SOS 23

Asheville, NC | Exascale and Beyond

SC18 PROGRAM EXHIBITS EXPERIENCE SUBMIT

Presentation

Based on a paper at SC18

FULL PROGRAM · PRESENTERS · ORGANIZATIONS · SEARCH PROGRAM · FLAGGED · HAPPENING NOW · MAPS · NOTIFICATIONS

Harnessing GPU's Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solvers

Session: Arithmetic and Optimization

Authors: Azzam Haidar, Stan Tomov, Jack Dongarra, Nicholas Higham

Azzam Haidar, Nvidia

Stan Tomov, UTK

Jack Dongarra, UTK/ORNL/U Manchester

Nick Higham, U of Manchester

Current #1 System Overview

System Performance

- Peak performance of 200 Pflop/s for modeling & simulation

Each node has

- 2 IBM POWER9 processors
  - Each w/22 cores
- 6 NVIDIA Tesla V100 GPUs
  - Each w/80 SMs
- 608 GB of fast memory
- 1.6 TB of NVMe memory

The system includes

- 4608 nodes
- Dual-rail Mellanox EDR InfiniBand network
- 250 PB IBM Spectrum Scale file system transferring data at 2.5 TB/s

Current #1 System Overview

System Performance

- Peak performance of 200 Pflop/s for modeling & simulation
- Peak performance of 3.3 Eflop/s for 16 bit floating point

Each node has

- 2 IBM POWER9 processors
  - Each w/22 cores
  - 2.3% performance of system
- 6 NVIDIA Tesla V100 GPUs
  - Each w/80 SMs
  - 97.7% performance of system
- 608 GB of fast memory
- 1.6 TB of NVMe memory

The system includes

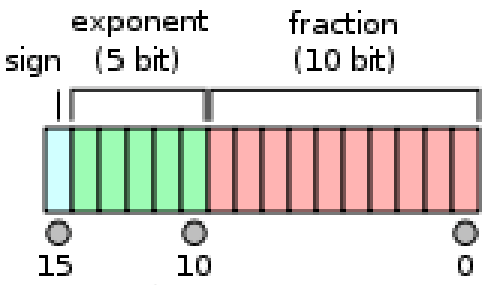
- 4608 nodes
- Dual-rail Mellanox EDR InfiniBand network
- 250 PB IBM Spectrum Scale file system transferring data at 2.5 TB/s

Mixed Precision

- Today many precisions to deal with (IEEE Standard)

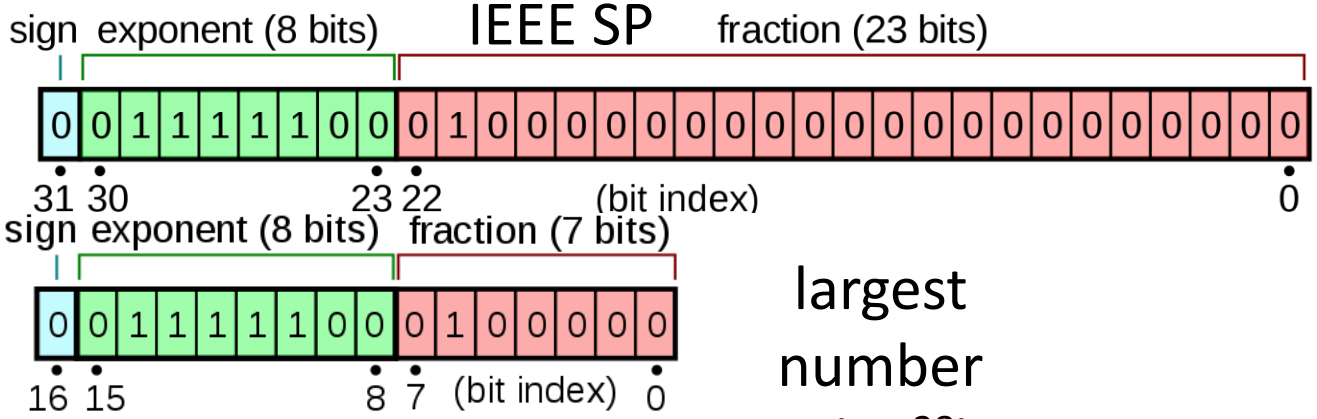
Type	Size	Range	$u = 2^{-t}$
half	16 bits	$10^{\pm 5}$	$2^{-11} \approx 4.9 \times 10^{-4}$
single	32 bits	$10^{\pm 38}$	$2^{-24} \approx 6.0 \times 10^{-8}$
double	64 bits	$10^{\pm 308}$	$2^{-53} \approx 1.1 \times 10^{-16}$
quadruple	128 bits	$10^{\pm 4932}$	$2^{-113} \approx 9.6 \times 10^{-35}$

- Note the number range with half precision (16 bit fl.pt.)



float16

largest number O(65,000)



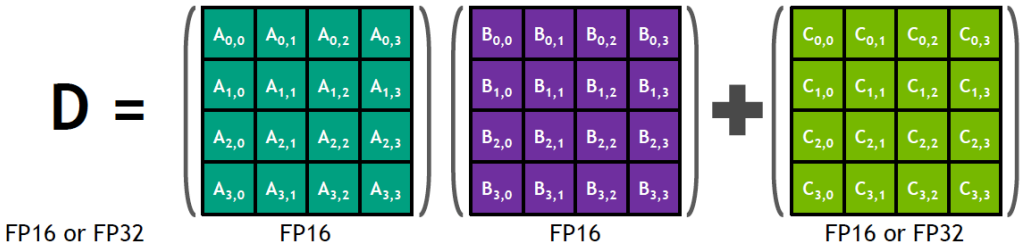
bfloat16

largest number O(10^{38})

Nvidia Volta Peak Rates

- Four Performance levels for the different precision
  - 64 bit floating point (FMA): 7.5 Tflop/s
  - 32 bit floating point (FMA): 15 Tflop/s
  - 16 bit floating point (FMA): 30 Tflop/s
  - 16 bit floating point with Tensor core: 120 Tflop/s

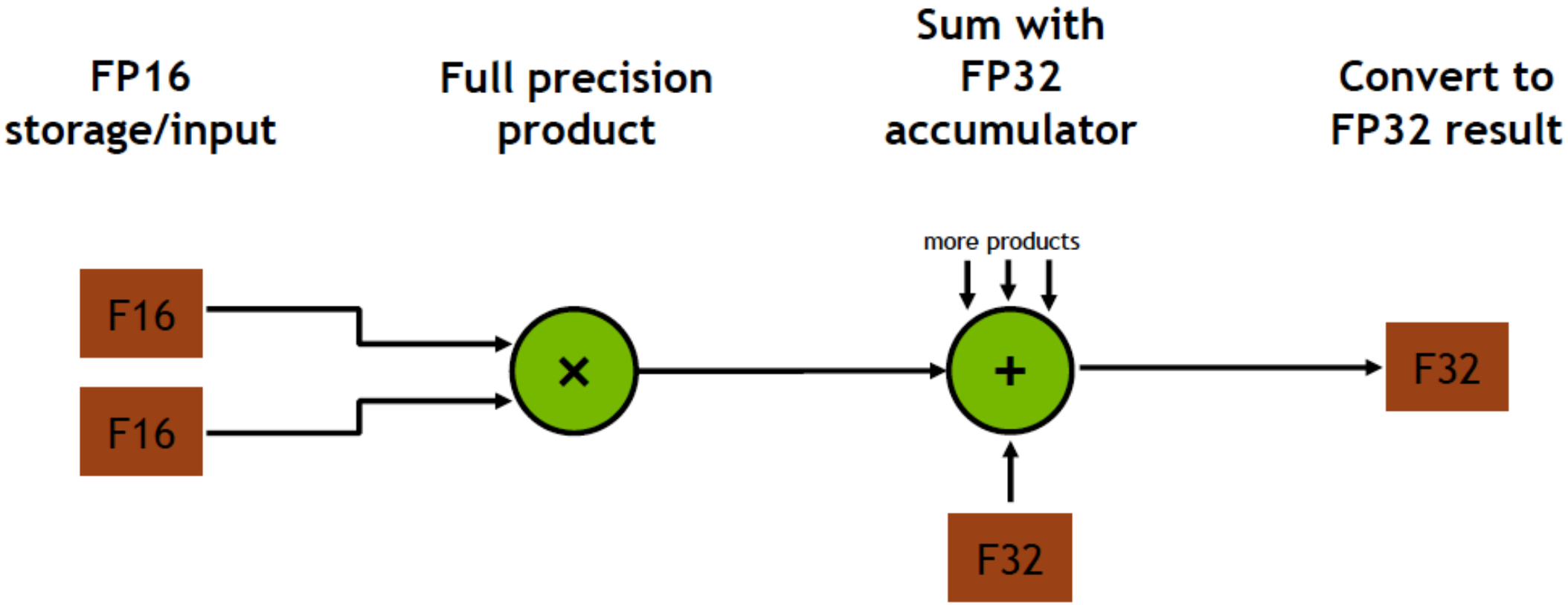
Mixed Precision Matrix Multiply 4x4 Matrices



$$D = AB + C$$

07

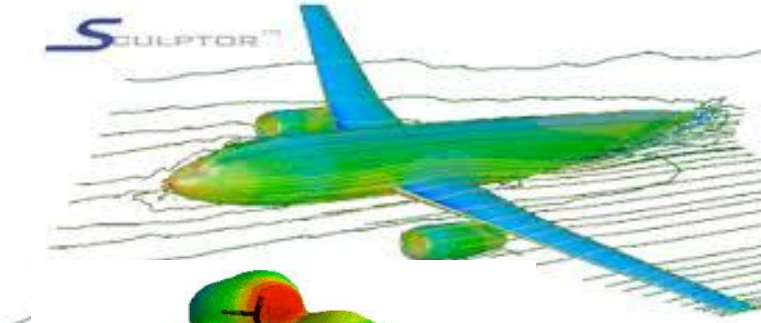
VOLTA TENSOR OPERATION

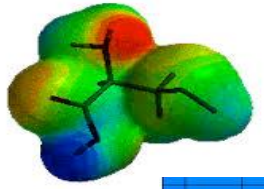


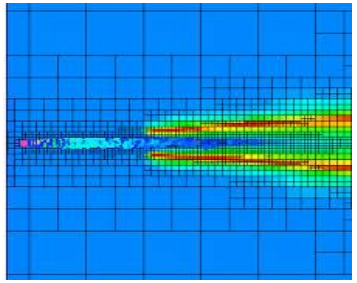
Also supports FP16 accumulator mode for inferencing

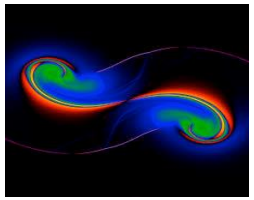
Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications:

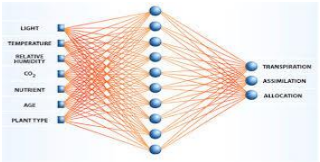
- **Linear systems:** Solve $Ax = b$
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** Find x to minimize $\| Ax - b \|$
  - Computational 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.
- **DLA is crucial to the development of sparse solvers**











