MIXED PRECISION LU FACTORIZATION ON GPU TENSOR CORES

Reducing Data Movement and Memory Footprint

Florent Lopez and Theo Mary

SIAM CSE'21

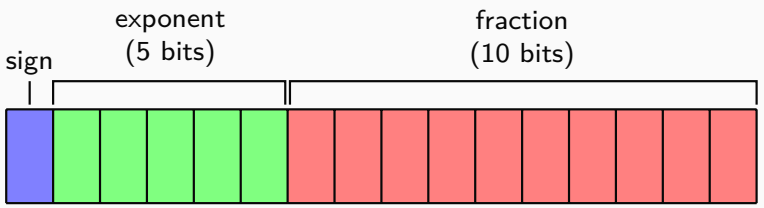
Livermore Software Technology, an ANSYS company

Today's floating-point precision arithmetics

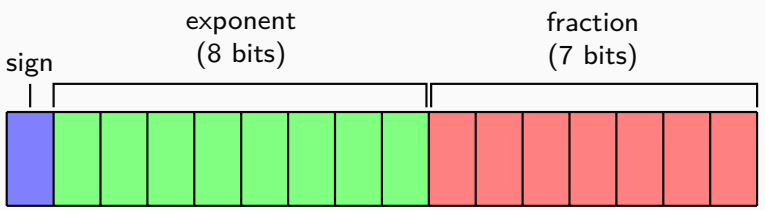
Type		Bits	Range	$u = 2^{-t}$
`fp128`	`quad`	128	$10^{\pm 4932}$	$2^{-113} \approx 1 \times 10^{-34}$
`fp64`	`double`	64	$10^{\pm 308}$	$2^{-53} \approx 1 \times 10^{-16}$
`fp32`	`single`	32	$10^{\pm 38}$	$2^{-24} \approx 6 \times 10^{-8}$
`fp16`	`half`	16	$10^{\pm 5}$	$2^{-11} \approx 5 \times 10^{-4}$
`bfloat16`	`half`	16	$10^{\pm 38}$	$2^{-8} \approx 4 \times 10^{-3}$

Half precision increasingly **supported by hardware**:

- Present: **NVIDIA** Pascal, Volta & Ampere GPUs, **AMD** Radeon Instinct MI25 GPU, **Google** TPU, **ARM** NEON
- Near future: Fujitsu A64FX ARM, **IBM** AI chips, **Intel** Xeon Cooper Lake and Intel Nervana Neural Network



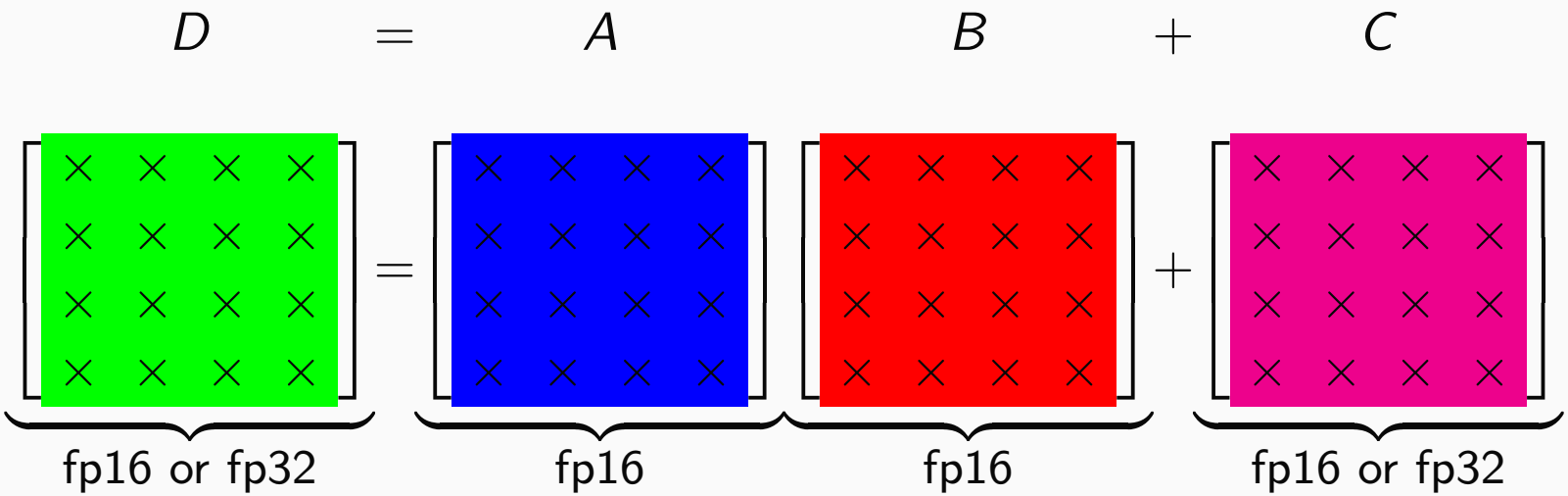
fp16



bfloat16

NVIDIA GPU tensor cores units

Block FMA: 4×4 matrix multiplication **in 1 clock cycle**:

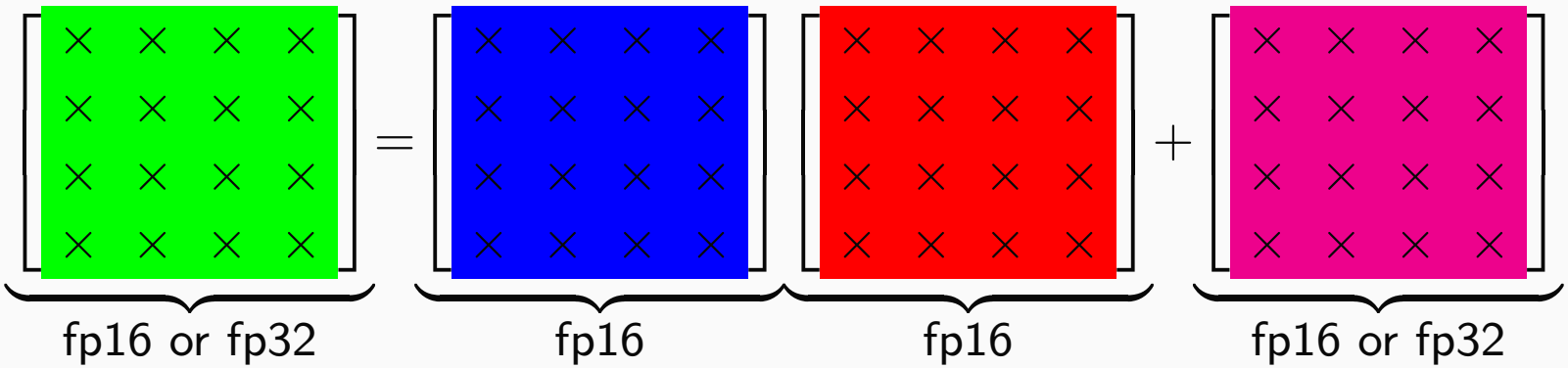


Performance peak: 125 TFlop/s (8× speedup vs fp32)

NVIDIA GPU tensor cores units

Block FMA: 4×4 matrix multiplication **in 1 clock cycle**:

$$D = A \quad B + C$$



Performance peak: 125 TFlop/s (8× speedup vs fp32)
For the matrix product $C = AB$ where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ using tensor cores, the computed $\widehat{C}$ satisfies (Blanchard et. al[1]):

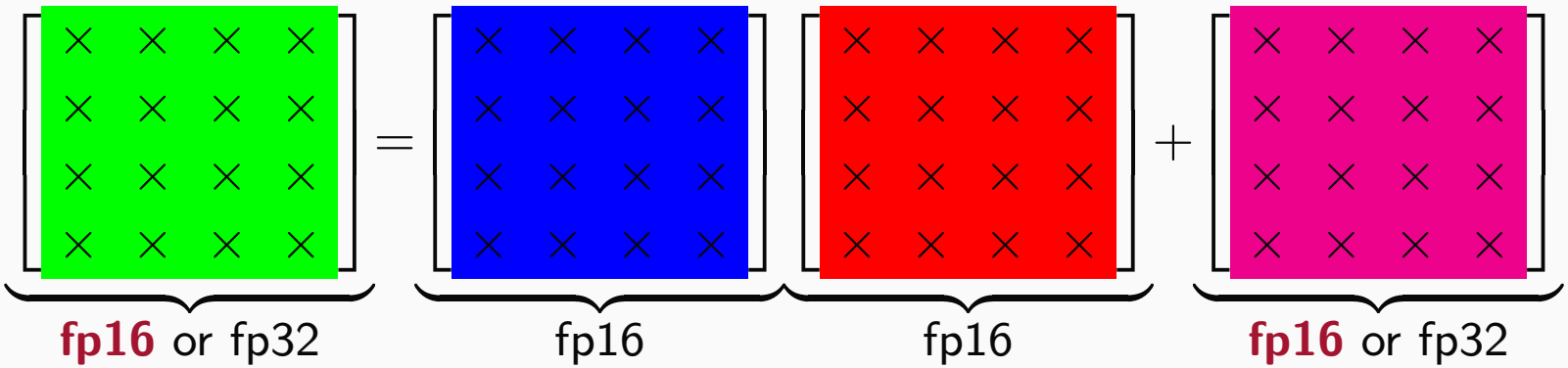
$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \left\{ \right.$$

[1] P. Blanchard, N. J. Higham, F. Lopez, T. Mary and S. Pranesh. *Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores* (SIAM SISC 2020)

NVIDIA GPU tensor cores units

Block FMA: 4×4 matrix multiplication **in 1 clock cycle**:

$$D = A \quad B + C$$



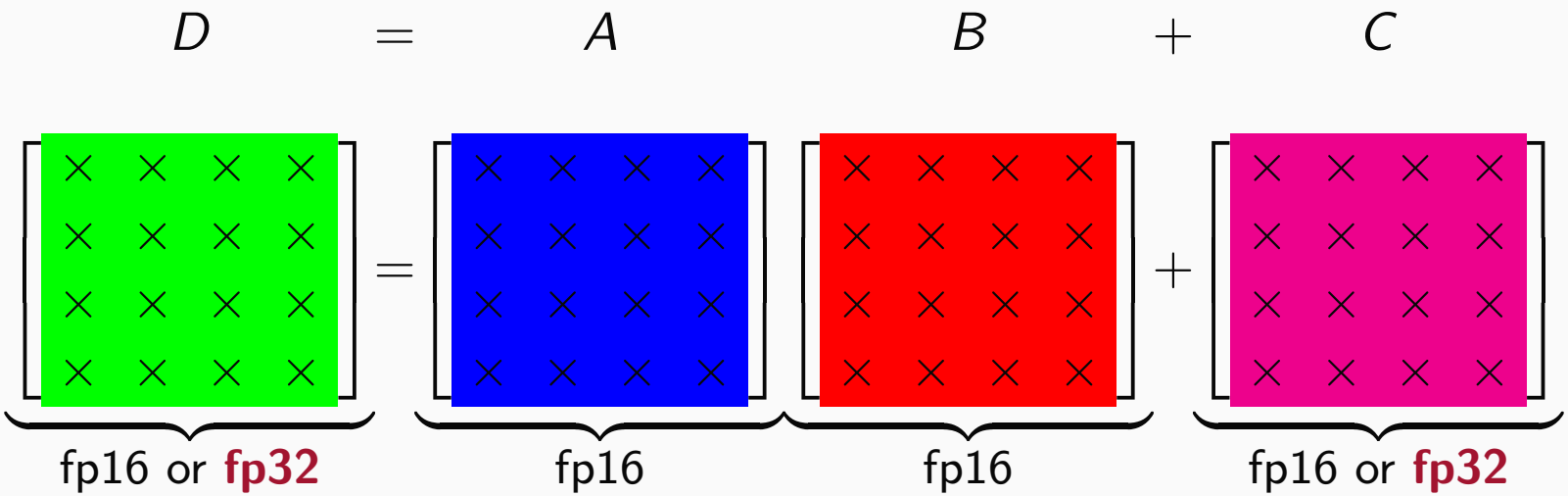
Performance peak: 125 TFlop/s (8× speedup vs fp32)
For the matrix product $C = AB$ where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ using tensor cores, the computed $\widehat{C}$ satisfies (Blanchard et. al[1]):

