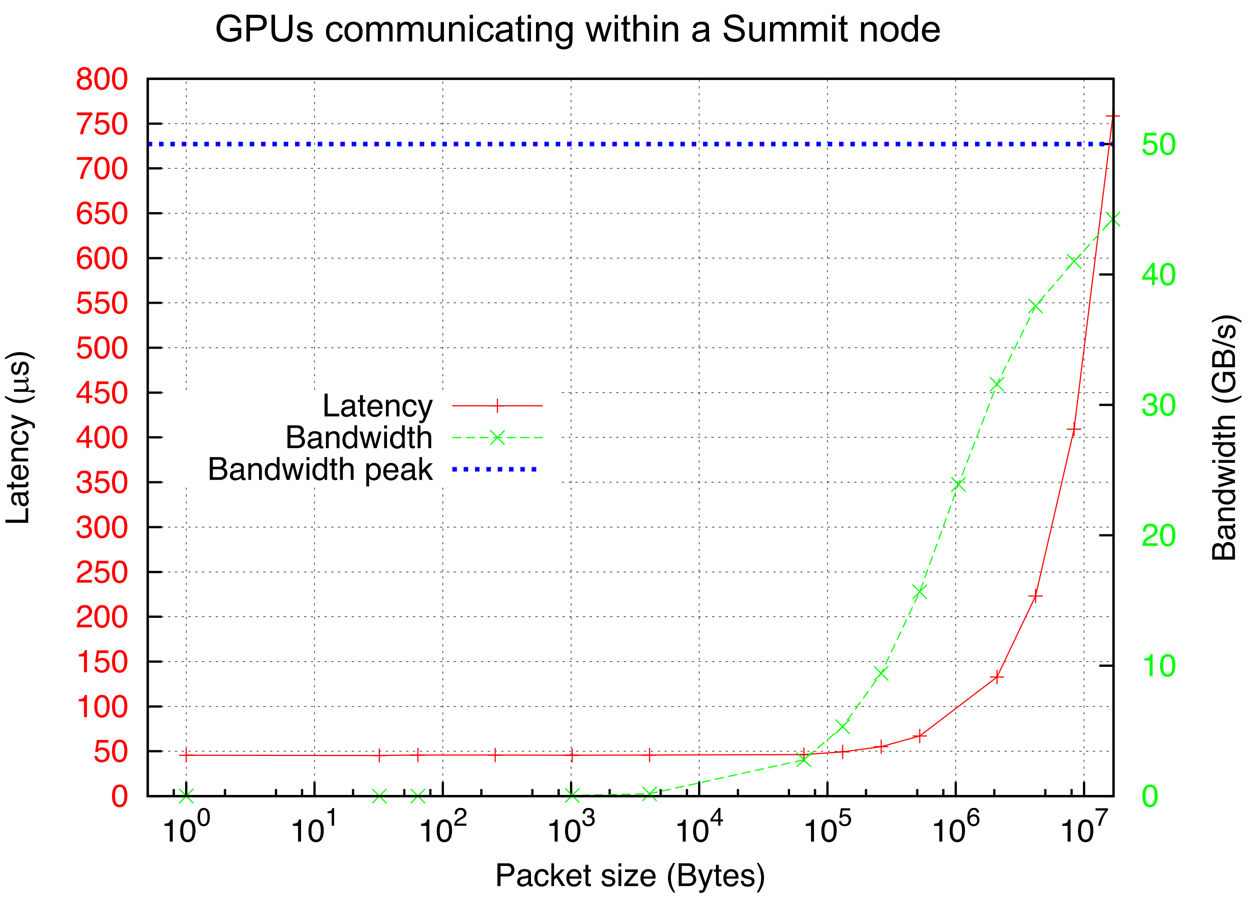


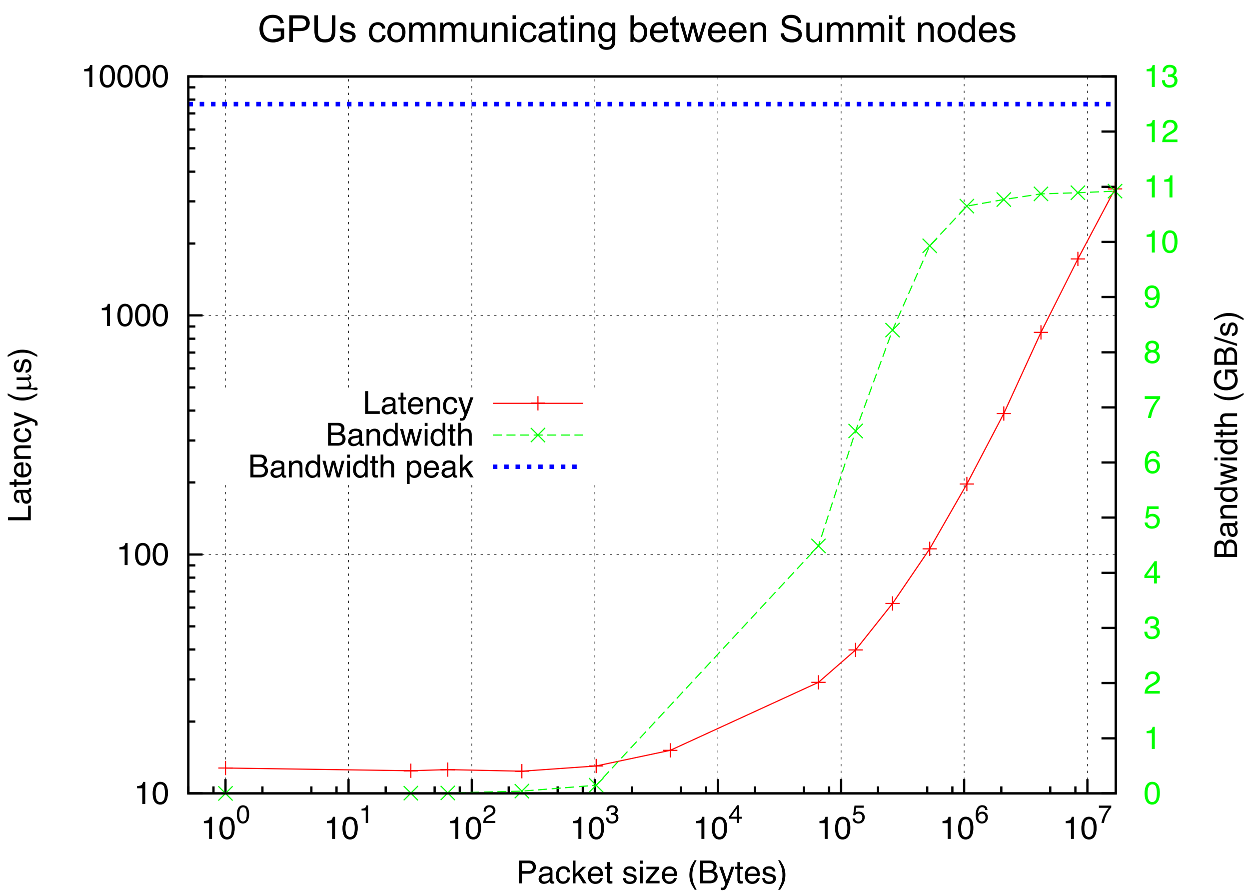
Remote Direct Memory Access (RDMA) for GPUs across nodes



FFT-ECP

- Point-to-point (GPU-to-GPU) communications achieve good asymptotic bandwidth
  Also, can benefit from duplexing, but latencies are high
- Collectives still need improvements





ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Performance bottlenecks

- Computations have been accelerated with GPUs (40x vs. CPU)