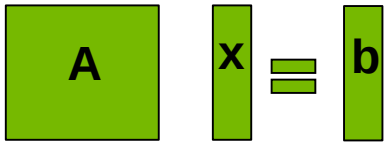
ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Leveraging Half Precision in HPC on V100 solving linear system Ax = b

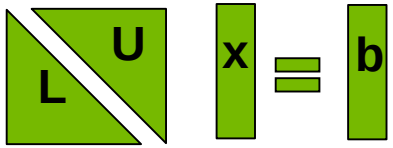
solving linear system Ax = b
LU factorization

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

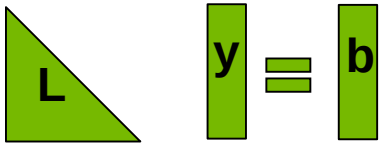
```
A   x = b
```




```
LUx = b
```




```
Ly    = b
```



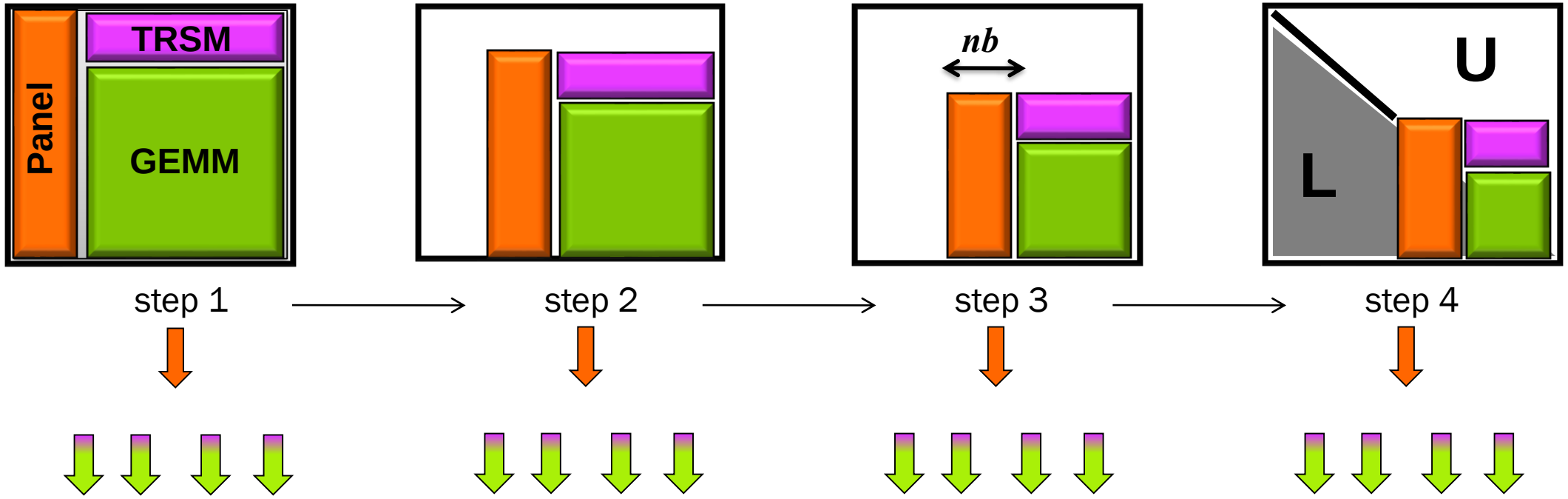

```
then
Ux    =
y
```

Leveraging Half Precision in HPC on V100 solving linear system Ax = b

```
For  s = 0, nb, .. N
```

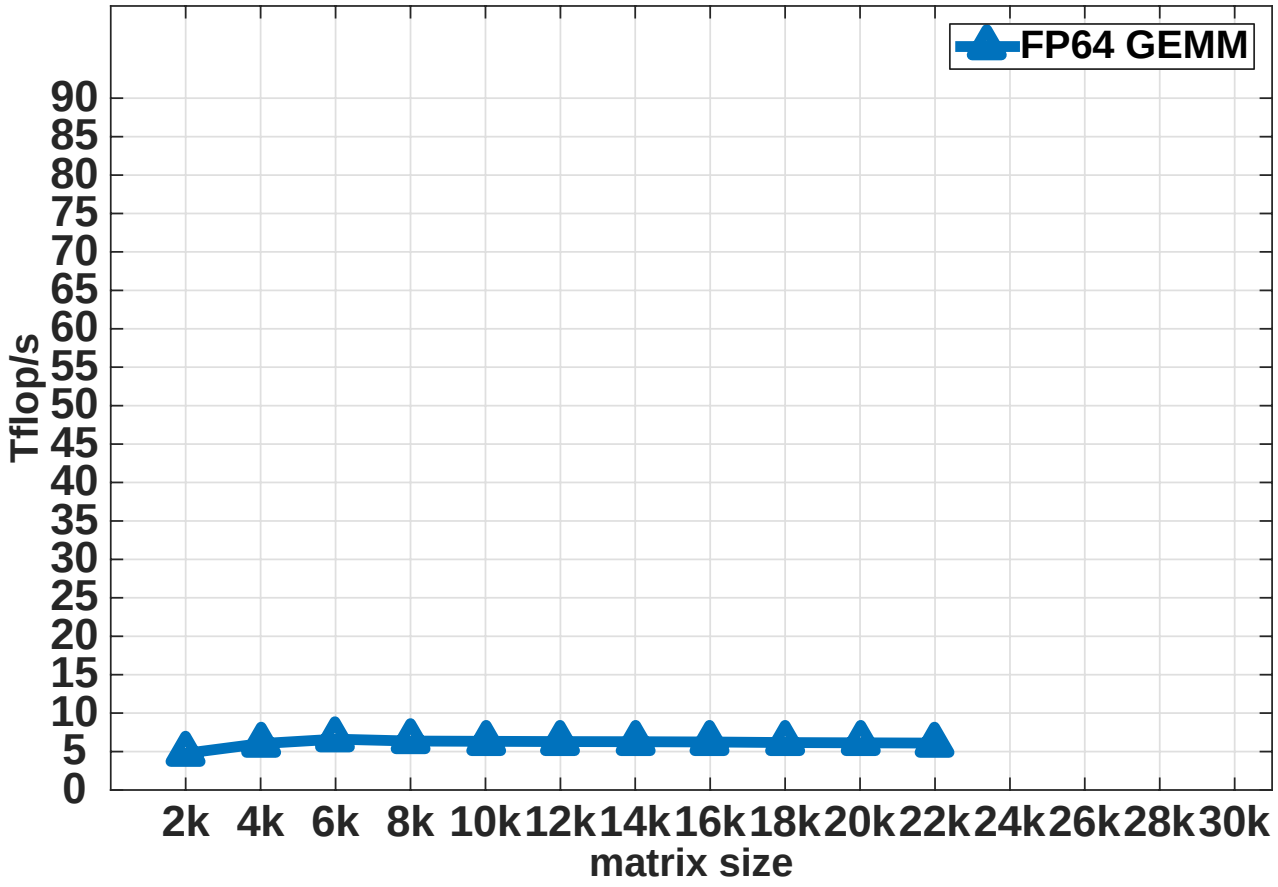
1. panel factorize
2. update trailing matrix

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



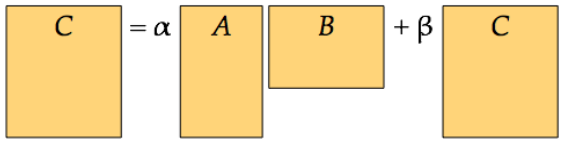
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100



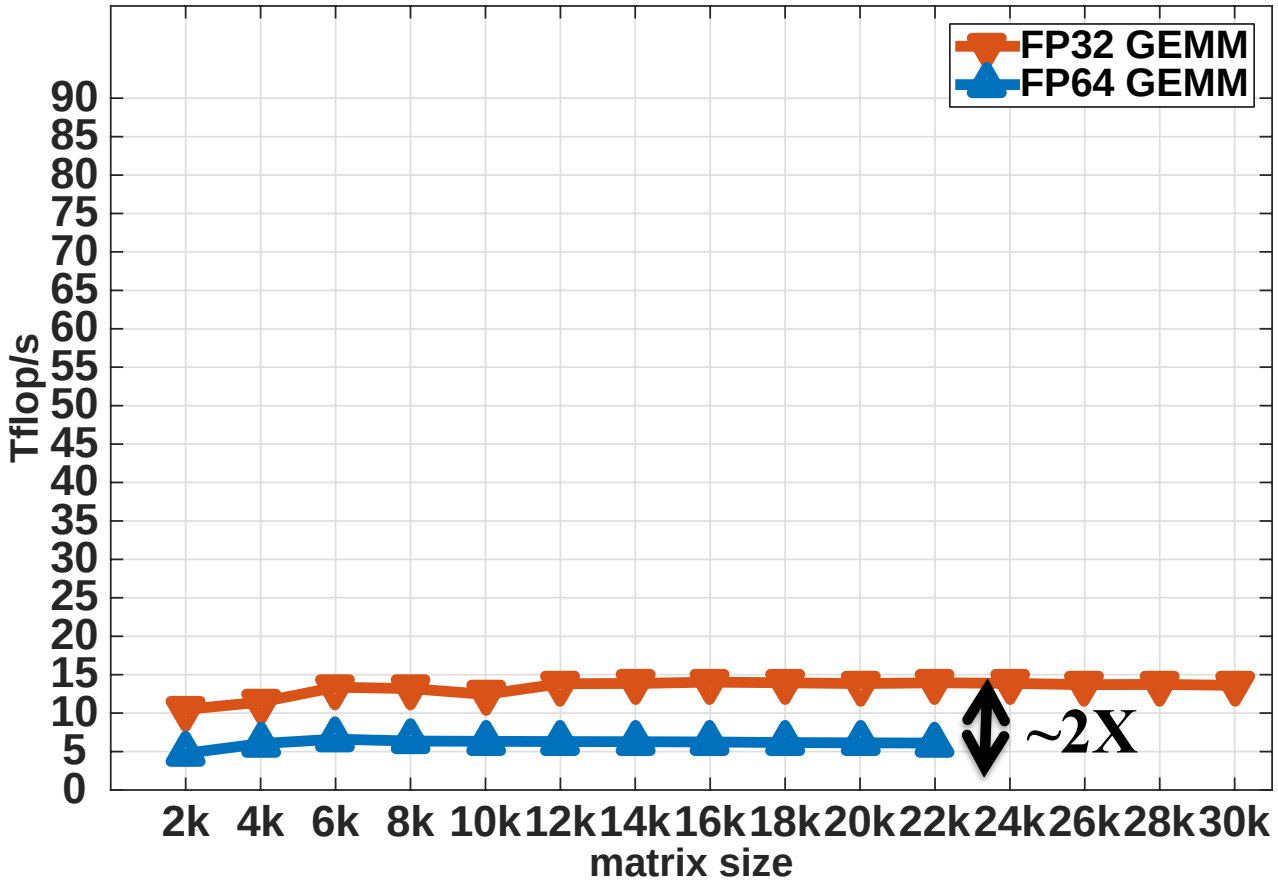
- dgemm achieve about 6.4 Tflop/s

Matrix matrix multiplication GEMM



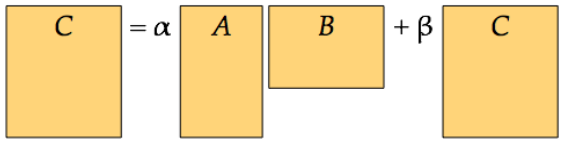
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100