$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \begin{cases} nu_{16} & \text{(TC16)} \end{cases}$$

[1] P. Blanchard, N. J. Higham, F. Lopez, T. Mary and S. Pranesh. *Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores* (SIAM SISC 2020)

NVIDIA GPU tensor cores units

Block FMA: 4×4 matrix multiplication **in 1 clock cycle**:



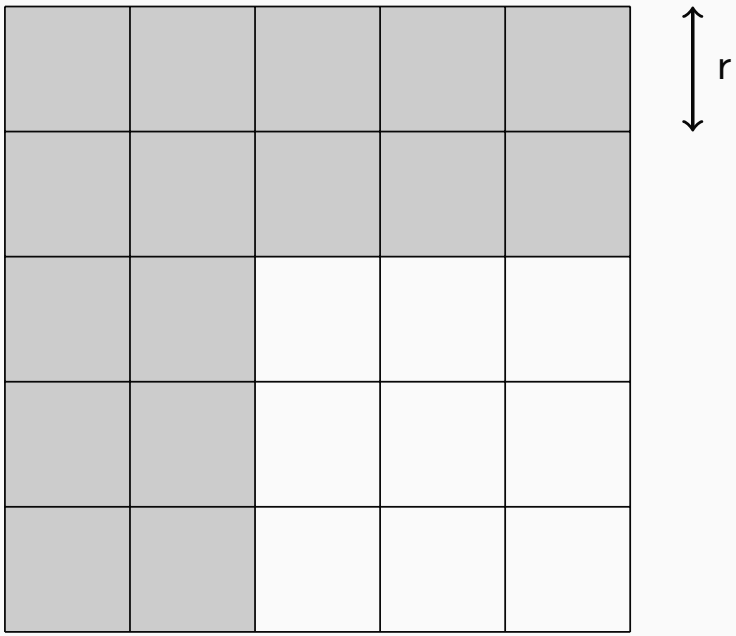
Performance peak: 125 TFlop/s (8× speedup vs fp32)
For the matrix product $C = AB$ where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ using tensor cores, the computed $\widehat{C}$ satisfies (Blanchard et. al[1]):

$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \begin{cases} nu_{16} & \text{(TC16)} \\ 2u_{16} + nu_{32} & \text{(TC32)} \end{cases}$$

[1] P. Blanchard, N. J. Higham, F. Lopez, T. Mary and S. Pranesh. *Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores* (SIAM SISC 2020)

Partitioned LU factorization: right-looking variant

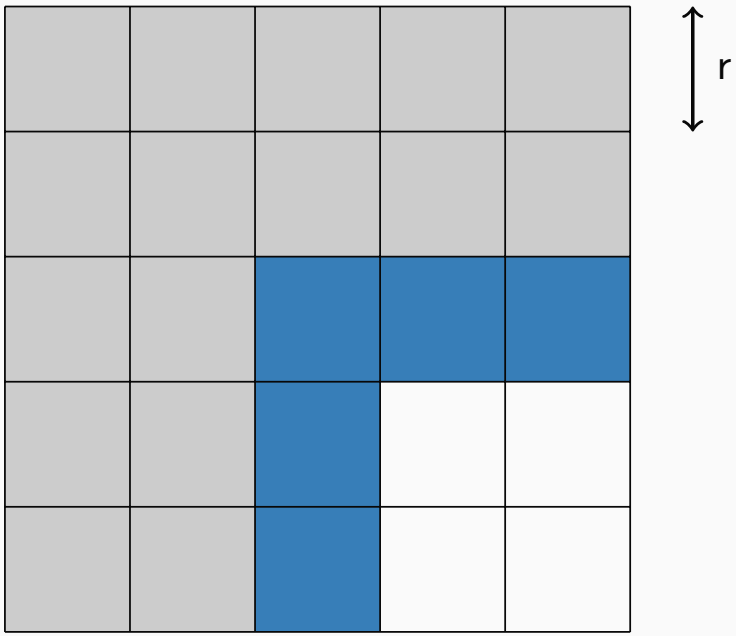
This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



for $k = 1 : q$ **do**
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : q$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
  for $i = k + 1 : q$ **do**
    for $j = k + 1 : q$ **do**
      Update $A_{ij} \leftarrow A_{ij} - L_{ik} U_{kj}$.
    end for
  end for
end for

Partitioned LU factorization: right-looking variant

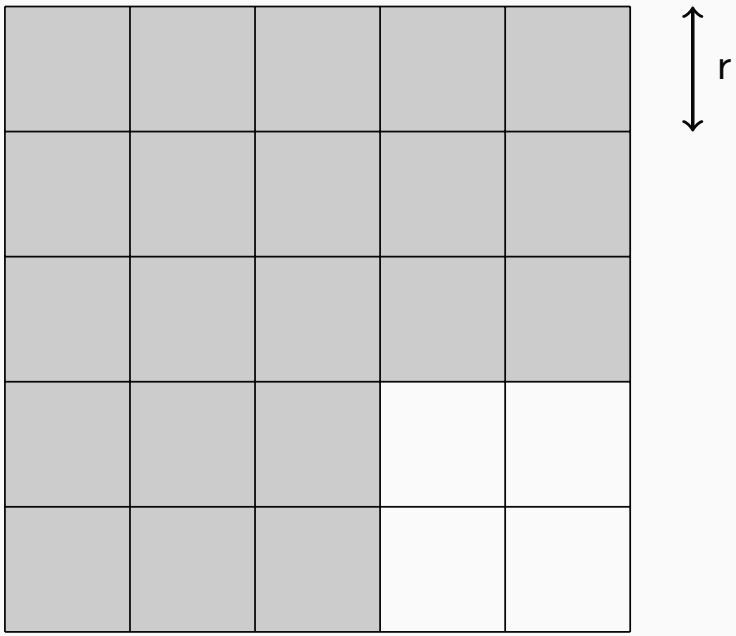
This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



for $k = 1 : q$ **do**
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : q$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
  for $i = k + 1 : q$ **do**
    for $j = k + 1 : q$ **do**
      Update $A_{ij} \leftarrow A_{ij} - L_{ik} U_{kj}$.
    end for
  end for
end for

Partitioned LU factorization: right-looking variant

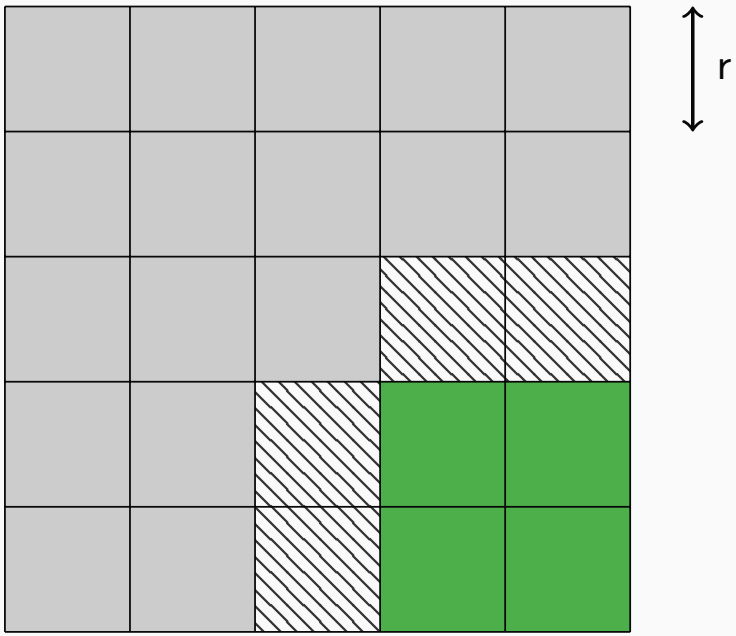
This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



for $k = 1 : q$ **do**
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : q$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
  for $i = k + 1 : q$ **do**
    for $j = k + 1 : q$ **do**
      Update $A_{ij} \leftarrow A_{ij} - L_{ik} U_{kj}$.
    end for
  end for
end for

Partitioned LU factorization: right-looking variant

This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



for $k = 1 : q$ **do**
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : q$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
  for $i = k + 1 : q$ **do**
    for $j = k + 1 : q$ **do**
      Update $A_{ij} \leftarrow A_{ij} - L_{ik} U_{kj}$.
    end for
  end for
end for

State-of-the-art mixed precision LU factorization

Mixed precision LU factorization algorithm proposed in Haidar et al [2]:

for $k = 1 : n/r$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$ (in precision u_{32}).
  for $i = k + 1 : n/r$ **do**
    Solve $L_{ik}U_{kk} = A_{ik}$ for L_{ik} (in precision u_{32}).
    Solve $L_{kk}U_{ki} = A_{ki}$ for U_{ki} (in precision u_{32}).
  end for
  for $i = k + 1 : n/r$ **do**
    for $j = k + 1 : n/r$ **do**

      Update $A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$
    end for
  end for
end for

[2] 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'2018)

State-of-the-art mixed precision LU factorization

Mixed precision LU factorization algorithm proposed in Haidar et al [2]:

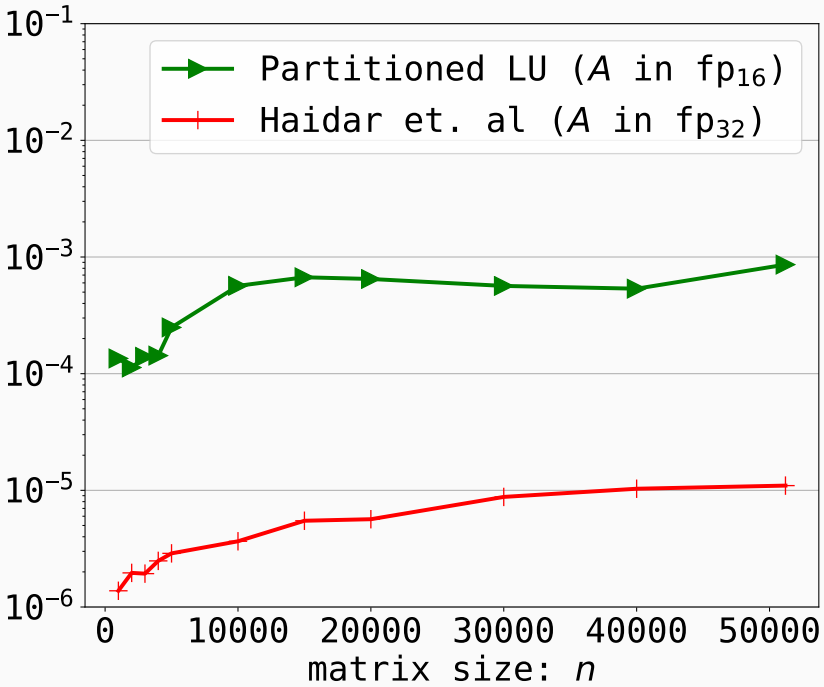
for $k = 1 : n/r$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$ (in precision u_{32}).
  for $i = k+1 : n/r$ **do**
    Solve $L_{ik}U_{kk} = A_{ik}$ for L_{ik} (in precision u_{32}).
    Solve $L_{kk}U_{ki} = A_{ki}$ for U_{ki} (in precision u_{32}).
  end for
  for $i = k+1 : n/r$ **do**
    for $j = k+1 : n/r$ **do**
      Convert to fp16: $\widetilde{L}_{ik} \leftarrow \mathrm{fl}_{16}(L_{ik})$ and $\widetilde{U}_{ki} \leftarrow \mathrm{fl}_{16}(U_{ki})$.
      Update $A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$ using tensor cores (TC32).
    end for
  end for
end for

[2] 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'2018)

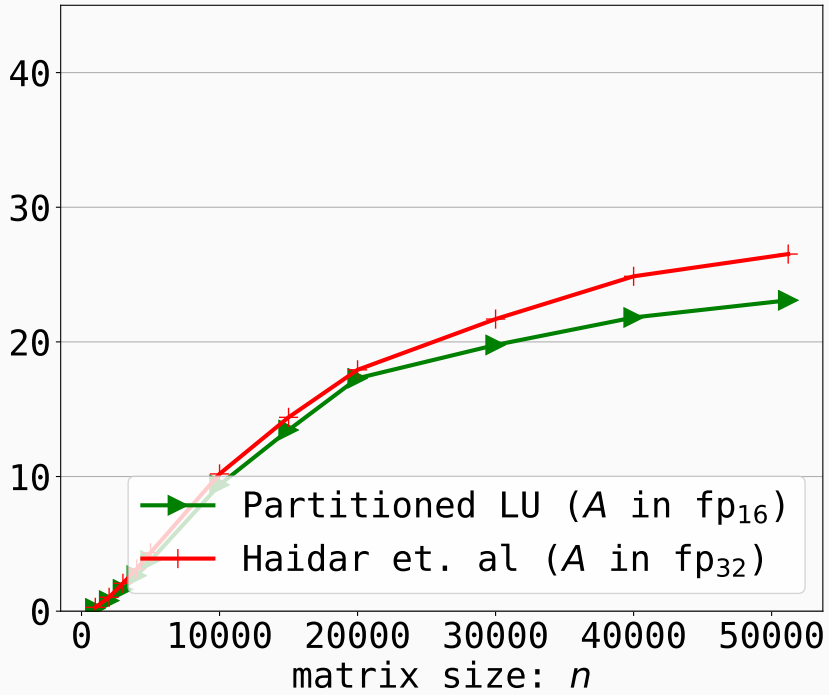
State-of-the-art mixed precision LU factorization

Backward error bounds from Blanchard et. al

Partitioned LU $u = u_{16}$	Haidar et. al $u = u_{32}$
nu_{16}	$2u_{16} + nu_{32}$



Backward error



Performance (TFlop/s)

Reduce memory footprint and data movement

Mixed precision LU factorization algorithm from Haidar et al:

- ▲ Dramatically more accurate compared to fp16 LU factorization
- ▼ Only marginally faster compared to half precision factorization algorithm
  $\Rightarrow$ poor exploitation of the tensor cores
- ▼ No reduction in memory footprint compared to full precision algorithm

Reduce memory footprint and data movement

Mixed precision LU factorization algorithm from Haidar et al:

- ▲ Dramatically more accurate compared to fp16 LU factorization
- ▼ Only marginally faster compared to half precision factorization algorithm
  $\Rightarrow$ poor exploitation of the tensor cores
- ▼ No reduction in memory footprint compared to full precision algorithm

Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Reduce memory footprint and data movement

Mixed precision LU factorization algorithm from Haidar et al:

- ▲ Dramatically more accurate compared to fp16 LU factorization
- ▼ Only marginally faster compared to half precision factorization algorithm
  $\Rightarrow$ poor exploitation of the tensor cores
- ▼ No reduction in memory footprint compared to full precision algorithm

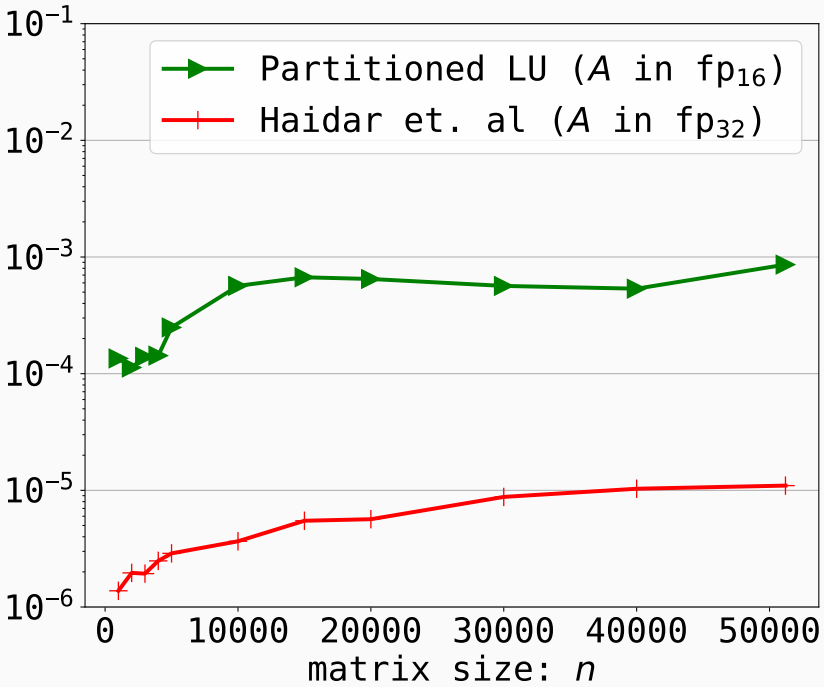
Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Idea: Store A in precision u_{16} and perform panel factorization and tensor core updates in precision u_{16} (TC16) in Haidar's algorithm

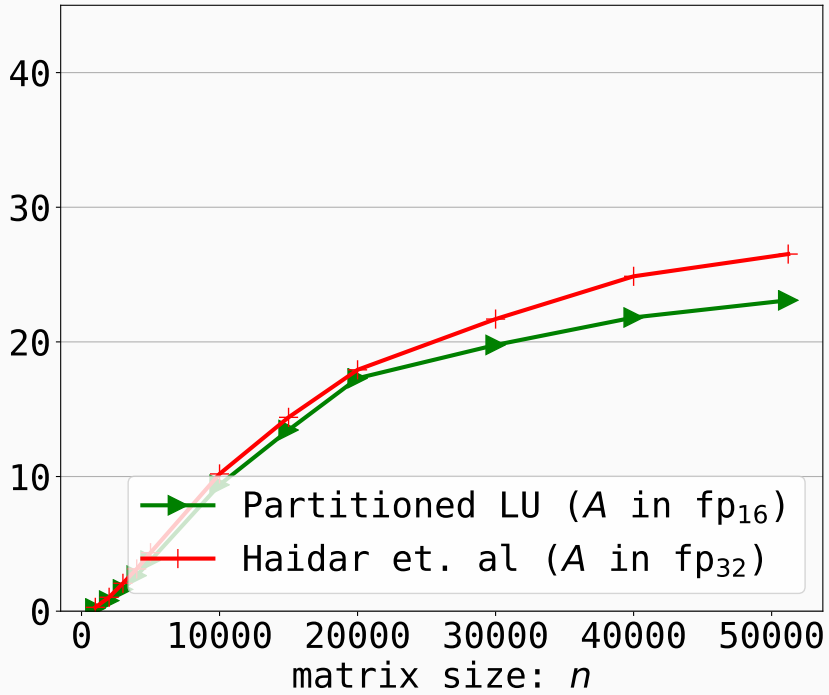
State-of-the-art mixed precision LU factorization

Backward error bounds

Partitioned LU $u = u_{16}$	Haidar et. al $u = u_{32}$	Haidar et. al $u = u_{16}$
nu_{16}	$2u_{16} + nu_{32}$	



Backward error

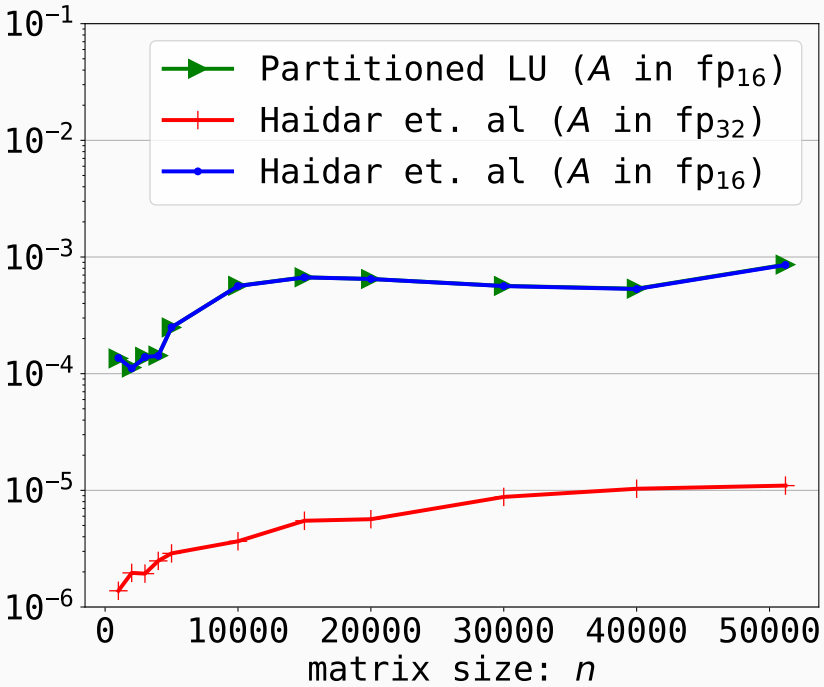


Performance (TFlop/s)

