SIAM PP 2022

25 February 2022

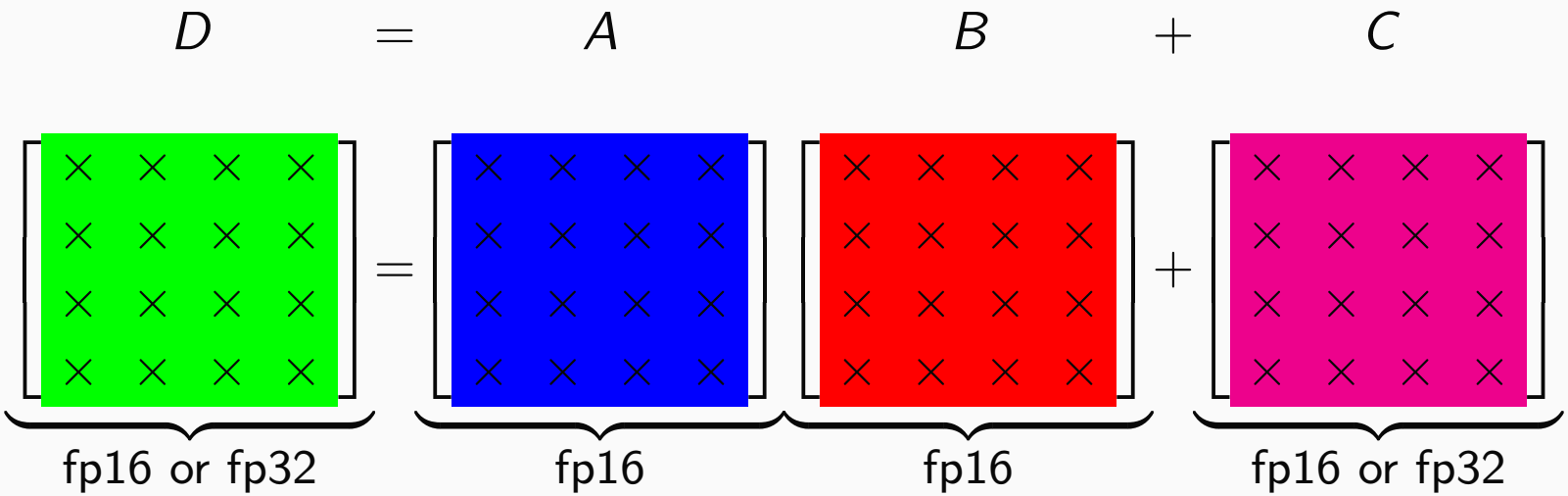
Reducing Data Movement in Mixed Precision LU Factorization with GPU Tensor Cores

Theo Mary

Sorbonne Université, CNRS, LIP6

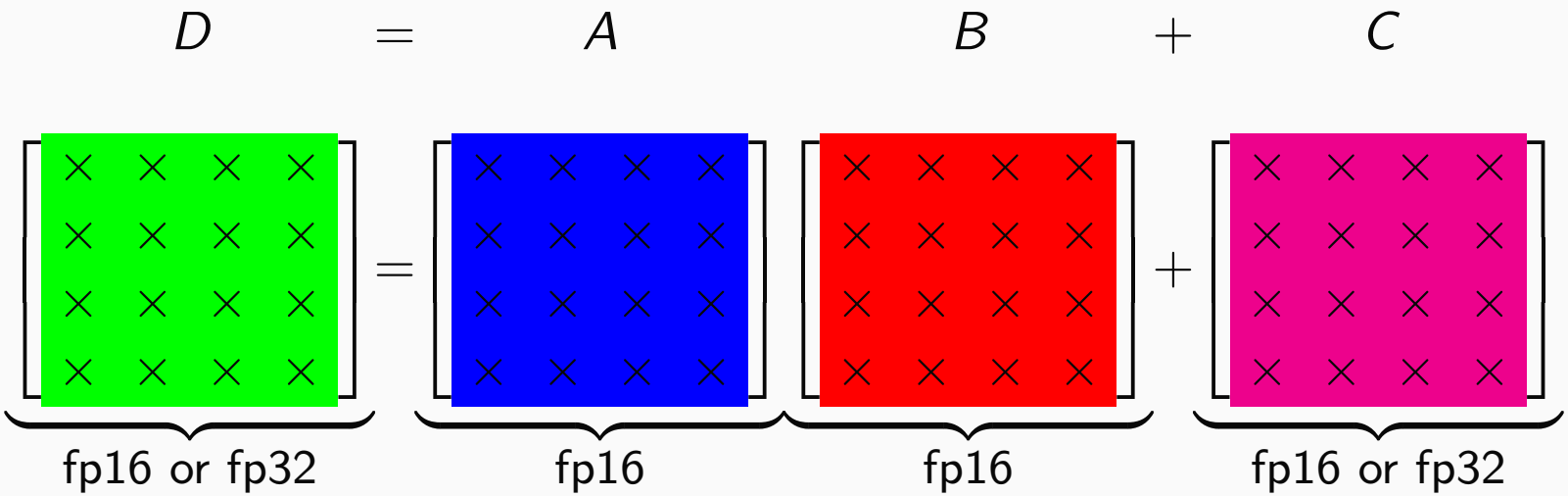
Joint work with Florent Lopez, Atef Dorai, Roman Iakymchuk