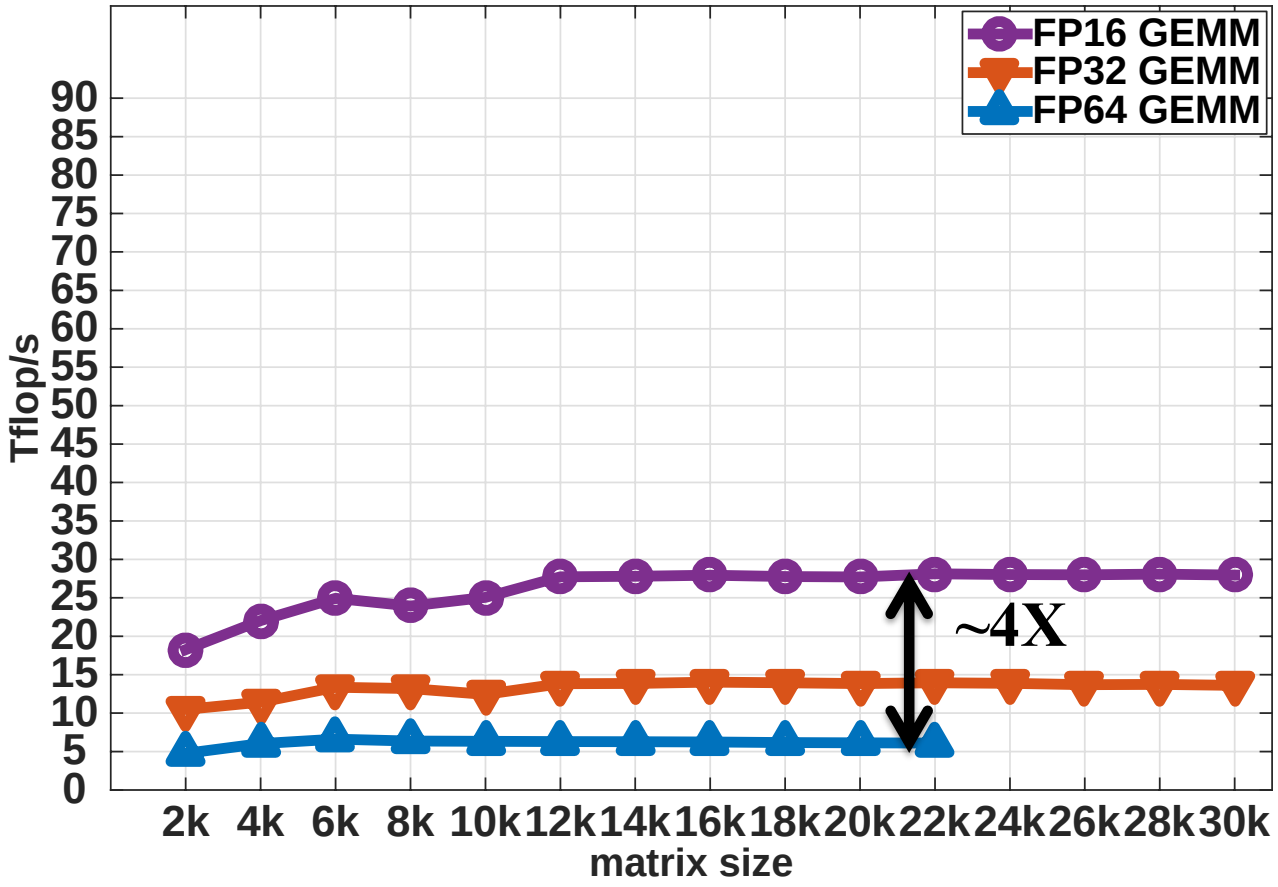
- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s

Matrix matrix multiplication GEMM



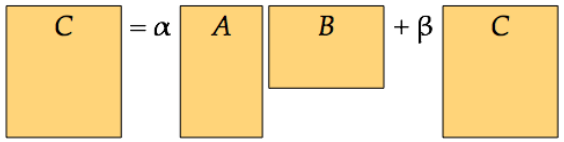
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100



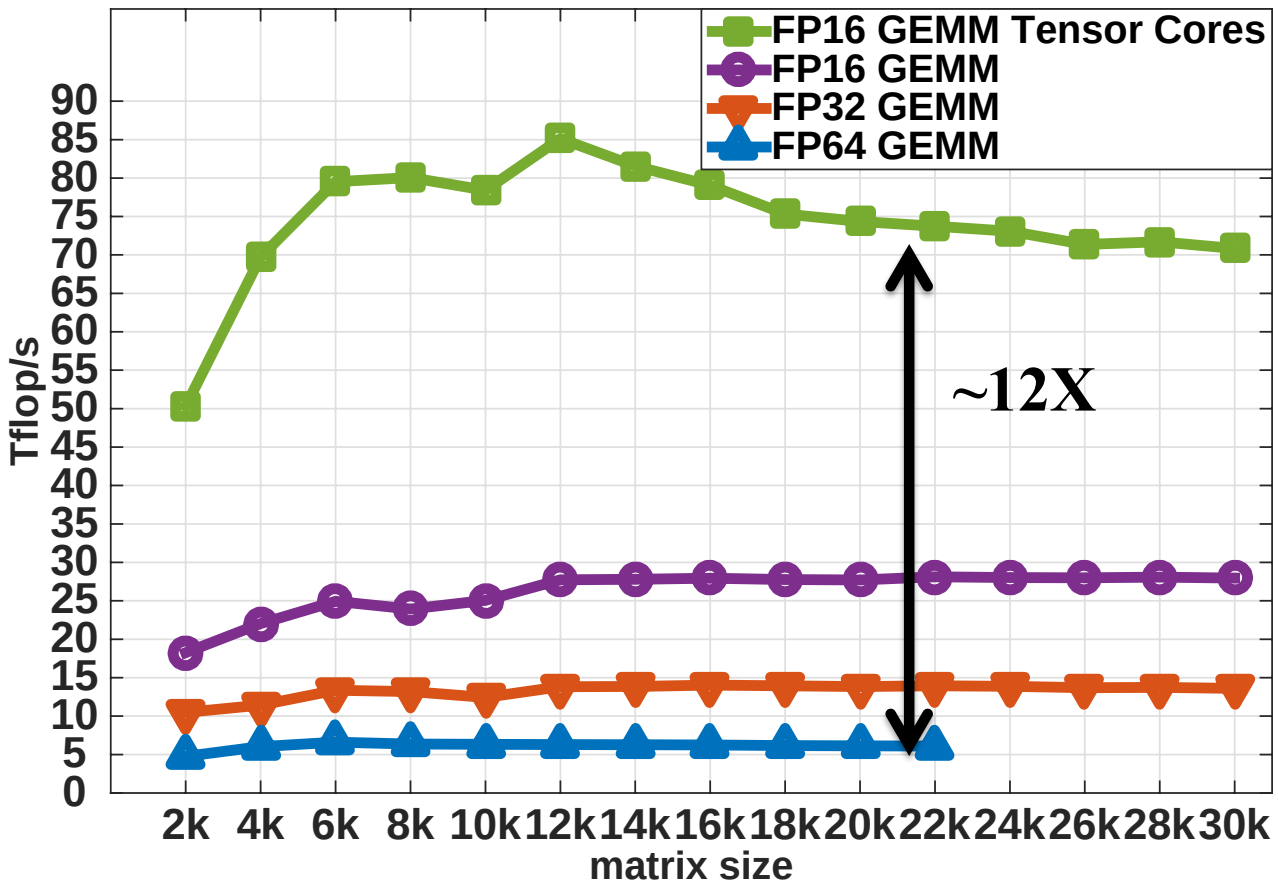
dgemm achieve about 6.4 Tflop/s
sgemm achieve about 14 Tflop/s
hgemm achieve about 27 Tflop/s

Matrix matrix multiplication GEMM



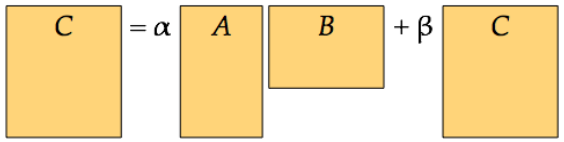
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100



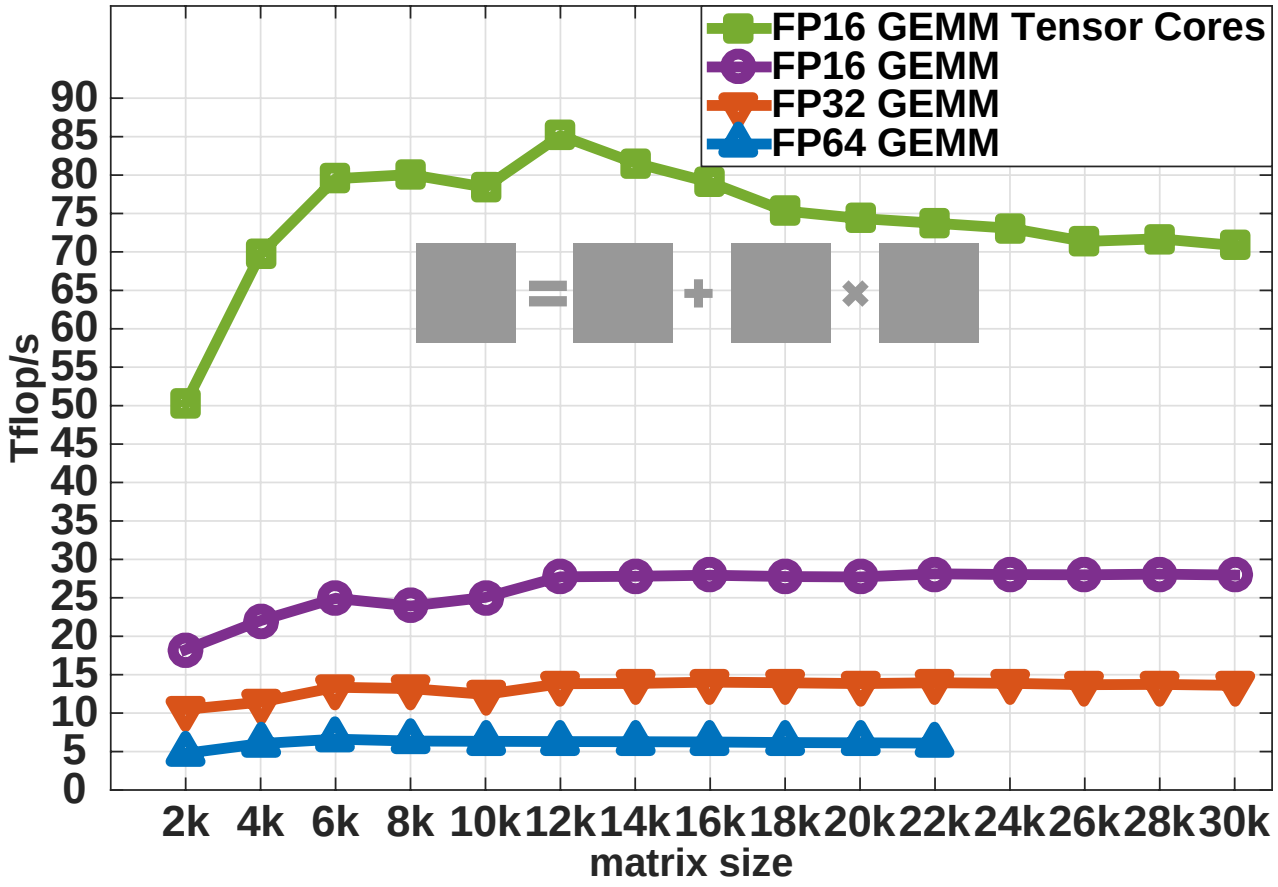
dgemm achieve about 6.4 Tflop/s
sgemm achieve about 14 Tflop/s
hgemm achieve about 27 Tflop/s
Tensor cores gemm reach about 85 Tflop/s