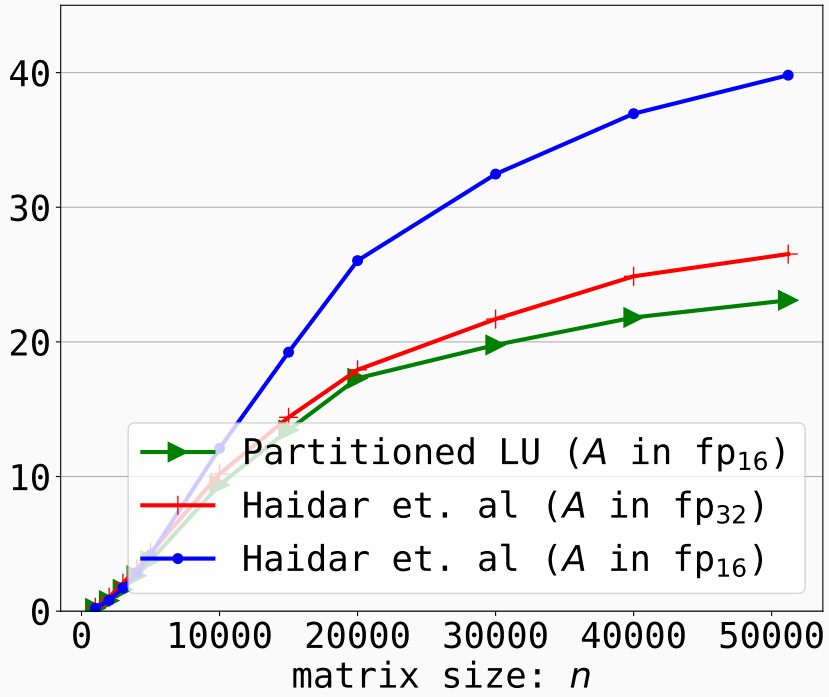
State-of-the-art mixed precision LU factorization

Backward error bounds

Partitioned LU $u = u_{16}$	Haidar et. al $u = u_{32}$	Haidar et. al $u = u_{16}$
nu_{16}	$2u_{16} + nu_{32}$	$(n/4)u_{16}$



Backward error



Performance (TFlop/s)

Storing A in fp16

Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Storing A in fp16

Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Idea: introduce temporary fp32 buffers and use them to accumulate update operations i.e. Replace

$$A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$$

Storing A in fp16

Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Idea: introduce temporary fp32 buffers and use them to accumulate update operations i.e. Replace

$$A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$$

with

$$\begin{aligned} B_{ij} &= \mathrm{fl}_{32}(A_{ij}) \\ B_{ij} &\leftarrow B_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj} \\ A_{ij} &\leftarrow \mathrm{fl}_{16}(B_{ij}) \end{aligned}$$

Storing A in fp16

Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Idea: introduce temporary fp32 buffers and use them to accumulate update operations i.e. Replace

$$A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$$

with

$$\begin{aligned} B_{ij} &= \mathrm{fl}_{32}(A_{ij}) \\ B_{ij} &\leftarrow B_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj} \\ A_{ij} &\leftarrow \mathrm{fl}_{16}(B_{ij}) \end{aligned}$$

Problem: In Haidar's algorithm, we need a f32 buffer corresponding to the trailing sub-matrix which, in the first few steps of the factorization is almost as large as the matrix itself.

Storing A in fp16

Objective: Improve the performance and reduce memory footprint mixed precision LU factorization without significant loss of accuracy

Idea: introduce temporary fp32 buffers and use them to accumulate update operations i.e. Replace

$$A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$$

with

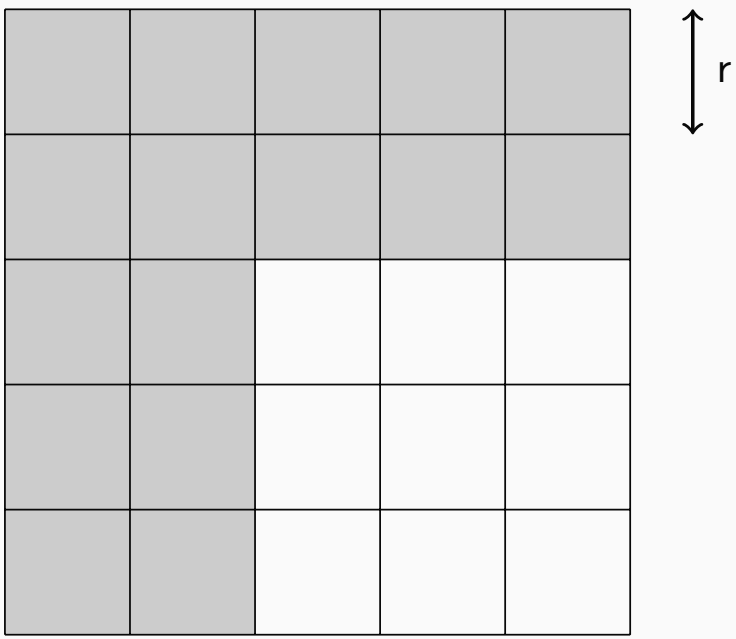
$$\begin{aligned} B_{ij} &= \mathrm{fl}_{32}(A_{ij}) \\ B_{ij} &\leftarrow B_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj} \\ A_{ij} &\leftarrow \mathrm{fl}_{16}(B_{ij}) \end{aligned}$$

Problem: In Haidar's algorithm, we need a f32 buffer corresponding to the trailing sub-matrix which, in the first few steps of the factorization is almost as large as the matrix itself.

$\Rightarrow$ Switch to a left-looking factorization.

Partitioned LU factorization: left-looking variant

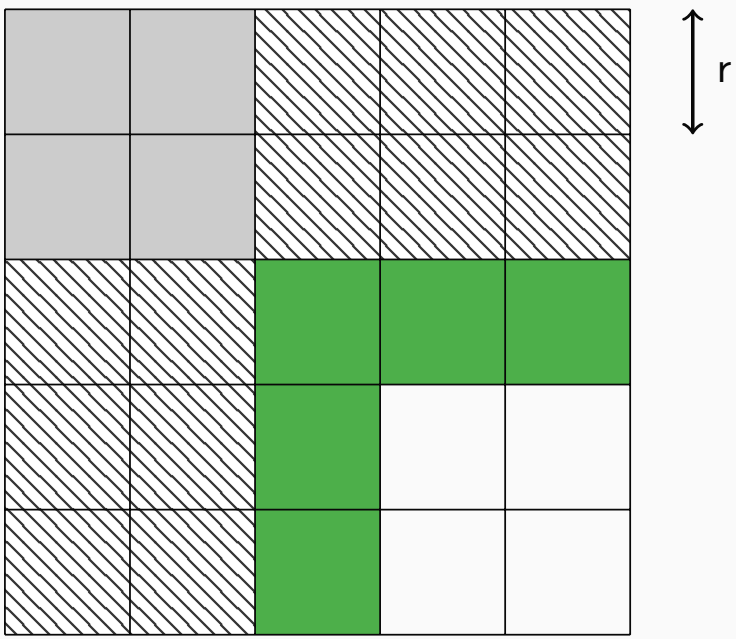
Left-looking LU factorization: at every step, the current panel is updated w.r.t the already computed factors (to the left) before being factored.



for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    for $j = 1 : k - 1$ **do**
      Update $A_{ik} \leftarrow A_{ik} - L_{ij} U_{jk}$.
      Update $A_{ki} \leftarrow A_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : n/r$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
end for

Partitioned LU factorization: left-looking variant

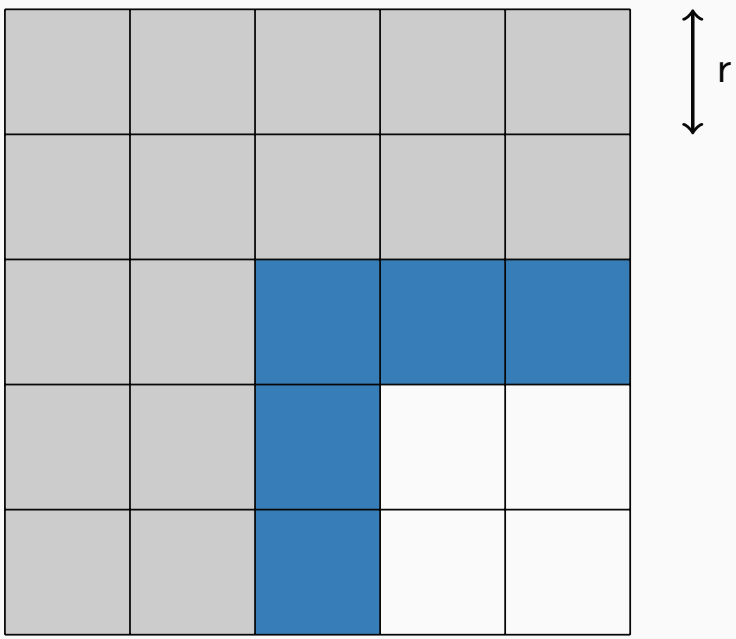
Left-looking LU factorization: at every step, the current panel is updated w.r.t the already computed factors (to the left) before being factored.



for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    for $j = 1 : k - 1$ **do**
      Update $A_{ik} \leftarrow A_{ik} - L_{ij} U_{jk}$.
      Update $A_{ki} \leftarrow A_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : n/r$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
end for

Partitioned LU factorization: left-looking variant

Left-looking LU factorization: at every step, the current panel is updated w.r.t the already computed factors (to the left) before being factored.



for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    for $j = 1 : k - 1$ **do**
      Update $A_{ik} \leftarrow A_{ik} - L_{ij} U_{jk}$.
      Update $A_{ki} \leftarrow A_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Factorize $L_{kk} U_{kk} = A_{kk}$.
  for $i = k + 1 : n/r$ **do**
    Solve $L_{ik} U_{kk} = A_{ik}$ for L_{ik}.
    Solve $L_{kk} U_{ki} = A_{ki}$ for U_{ki}.
  end for
end for

New left-looking mixed-precision LU factorization

A is now stored in precision u_{16}.

for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**

    for $j = 1 : k-1$ **do**
      Update $A_{ik} \leftarrow A_{ik} - \widetilde{L}_{ij}\widetilde{U}_{jk}$
      Update $A_{ki} \leftarrow A_{ki} - \widetilde{L}_{kj}\widetilde{U}_{ji}$
    end for
  end for
  Factorize $\widetilde{L}_{kk}\widetilde{U}_{kk} = A_{kk}$ (in precision u_{16}).
  for $i = k+1 : n/r$ **do**
    Solve $\widetilde{L}_{ik}\widetilde{U}_{kk} = A_{ik}$ for L_{ik} (in precision u_{16}).
    Solve $\widetilde{L}_{kk}\widetilde{U}_{ki} = A_{ki}$ for U_{ki} (in precision u_{16}).
  end for

end for

New left-looking mixed-precision LU factorization

A is now stored in precision u_{16}. Using tensor cores $\Rightarrow$ TC16.