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



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

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



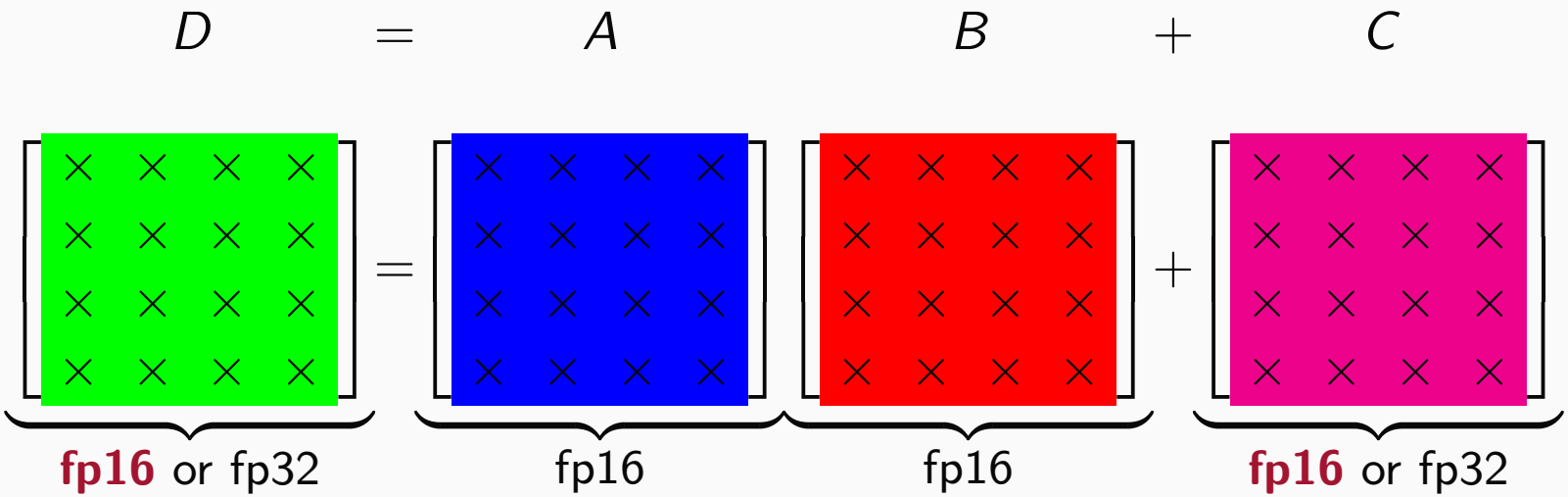
Performance peak: 125 TFlop/s (8× speedup vs fp32)

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]):

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

[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)