3D FFT on 4 nodes of summit for N = 1024 (24 V100 GPUs vs. 160 Power9 cores)

Main 3D FFT kernels	Time (ms) GPU	Time (ms) CPU	Overall GPU	Overall CPU
Unpack		123		216 ms
Batched 1D FFTs		63		
Pack		30		

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Performance bottlenecks

- Computations have been accelerated with GPUs (40x vs. CPU)

3D FFT on 4 nodes of summit for N = 1024 (24 V100 GPUs vs. 160 Power9 cores)

Main 3D FFT kernels	Time (ms) GPU	Time (ms) CPU	Overall GPU	Overall CPU
Unpack	2.1	123	5.7 ms	216 ms
Batched 1D FFTs	1.8	63		
Pack	1.8	30		

Performance bottlenecks

- Computations have been accelerated with GPUs (40x vs. CPU)
- MPI GPU communication though is slower even with GPU direct

3D FFT on 4 nodes of summit for N = 1024 (24 V100 GPUs vs. 160 Power9 cores)

Main 3D FFT kernels	Time (ms) GPU	Time (ms) CPU	Overall GPU	Overall CPU
Unpack	2.1	123	285.7 ms (140 Gflop/s)	368 ms (108 Glop/s)
Batched 1D FFTs	1.8	63		
Pack	1.8	30		
MPI A2A	280	152		

Performance bottlenecks

- Computations have been accelerated with GPUs (40x vs. CPU)
- MPI GPU communication though is slower even with GPU direct

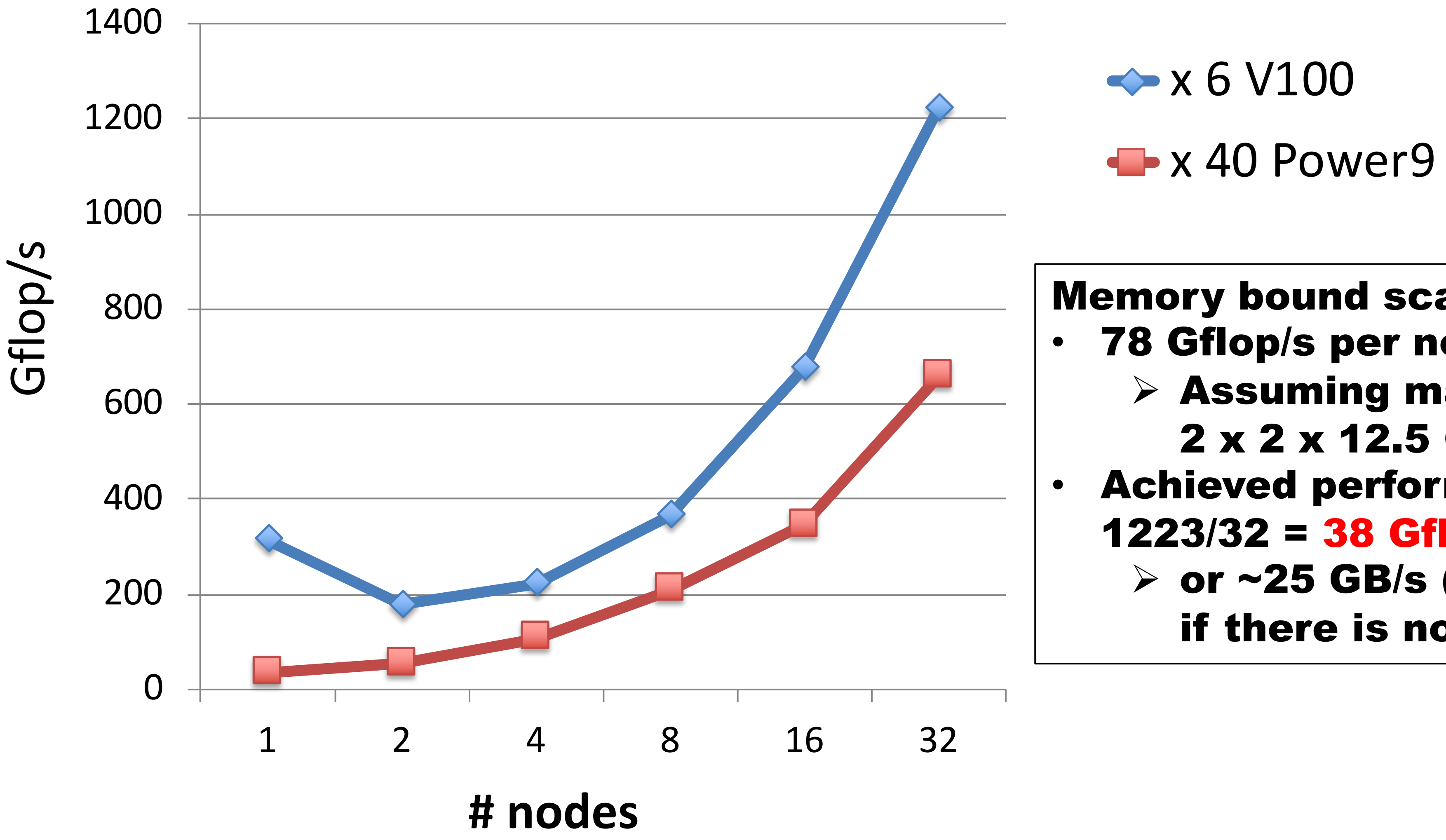
3D FFT on 4 nodes of summit for N = 1024 (24 V100 GPUs vs. 160 Power9 cores)