for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**

    for $j = 1 : k-1$ **do**
      Update $A_{ik} \leftarrow A_{ik} - \widetilde{L}_{ij}\widetilde{U}_{jk}$ using tensor cores.
      Update $A_{ki} \leftarrow A_{ki} - \widetilde{L}_{kj}\widetilde{U}_{ji}$ using tensor cores.
    end for
  end for
  Factorize $\widetilde{L}_{kk}\widetilde{U}_{kk} = A_{kk}$ (in precision u_{16}).
  for $i = k+1 : n/r$ **do**
    Solve $\widetilde{L}_{ik}\widetilde{U}_{kk} = A_{ik}$ for L_{ik} (in precision u_{16}).
    Solve $\widetilde{L}_{kk}\widetilde{U}_{ki} = A_{ki}$ for U_{ki} (in precision u_{16}).
  end for

end for

New left-looking mixed-precision LU factorization

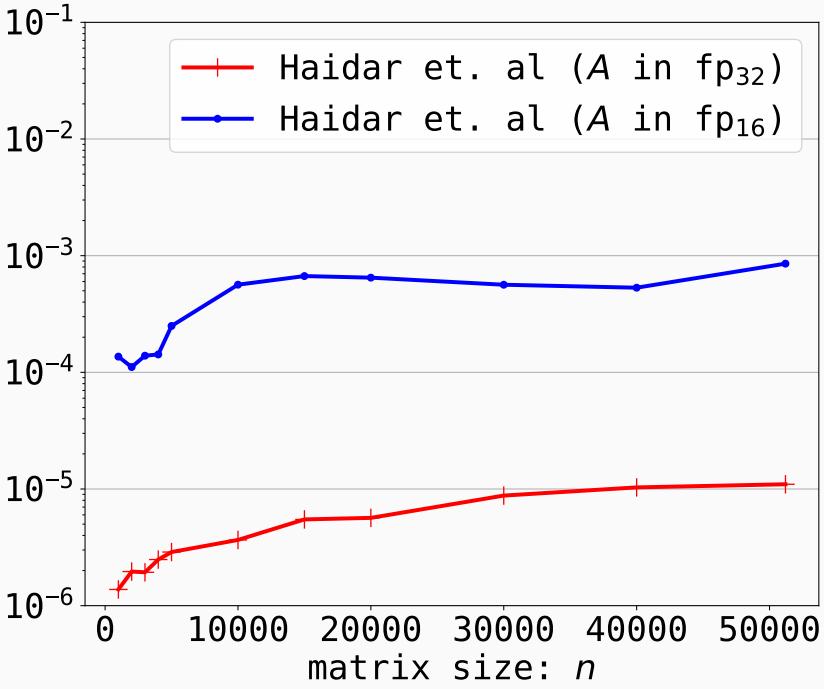
A is now stored in precision u_{16}. Using tensor cores + fp32 buffer $\Rightarrow$ TC32

for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32: $B_{ik} = \mathrm{fl}_{32}(A_{ik})$ and $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k-1$ **do**
      Update $B_{ik} \leftarrow B_{ik} - \widetilde{L}_{ij}\widetilde{U}_{jk}$ using tensor cores.
      Update $B_{ki} \leftarrow B_{ki} - \widetilde{L}_{kj}\widetilde{U}_{ji}$ using tensor cores.
    end for
  end for
  Factorize $L_{kk}U_{kk} = B_{kk}$ (in precision u_{32}).
  for $i = k+1 : n/r$ **do**
    Solve $L_{ik}U_{kk} = B_{ik}$ for L_{ik} (in precision u_{32}).
    Solve $L_{kk}U_{ki} = B_{ki}$ for U_{ki} (in precision u_{32}).
  end for
  for $i = k : n/r$ **do**
    Convert $\widetilde{L}_{ik} \leftarrow \mathrm{fl}_{16}(L_{ik})$ and $\widetilde{U}_{ki} \leftarrow \mathrm{fl}_{16}(U_{ki})$.
  end for
end for

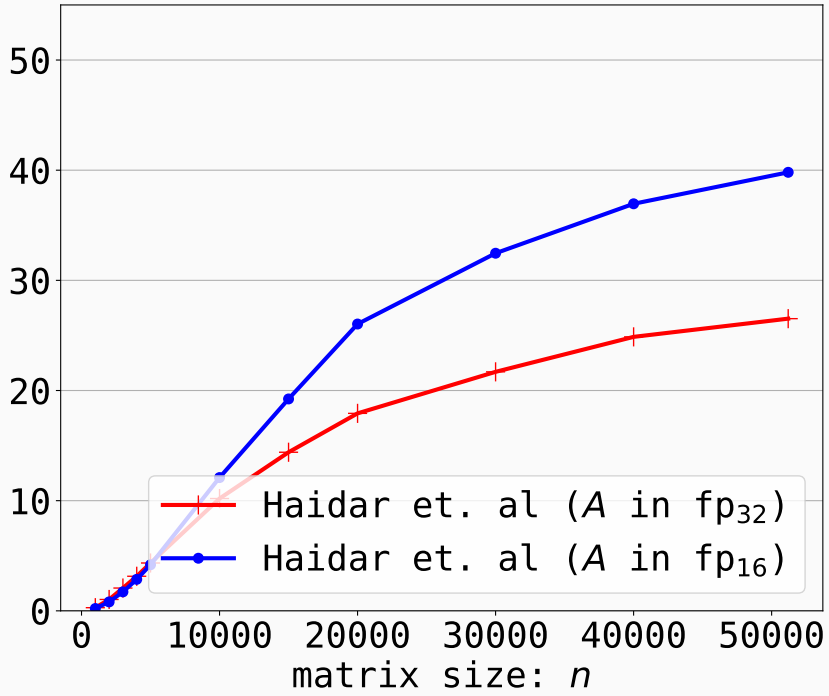
New left-looking mixed-precision LU factorization

Backward error bounds

Haidar et. al $u = u_{32}$	Haidar et. al $u = u_{16}$	New left-looking $u = u_{16}$
$2u_{16} + nu_{32}$	$(n/4)u_{16}$	



Backward error

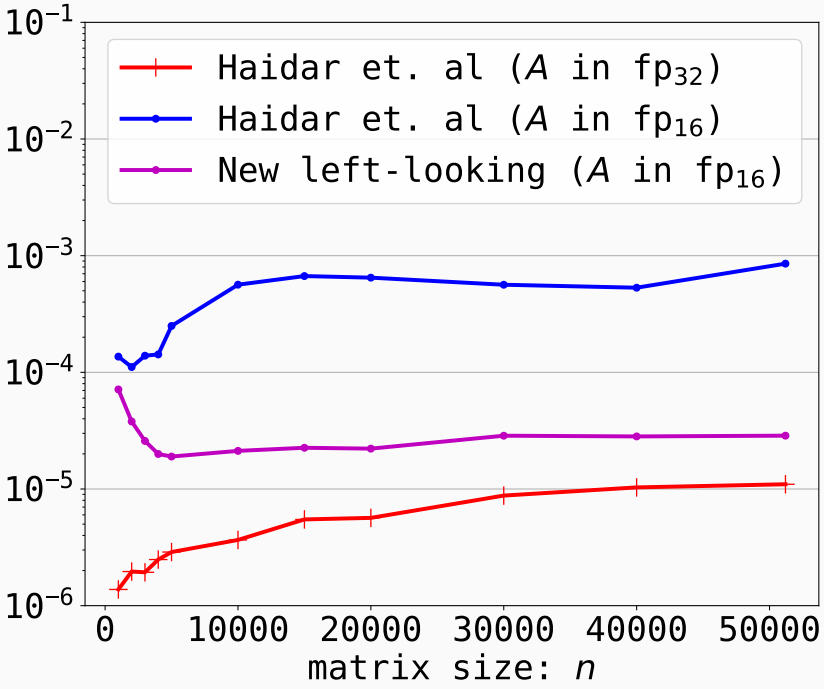


Performance (TFlop/s)

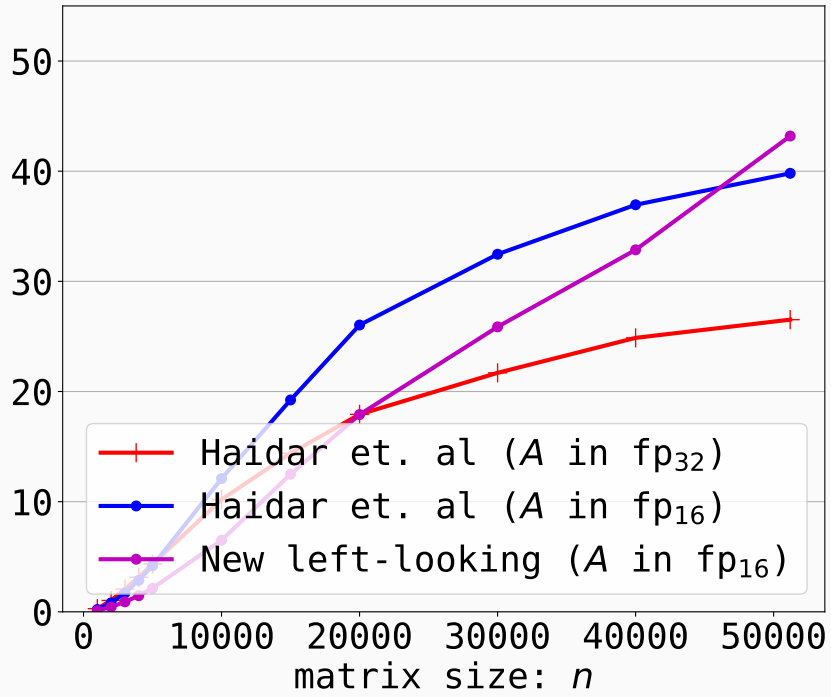
New left-looking mixed-precision LU factorization

Backward error bounds

Haidar et. al $u = u_{32}$	Haidar et. al $u = u_{16}$	New left-looking $u = u_{16}$
$2u_{16} + nu_{32}$	$(n/4)u_{16}$	$2u_{16} + nu_{32}$



Backward error

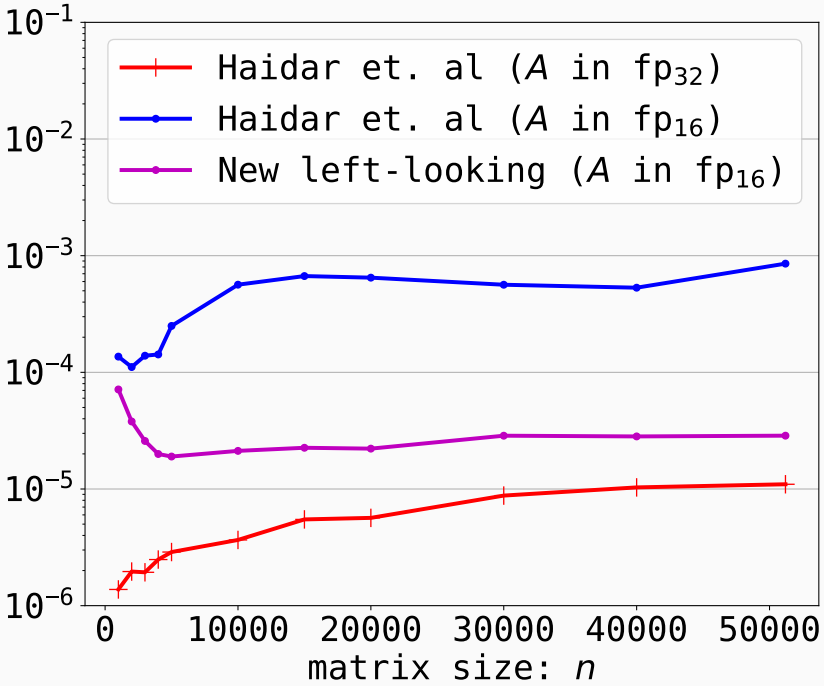


Performance (TFlop/s)