Matrix matrix multiplication GEMM



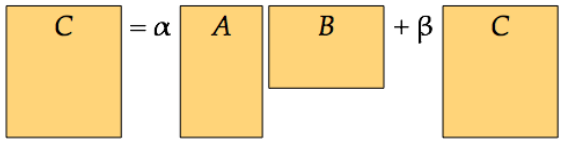
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100



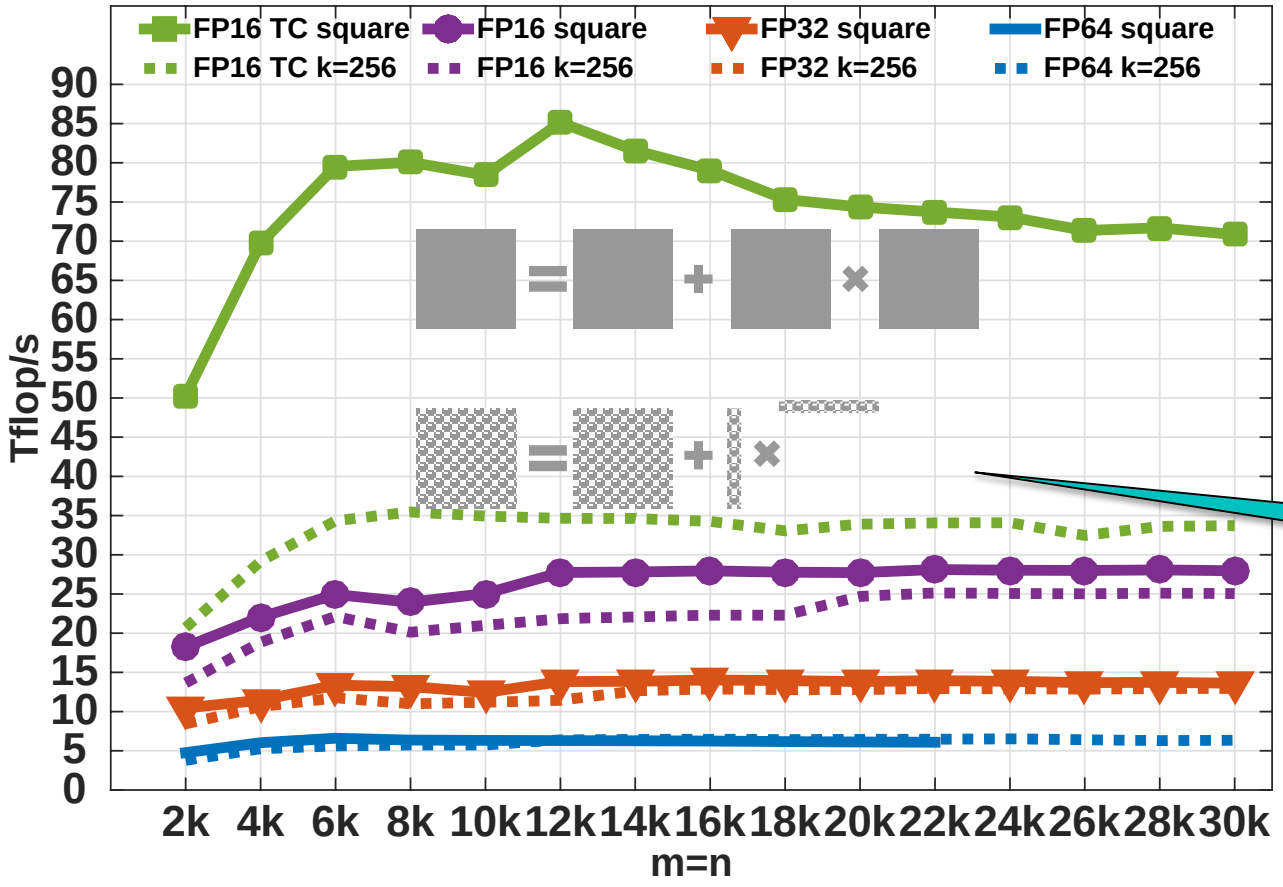
dgemm achieve about 6.4 Tflop/s
sgemm achieve about 14 Tflop/s
hgemm achieve about 27 Tflop/s
Tensor cores gemm reach about 85 Tflop/s

Matrix matrix multiplication GEMM



Leveraging Half Precision in HPC on V100

Study of the rank k update used by the LU factorization algorithm on Nvidia V100

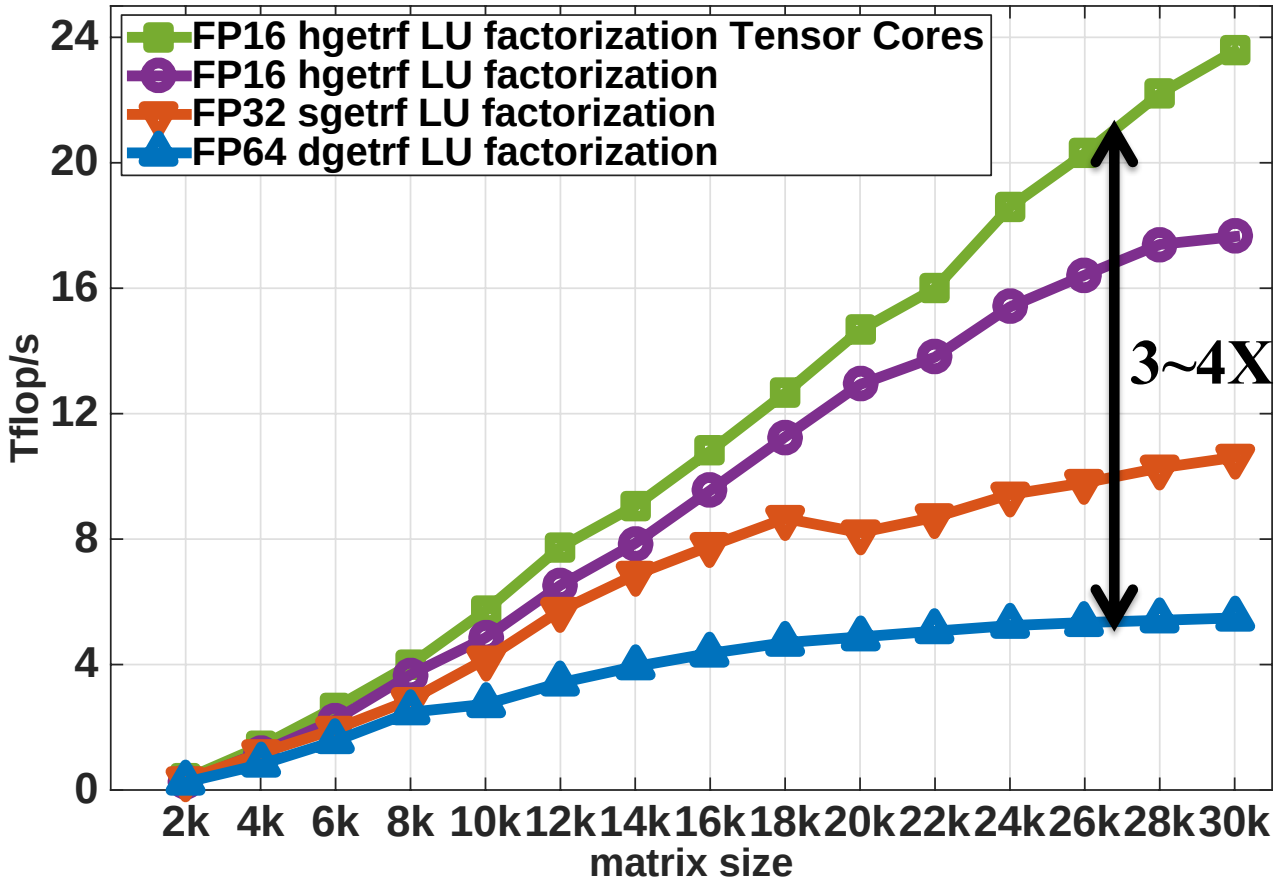


- In LU factorization need matrix multiple but operations is a rank-k update computing the Schur complement

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

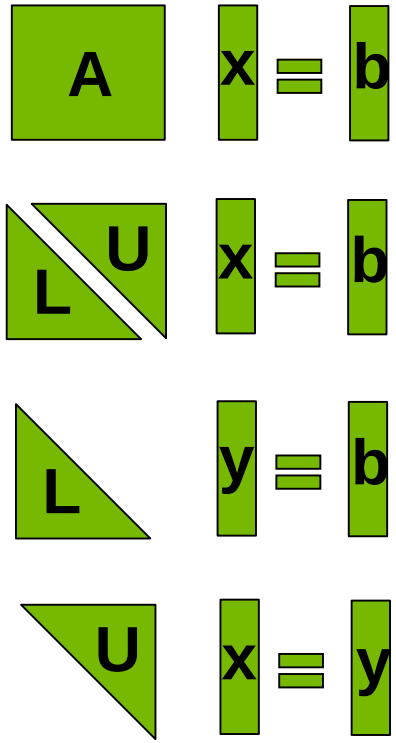
Leveraging Half Precision in HPC on V100

Study of the LU factorization algorithm on Nvidia V100



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

```
A   x = b
LUx = b
Ly    = b
then
Ux    = y
```



Leveraging Half Precision in HPC on V100

Use Mixed Precision algorithms

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

Reference:

A. Haidar, P. Wu, S. Tomov, J. Dongarra,
Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers,
SC-17, ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ACM, Denver, Colorado, November 12-17, 2017.