Main 3D FFT kernels	Time (ms) GPU	Time (ms) CPU	Overall GPU	Overall CPU
Unpack	2.1	123	285.7 ms (140 Gflop/s)	368 ms (108 Glop/s)
Batched 1D FFTs	1.8	63		
Pack	1.8	30		
MPI A2A	280	152		

- MPI GPU Direct A2A can/must be significantly improved
  - P2P are better; using them GPU version becomes 192 Gflop/s

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

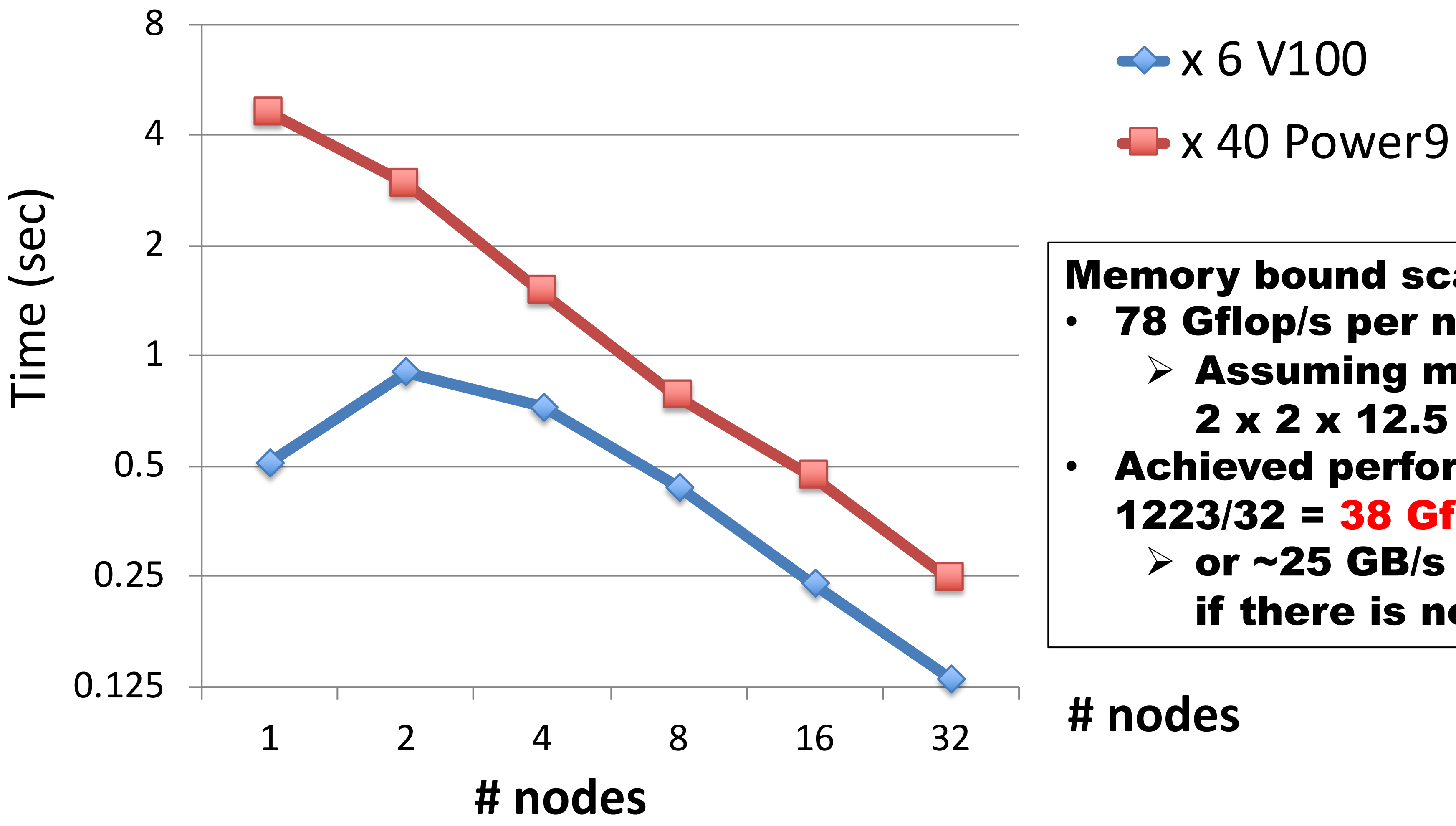
Strong scalability of 3D FFT on Summit (N = 1024)



Memory bound scalability peak:

- **78 Gflop/s per node**
  - ➢ **Assuming max bandwidth 2 x 2 x 12.5 GB/s = 50 GB/s**
- **Achieved performance is 1223/32 = 38 Gflop/s per node**
  - ➢ **or ~25 GB/s (this is maximum if there is no duplexing)**

Strong scalability of 3D FFT on Summit (N = 1024)



Memory bound scalability peak:

- **78 Gflop/s per node**
  - ➢ **Assuming max bandwidth 2 x 2 x 12.5 GB/s = 50 GB/s**
- **Achieved performance is 1223/32 = 38 Gflop/s per node**
  - ➢ **or ~25 GB/s (this is maximum if there is no duplexing)**

 # nodes

Conclusions

- Presented a number of software technologies for high-performance linear algebra targeting exascale computing
- Mixed-precision algorithms can accelerate significantly numerical solvers and applications
- Batched computations have many applications and can be accelerated significantly
- MAGMA and SLATE – redesigning dense linear algebra to still get close to machine peaks on new architectures
- Memory-bound codes, and 3D FFTs in particular, were redesigned to use GPU-Direct technologies for GPU-only execution
  – Computation accelerated 40x, leaving communication as main bottleneck (now 98% spent in MPI communications in targeted benchmarks)

Collaborators and Support

MAGMA team

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

PLASMA team

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

Collaborating partners

University of Tennessee, Knoxville
LLNL
ORNL
ANL
SANDIA
University of California, Berkeley
University of Colorado, Denver
TAMU
INRIA, France
KAUST, Saudi Arabia
University of Manchester, UK

EXASCALE COMPUTING PROJECT

CEED: Center for
Efficient Exascale Discretizations