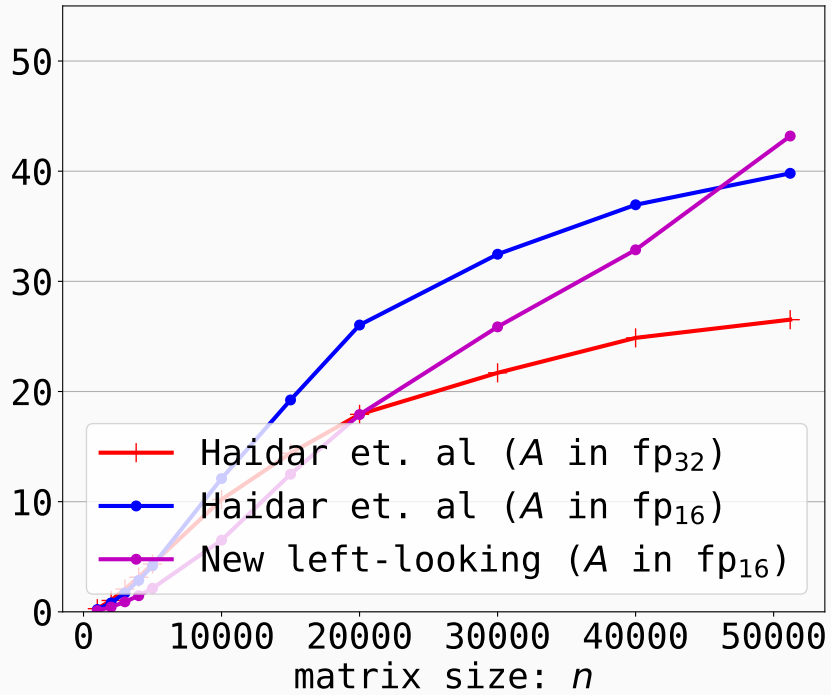
New left-looking mixed-precision LU factorization

fp32 storage (# of entries), r = 256

Haidar et. al $u = u_{32}$	Haidar et. al $u = u_{16}$	New left-looking $u = u_{16}$
n^2	0	nr



Backward error



Performance (TFlop/s)

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm.
- ▲ Similar accuracy.
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm.
- ▲ Similar accuracy.
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

Problems:

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm.
- ▲ Similar accuracy.
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

Problems:

- Performing the panel factorization in precision u_{32} rather than u_{16} slows down the execution.

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm.
- ▲ Similar accuracy.
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

Problems:

- Performing the panel factorization in precision u_{32} rather than u_{16} slows down the execution.
- We do not take advantage of tensor cores in the panel factorization.

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm.
- ▲ Similar accuracy.
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

Problems:

- Performing the panel factorization in precision u_{32} rather than u_{16} slows down the execution.
- We do not take advantage of tensor cores in the panel factorization.

Idea: Keep the panel in precision u_{16} (i.e convert the fp32 buffer to fp16 before the panel factorization) and perform the panel factorization using fp16 or TC16 arithmetic.

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm.
- ▲ Similar accuracy.
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

Problems:

- Performing the panel factorization in precision u_{32} rather than u_{16} slows down the execution.
- We do not take advantage of tensor cores in the panel factorization.

Idea: Keep the panel in precision u_{16} (i.e convert the fp32 buffer to fp16 before the panel factorization) and perform the panel factorization using fp16 or TC16 arithmetic.

- ▼ Error bound varies from $2u_{16} + nu_{32}$ to $u_{16} + \max(nu_{32}, ru_{16})$.
  With $r = 256$, we have $nu_{32} \geq ru_{16}$ for $n \gtrsim 2.1 \times 10^6 \Rightarrow$ in practice, especially on single GPU kernel, ru_{16} dominates the error.

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm
- ▲ Similar accuracy
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

Problems:

- Performing the panel factorization in precision u_{32} rather than u_{16} slows down the execution.
- We do not take advantage of tensor cores in the panel factorization.

Idea:

New left-looking mixed-precision LU factorization

New left-looking mixed precision LU factorization algorithm:

- ▲ Considerably faster than state-of-the-art algorithm
- ▲ Similar accuracy
- ▲ A stored in fp16.
- ▼ Slower on smaller matrices. Can we do better?

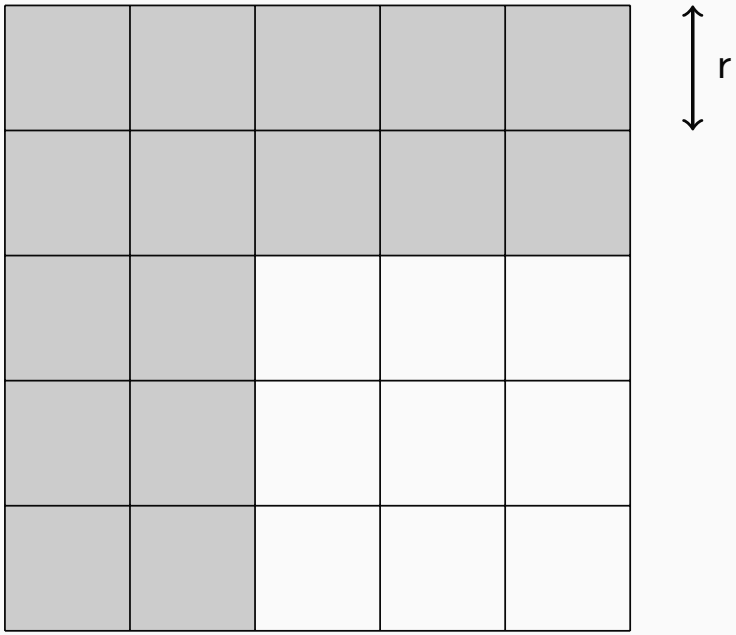
Problems:

- Performing the panel factorization in precision u_{32} rather than u_{16} slows down the execution.
- We do not take advantage of tensor cores in the panel factorization.

Idea:

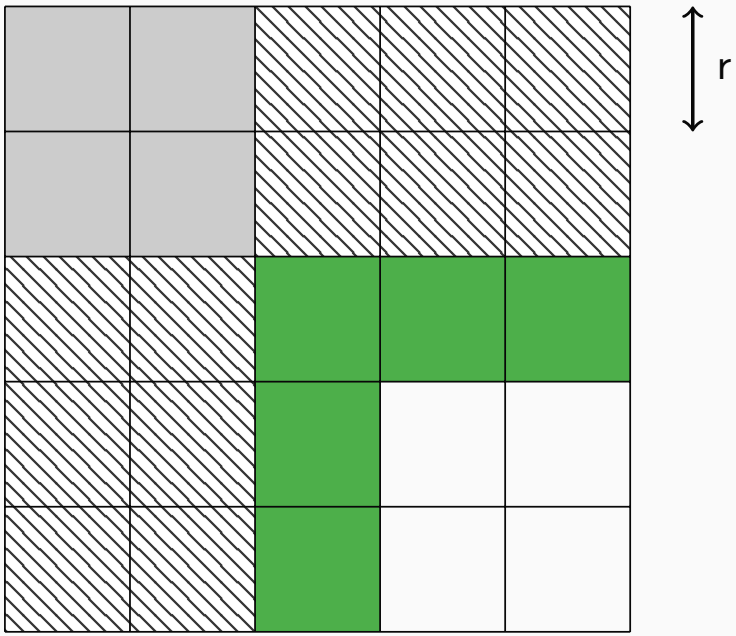
- Partition the panels into a smaller blocks of size s.
- Accumulate the inner update into the fp32 buffer (left-looking) using the tensor cores (TC32).
- Perform the inner panel factorization in precision u_{32}.

New doubly partitioned left-looking algorithm



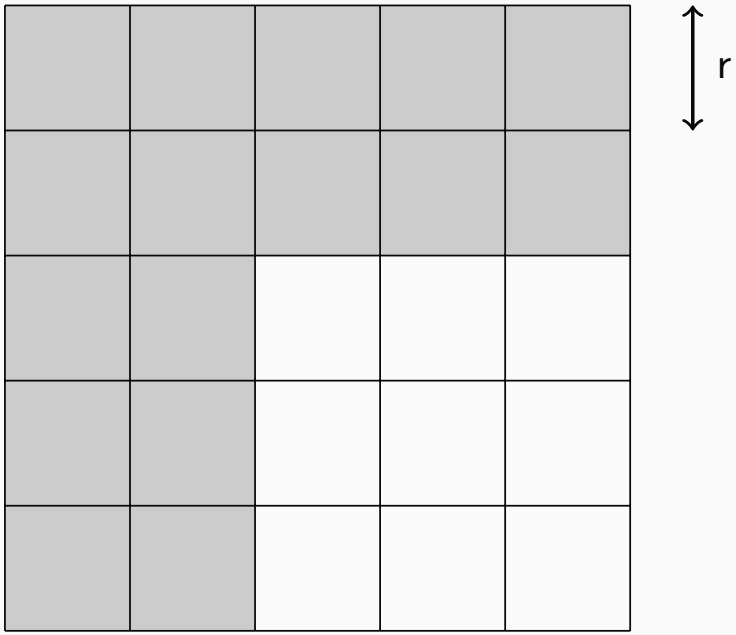
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

New doubly partitioned left-looking algorithm



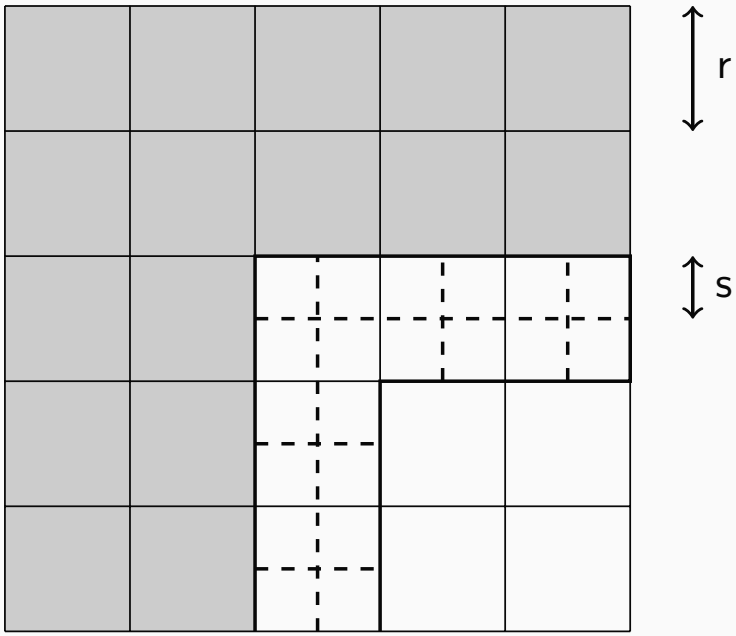
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