A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham,
Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers, SC-18, Dallas, TX, IEEE, November 2018.

Leveraging Half Precision in HPC on V100

Idea: use low 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.

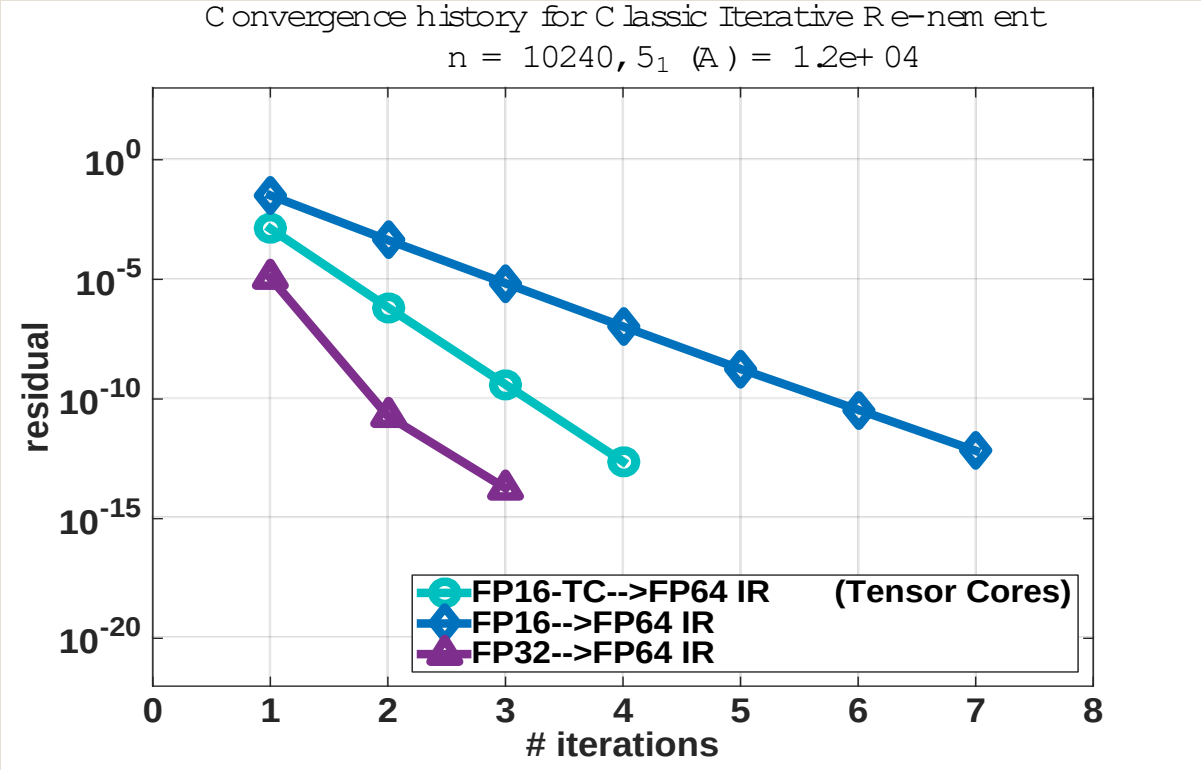
L U = lu(A)		lower precision	$O(n^3)$
x = U\(L\b)		lower precision	$O(n^2)$
r = b – Ax		FP64 precision	$O(n^2)$
WHILE || r || not small enough			
1. find a correction "z" to adjust x that satisfy Az=r			
solving Az=r could be done by either:			
➢ z = U\(L\r)	Classical Iterative Refinement	lower precision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

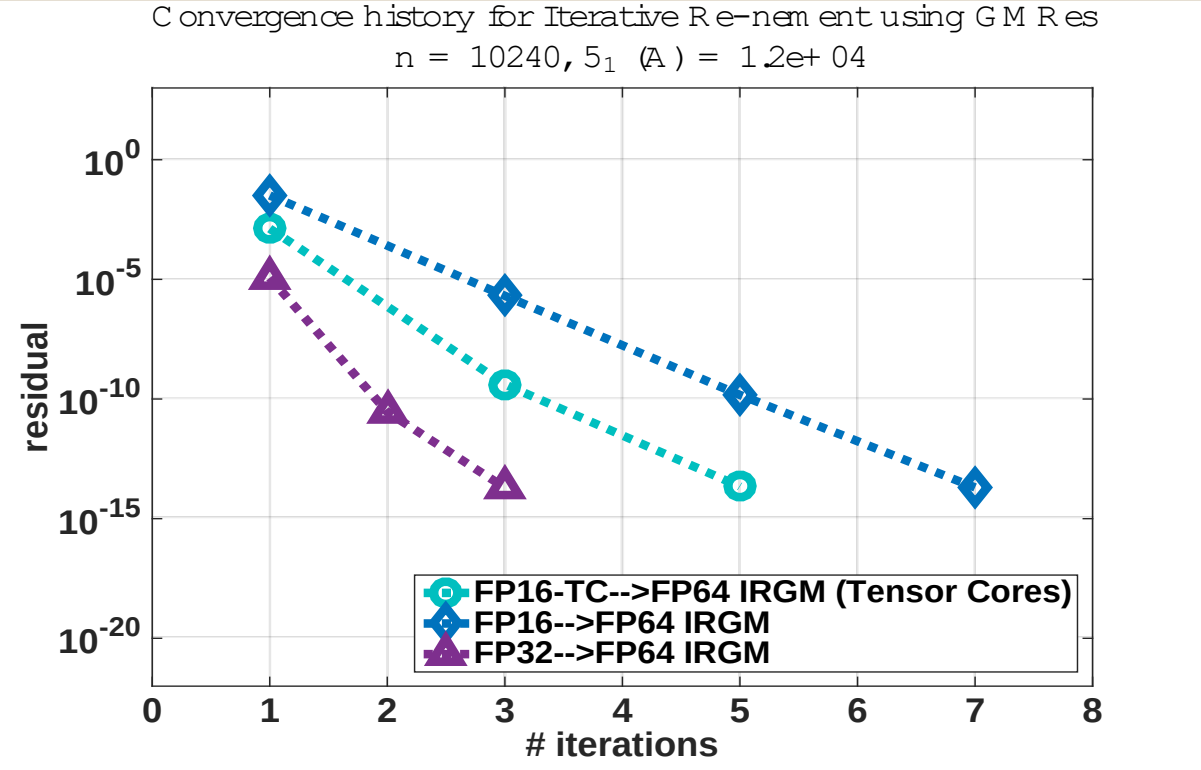
Higham and Carson showed can solve the inner problem with iterative method and not infect the solution.

- 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.
- Need the original matrix to compute residual (r) and matrix cannot be too badly conditioned

E. Carson & N. Higham, "Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions *SIAM J. Sci. Comput.*, 40(2), A817–A847.

Leveraging FP16 in HPC on V100: Numerical Behavior

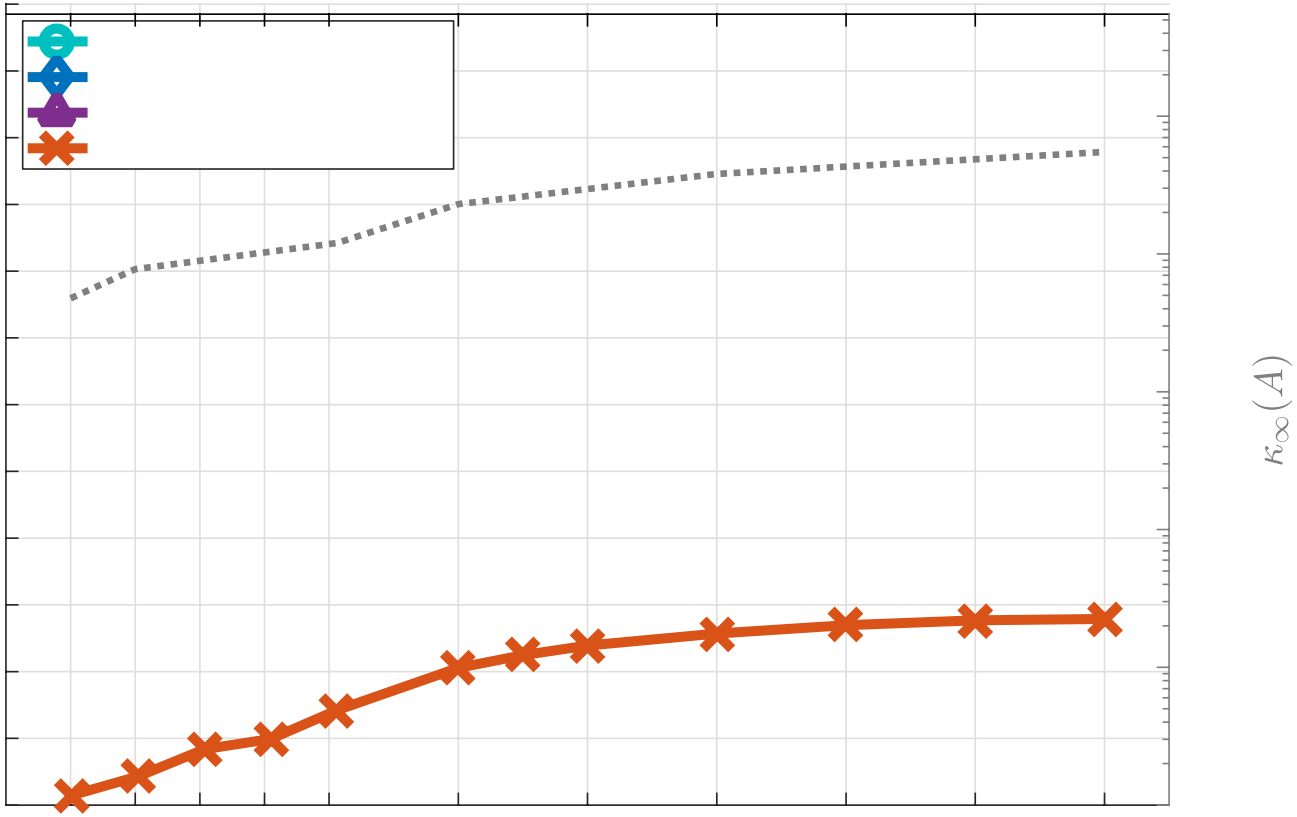




- Convergence history of the iterative refinement solver to achieve FP64 solution accuracy.
- Left graph shows the classical iterative refinement method.
- Right graph shows the iterative refinement using GMRes.
- Problem generated with an arithmetic distribution of the singular values $s_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

- The **FP32->64** algorithm converge as expected and is able to achieve the FP64 solution accuracy in about 3-5 iterations.
- The **FP16->64** algorithm requires more iterations than FP32→64 because of the lower precision factorization.
- The **FP16->64 (Tensor Cores)** outperforms the FP16->64 because the **accumulation** during the FP16-TC GEMM (Schur update) **use FP32-bit**.