New doubly partitioned left-looking algorithm



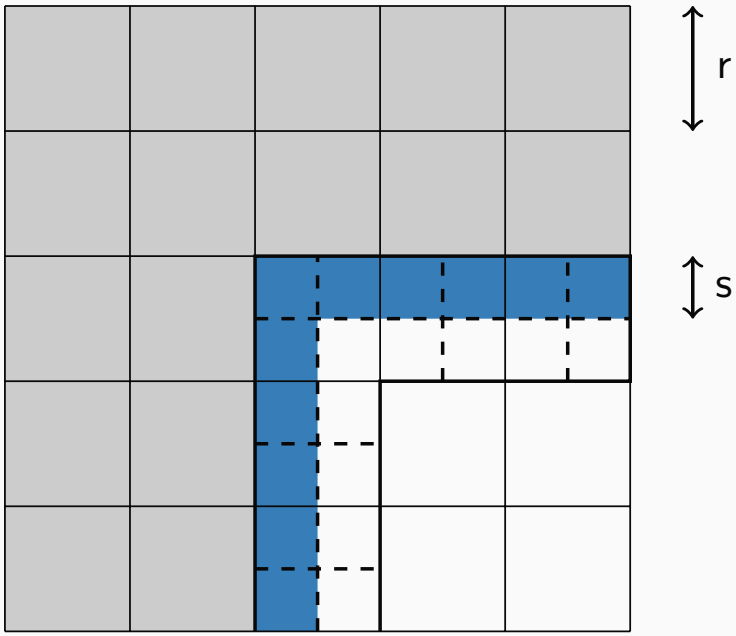
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

New doubly partitioned left-looking algorithm



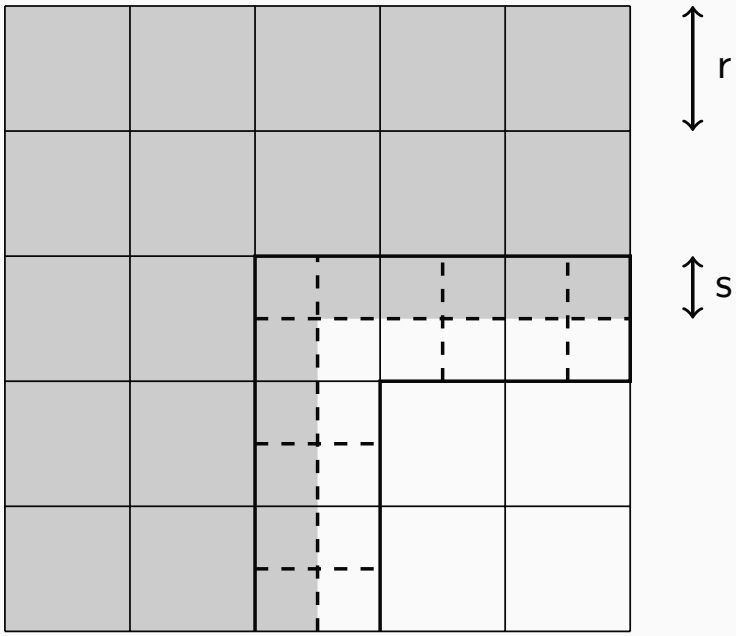
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

New doubly partitioned left-looking algorithm



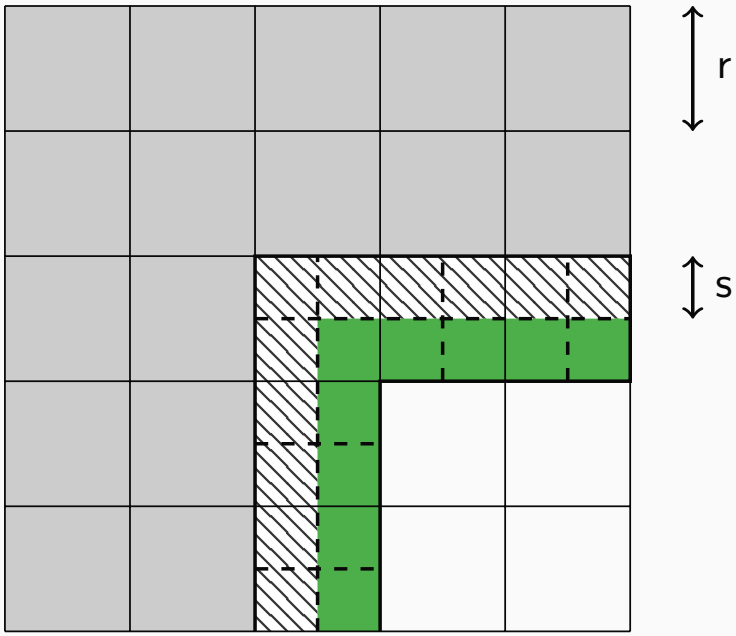
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

New doubly partitioned left-looking algorithm



for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

New doubly partitioned left-looking algorithm

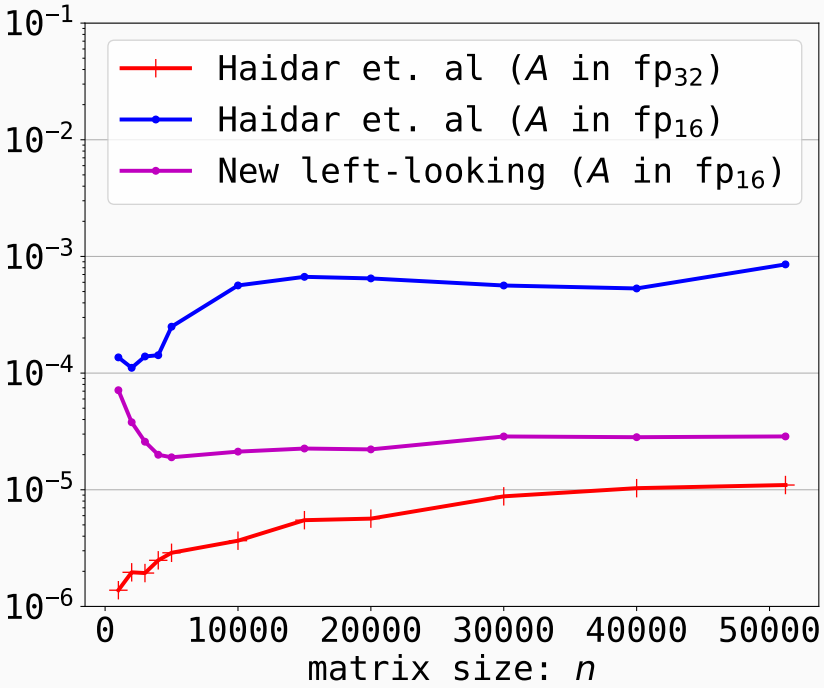


for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij} U_{jk}$.
      Update $B_{ki} \leftarrow B_{ki} - L_{kj} U_{ji}$.
    end for
  end for
  Partition L_{ik} and U_{ki} (s×s blocks).
  Compute L_{ik} and U_{ki}, using new
  left-looking algorithm (TC32).
end for

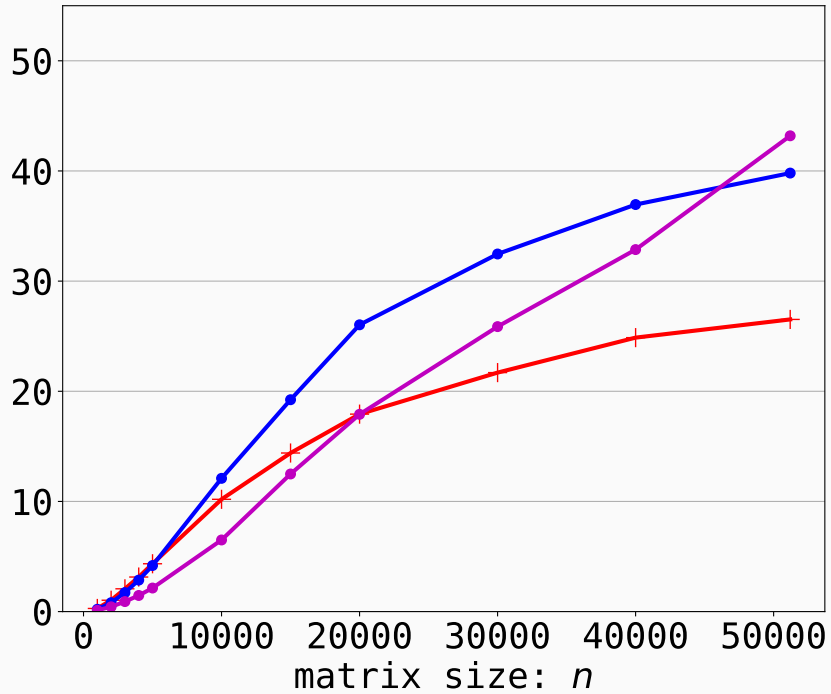
New doubly partitioned left-looking algorithm

Backward error bounds

Haidar et. al $u = u_{32}$	New left-looking $u = u_{16}$	New 2-lvl left-looking $u = u_{16}$
$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$	



Backward error

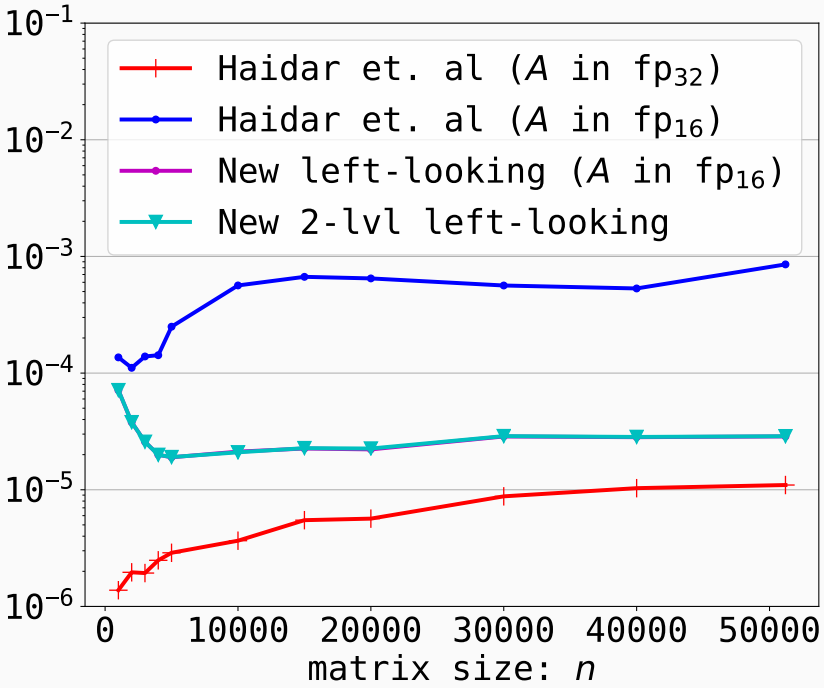


Performance (TFlop/s)

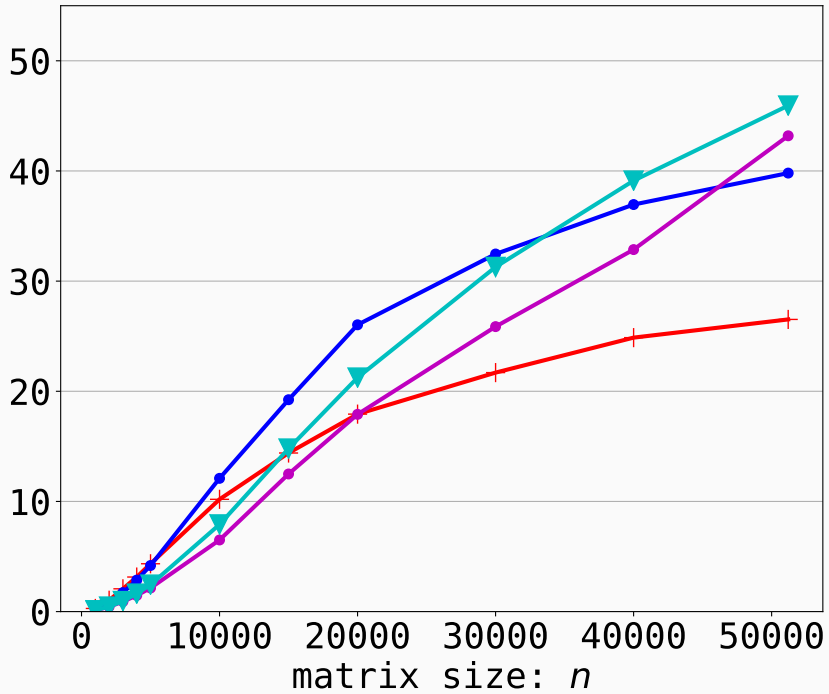
New doubly partitioned left-looking algorithm

Backward error bounds

Haidar et. al $u = u_{32}$	New left-looking $u = u_{16}$	New 2-lvl left-looking $u = u_{16}$
$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$



Backward error



Performance (TFlop/s)

New doubly partitioned left-looking algorithm

Would there be any benefits to using fp16 precision in the inner panel factorization rather than fp32?

New doubly partitioned left-looking algorithm

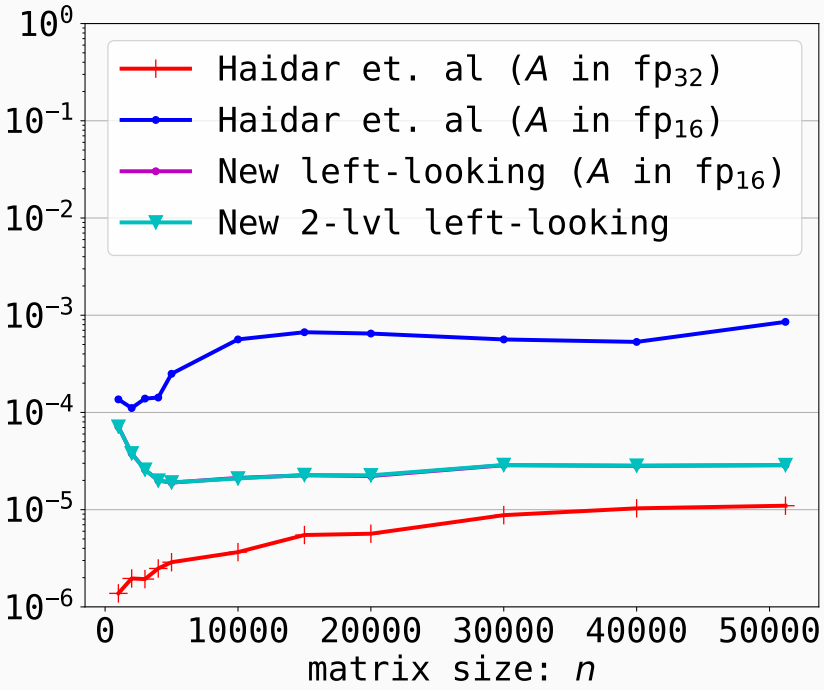
Would there be any benefits to using fp16 precision in the inner panel factorization rather than fp32?

Error bound varies from $2u_{16} + nu_{32}$ to $u_{16} + \max(nu_{32}, su_{16})$ with $s = 8$, we have $nu_{32} \geq su_{16}$ for $n \gtrsim 6.6 \times 10^4 \Rightarrow$ on a single GPU su_{16} dominates the error.

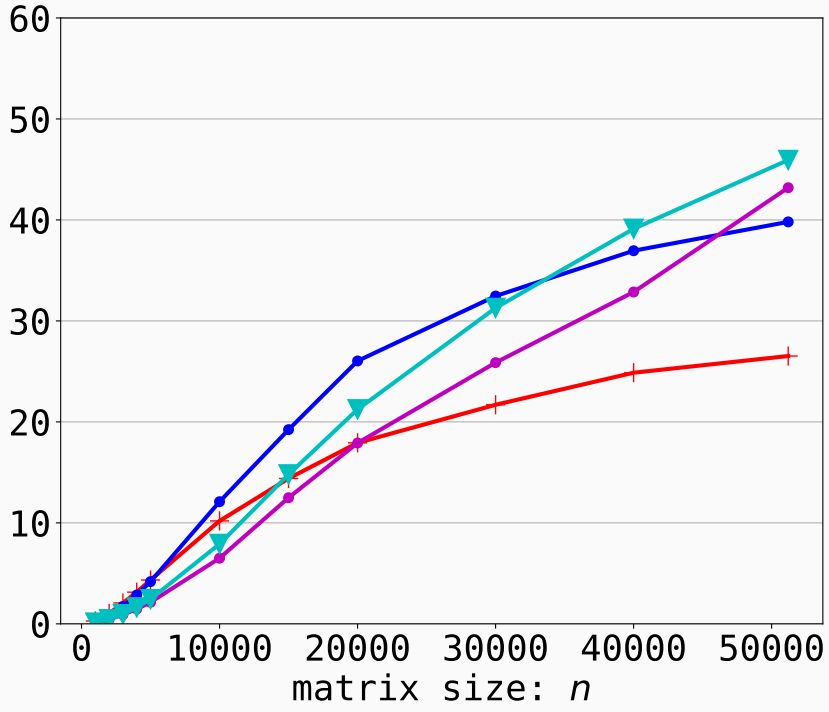
New doubly partitioned left-looking algorithm

Would there be any benefits to using fp16 precision in the inner panel factorization rather than fp32?

Error bound varies from $2u_{16} + nu_{32}$ to $u_{16} + \max(nu_{32}, su_{16})$ with $s = 8$, we have $nu_{32} \geq su_{16}$ for $n \gtrsim 6.6 \times 10^4 \Rightarrow$ on a single GPU su_{16} dominates the error.



Backward error

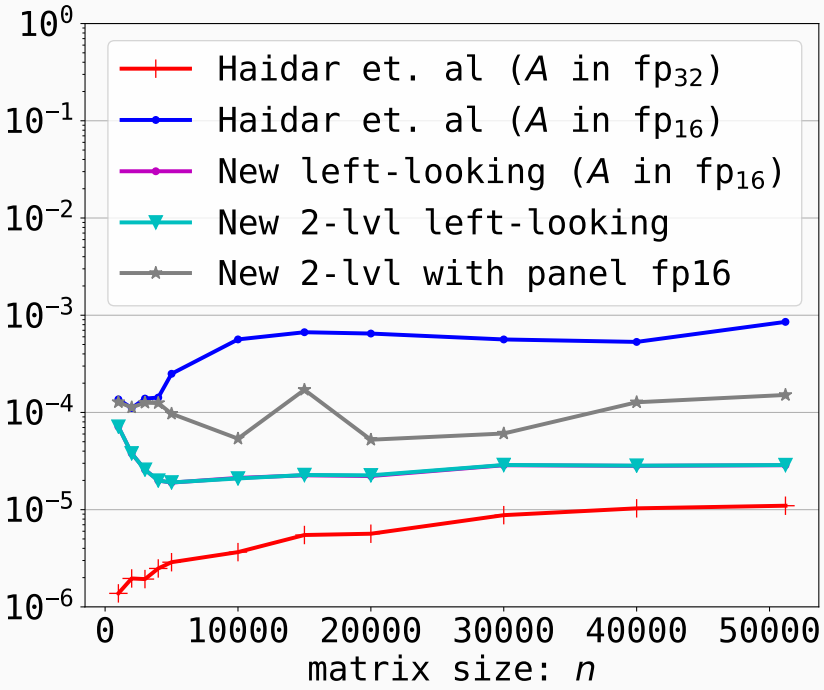


Performance (TFlop/s)

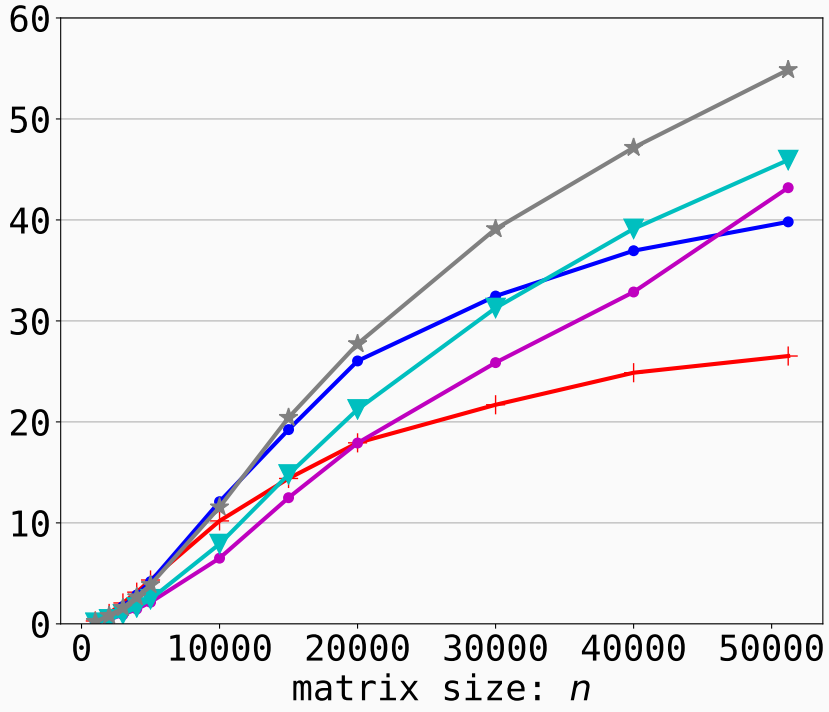
New doubly partitioned left-looking algorithm

Would there be any benefits to using fp16 precision in the inner panel factorization rather than fp32?

Error bound varies from $2u_{16} + nu_{32}$ to $u_{16} + \max(nu_{32}, su_{16})$ with $s = 8$, we have $nu_{32} \geq su_{16}$ for $n \gtrsim 6.6 \times 10^4 \Rightarrow$ on a single GPU su_{16} dominates the error.



Backward error



Performance (TFlop/s)

Conclusions and future work

New mixed precision LU factorization algorithm:

- A stored in fp16 $\Rightarrow$ half the memory footprint.
- A stored in fp16 $\Rightarrow$ half data movement.
- Up to 2$\times$ faster (55 TFlop/s).
- No significant loss of accuracy from storing A in fp16.

Conclusions and future work

New mixed precision LU factorization algorithm:

- A stored in fp16 $\Rightarrow$ half the memory footprint.
- A stored in fp16 $\Rightarrow$ half data movement.
- Up to 2$\times$ faster (55 TFlop/s).
- No significant loss of accuracy from storing A in fp16.

See our preprint: F. Lopez and T. Mary, *Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint* (MIMS EPrint 2020.20)

Thanks for listening!

Questions?