Leveraging Half Precision in HPC on V100
Performance Behavior

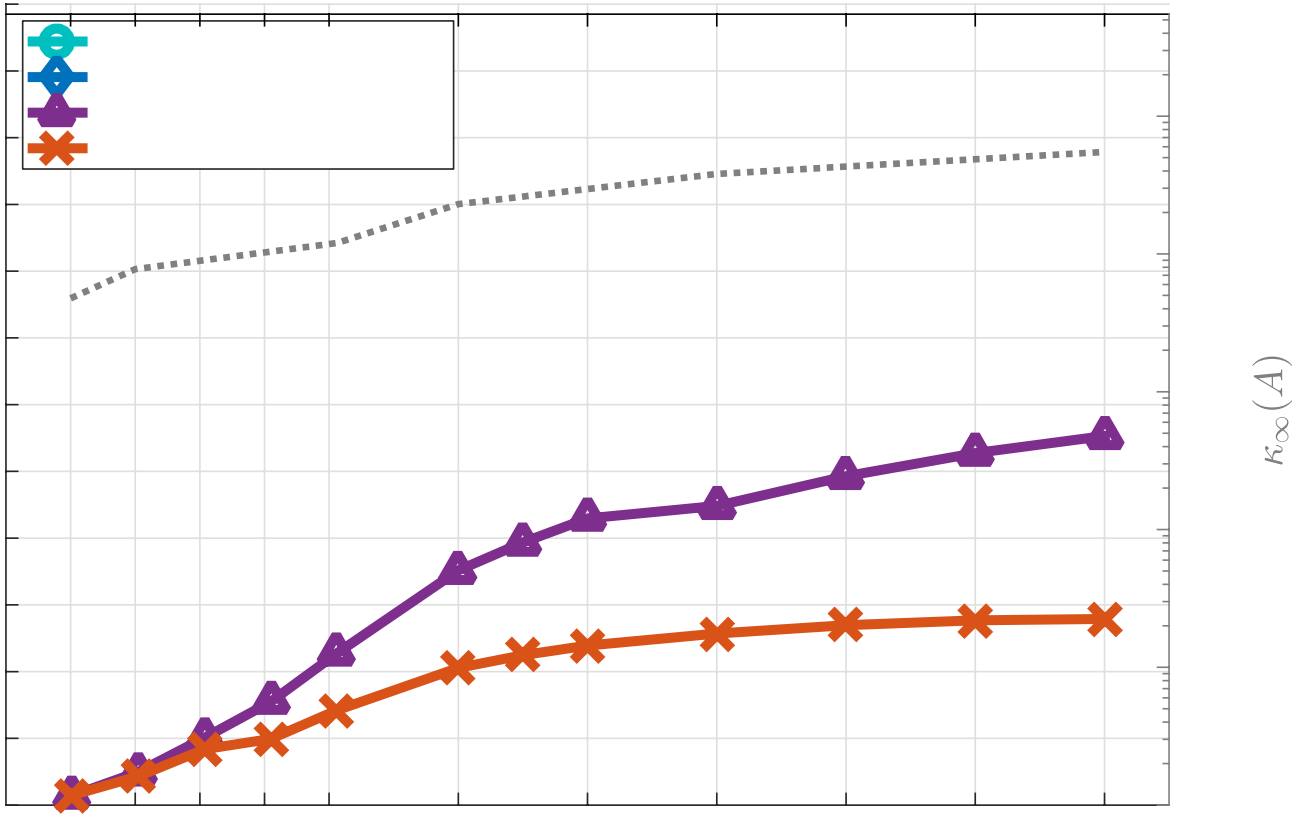


Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using FP64 LU

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

Leveraging Half Precision in HPC on V100
Performance Behavior

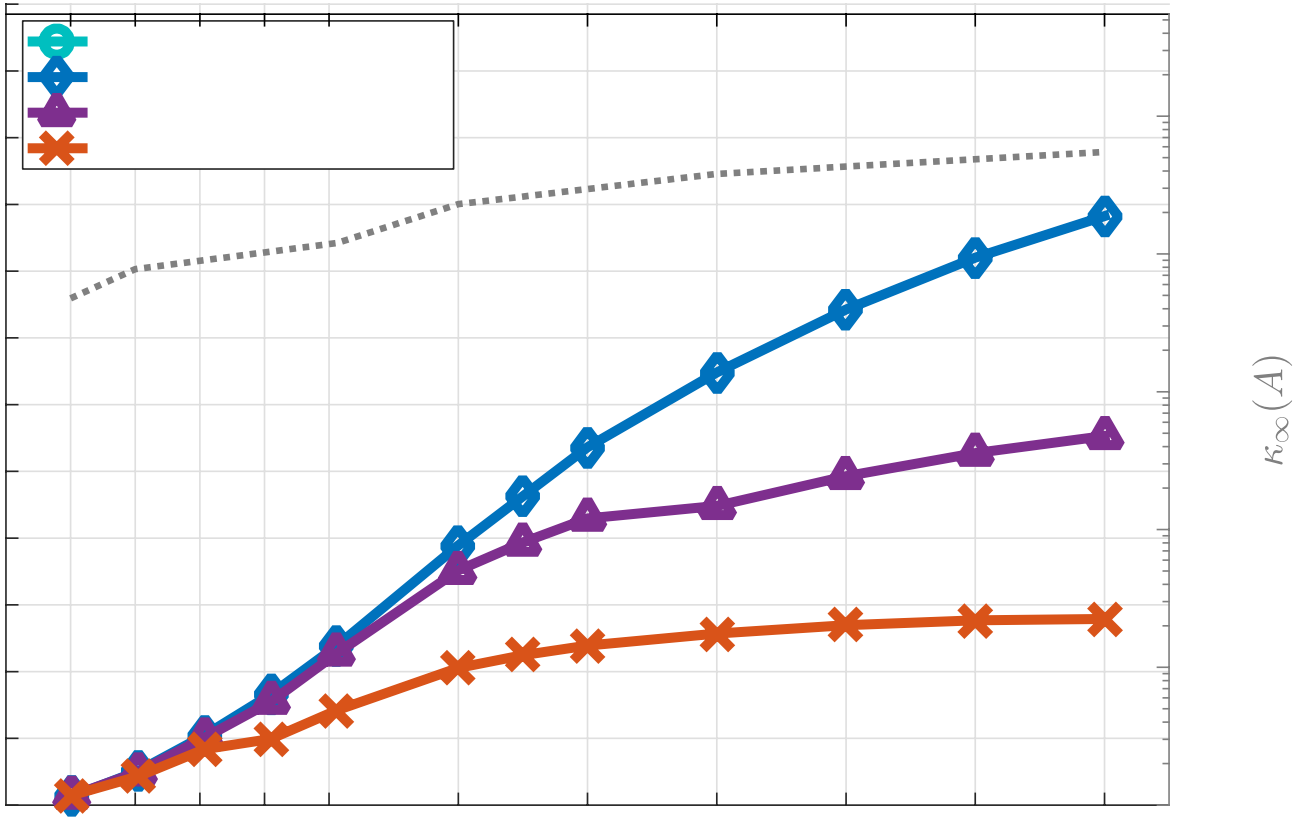


Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- 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 $s_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

Leveraging Half Precision in HPC on V100
Performance Behavior

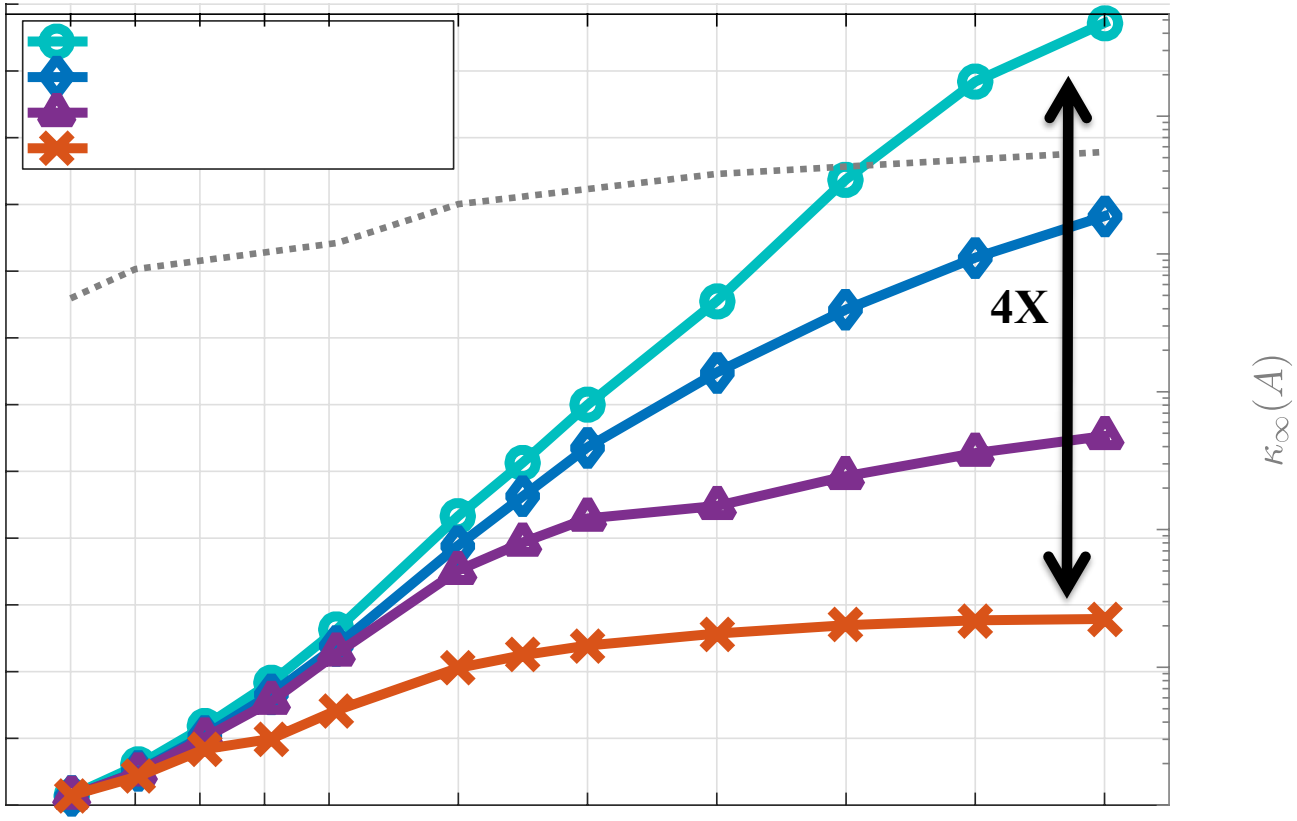


Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- 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 $s_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

Leveraging Half Precision in HPC on V100
Performance Behavior

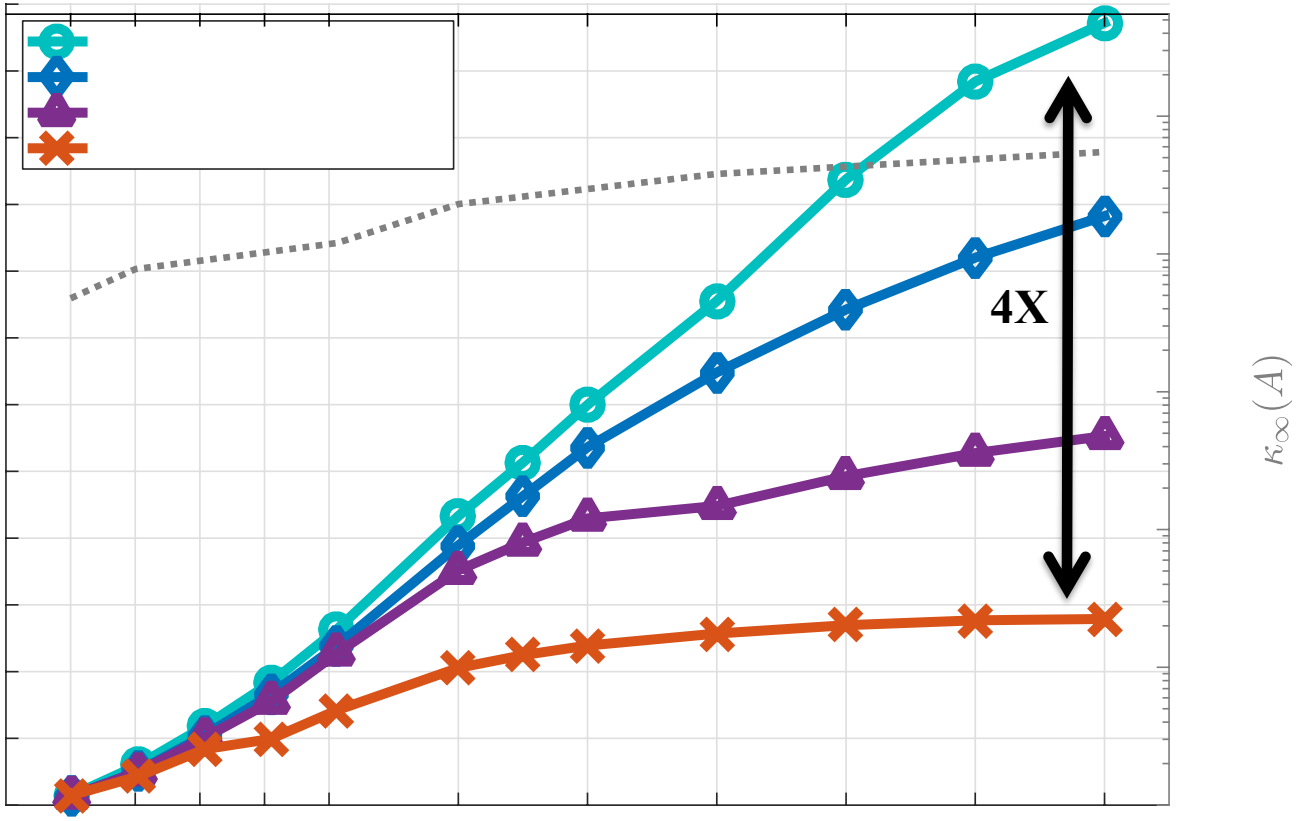


Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- 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 $s_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

Leveraging Half Precision in HPC on V100
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 $s_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{cond})$ and positive eigenvalues.

Leveraging Half Precision in HPC on V100

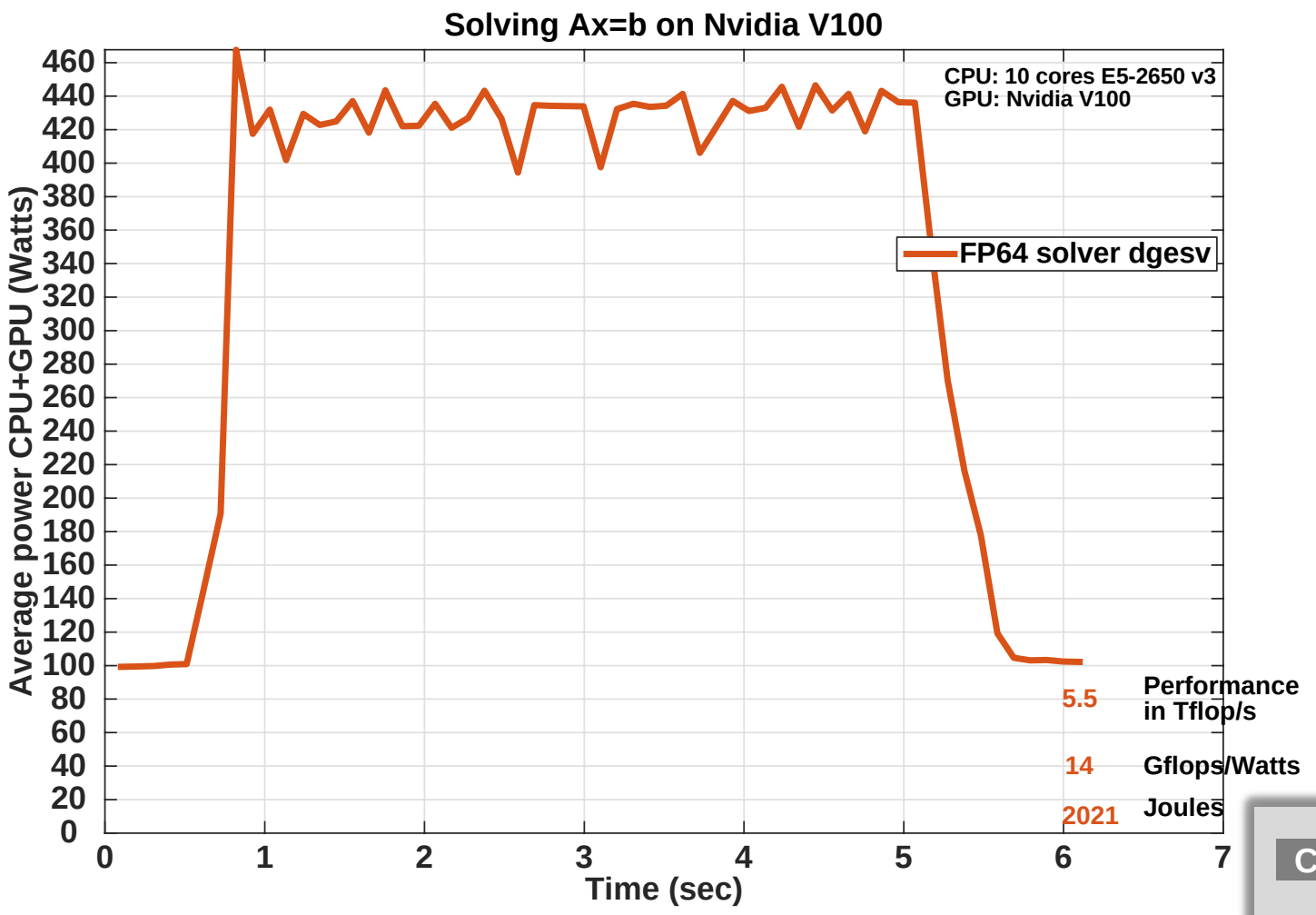
Use Mixed Precision algorithms

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 !!!

Leveraging Half Precision in HPC

Power awareness



- Power consumption of the FP64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 5.5 Tflop/s and requires about 2021 joules providing about 14 Gflops/Watts.

Power is for GPU + CPU + DRAM

CPU Intel Xeon E5-2650 v3 (Haswell) 2x10 cores @ 2.30 GHz

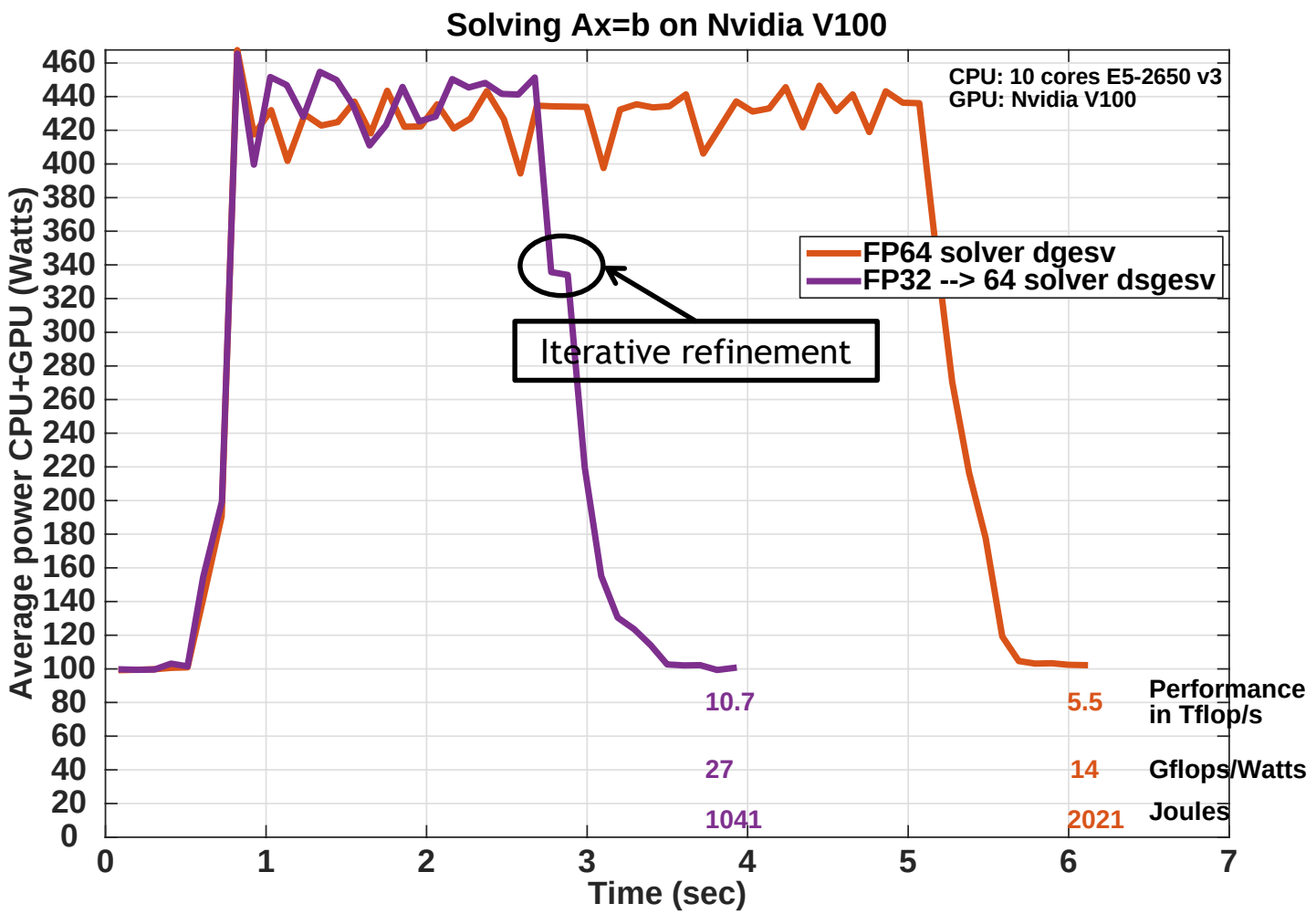
V100 NVIDIA Volta GPU 80 MP x 64 @ 1.38 GHz

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

Leveraging Half Precision in HPC

Power awareness

Mixed precision techniques can provide a large gain in energy efficiency



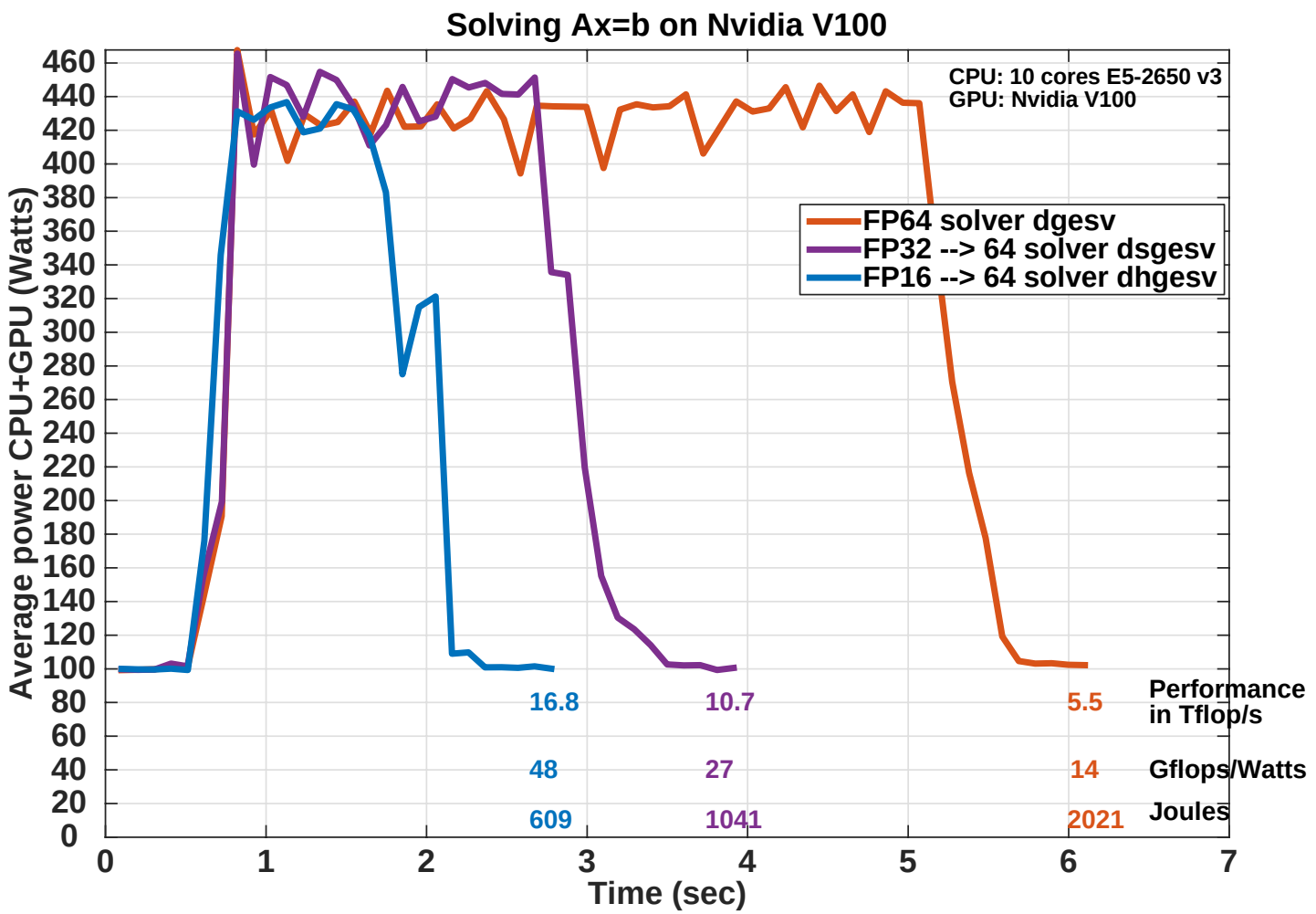
- Power consumption of the FP64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 5.5 Tflop/s and requires about 2021 joules providing about 14 Gflops/Watts.
- Power consumption of the mixed precision FP32→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 10.7 Tflop/s and requires about 1041 joules providing about 30 Gflops/Watts.

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

Leveraging Half Precision in HPC

Power awareness

Mixed precision techniques can provide a large gain in energy efficiency



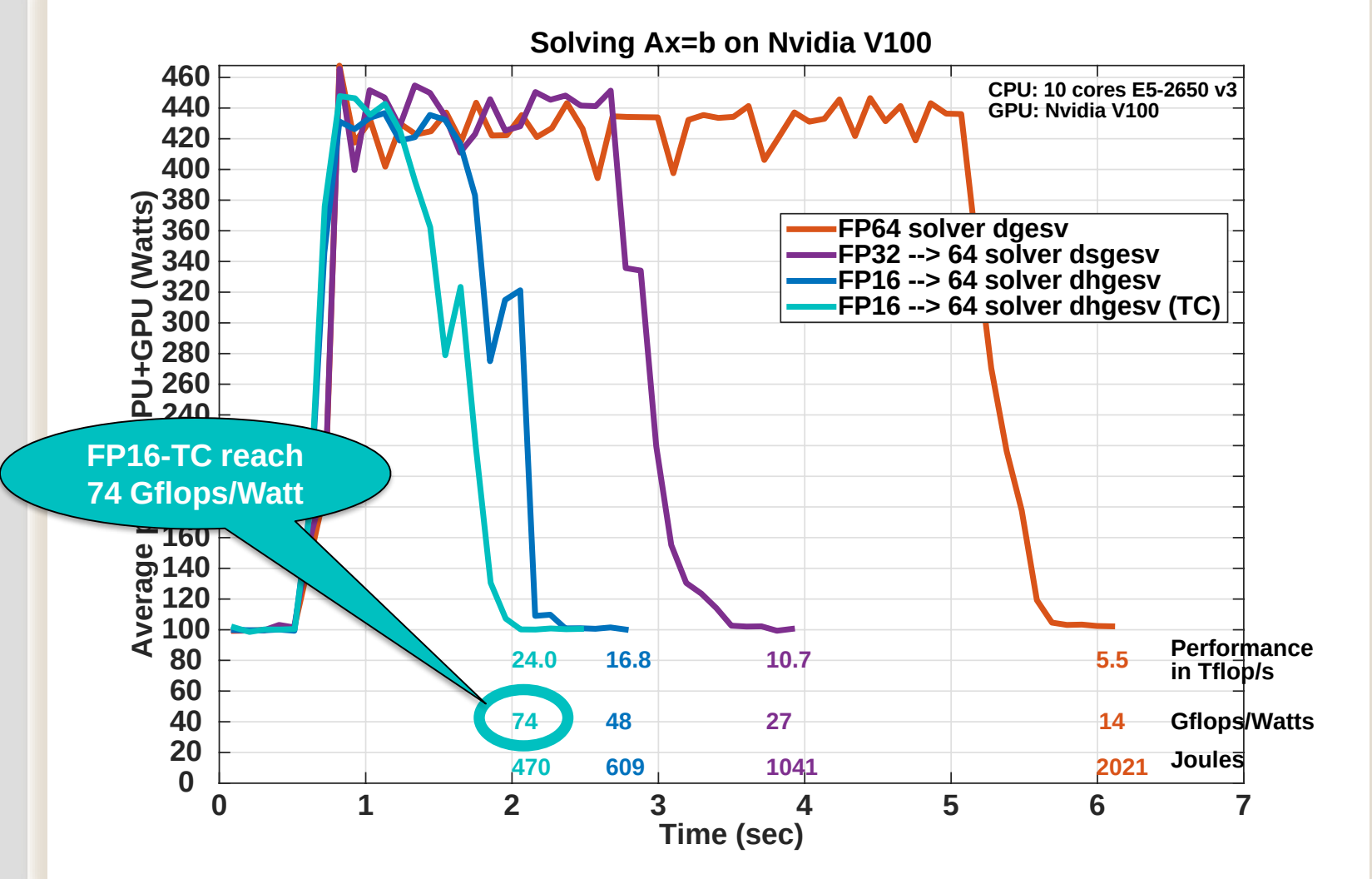
- Power consumption of the FP64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 5.5 Tflop/s and requires about 2021 joules providing about 14 Gflops/Watts.
- Power consumption of the mixed precision FP32→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 10.7 Tflop/s and requires about 1041 joules providing about 30 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 16.8 Tflop/s and requires about 609 joules providing about 48 Gflops/Watts.

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

Leveraging Half Precision in HPC

Power awareness

Mixed precision techniques can provide a large gain in energy efficiency



- Power consumption of the FP64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 5.5 Tflop/s and requires about 2021 joules providing about 14 Gflops/Watts.
- Power consumption of the mixed precision FP32→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 10.7 Tflop/s and requires about 1041 joules providing about 30 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 16.8 Tflop/s and requires about 609 joules providing about 48 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 TC algorithm using Tensor Cores to solve Ax=b for a matrix of size 34K, it achieve 24 Tflop/s and requires about 470 joules providing about 74 Gflops/Watts.

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

Conclusion:

- We accelerated the solution of linear system **Ax = b** solver using hardware-accelerated FP16 arithmetic on GPUs;

- We introduced a framework for exploiting mixed-precision FP16-FP32/FP64 iterative refinement solvers and describe the path to draw high-performance and energy-aware GPU implementations;
  - Ideas can be applied to other 1 sided reductions (LU, LL^T, LDL^T, QR) and also for 2 sided in the case of eigenvalue/vectors. Will find its way into SLATE (ECP LA library effort).

- Our technique shows that a number of problems can be **accelerated** up to **4X** by the usage of the **FP16-TC** or **2X** using the **FP32** arithmetic.

- We studied the energy-efficiency of our approach that showed significant energy savings, **5X energy savings** using the **FP16-TC** compared to the FP64 implementation.
  - We illustrated a technique to use **V100 Tensor Cores FP16-TC** that achieves FP64 accuracy at a highly efficient/accelerated performance equating to **74 Gflops/Watt** and **24 Tflops/s**.

- We have rigorous error analysis to support everything

The New York Times

Three Pioneers in Artificial Intelligence Win Turing Award

From left, Yann LeCun, Geoffrey Hinton and Yoshua Bengio. The researchers worked on key developments for neural networks, which are reshaping how computer systems are built.