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



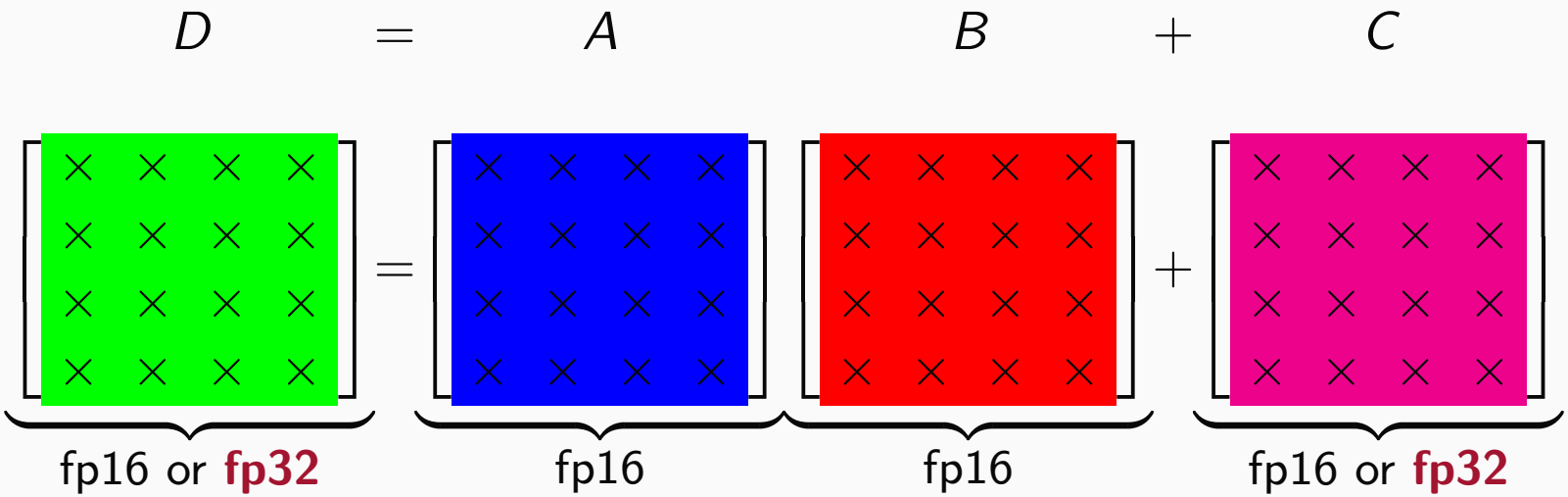
Performance peak: 125 TFlop/s ($8\times$ speedup vs fp32)

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)

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



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

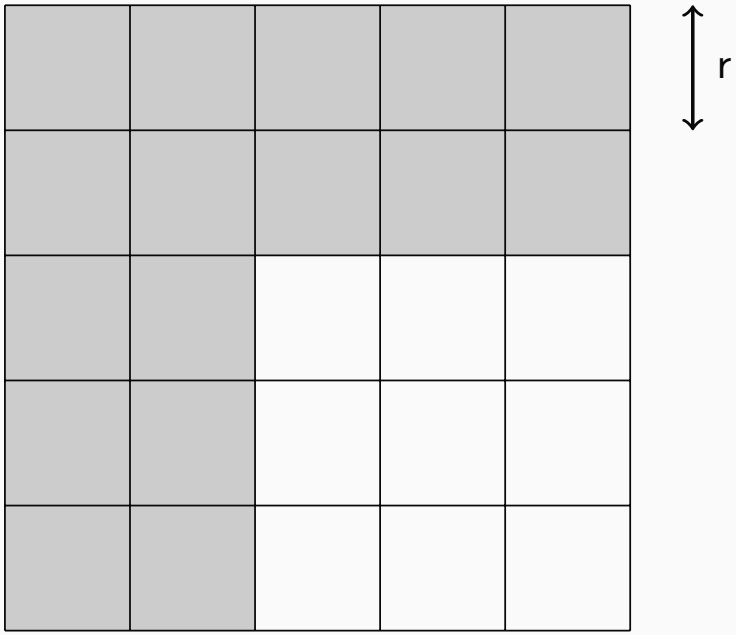
For the matrix product $C = AB$ where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ using tensor cores, the computed $\widehat{C}$ satisfies (Blanchard et al.[1]):

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

[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

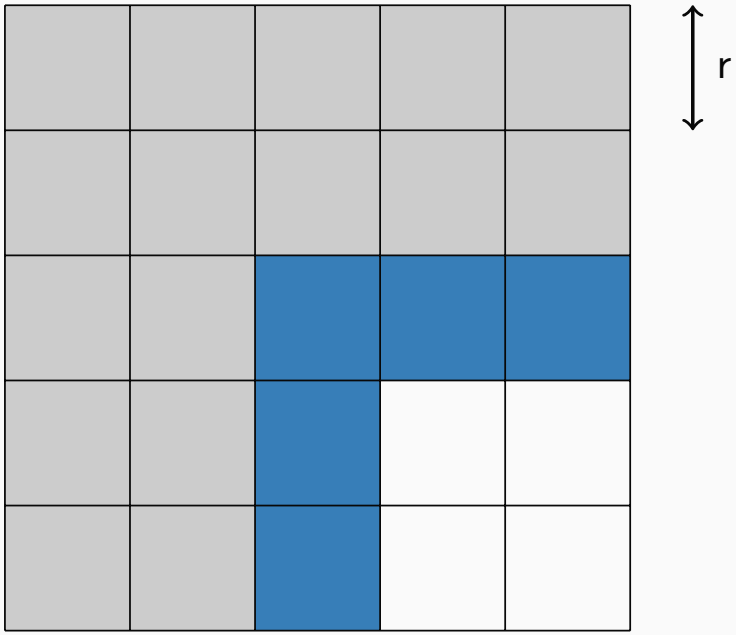
This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



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

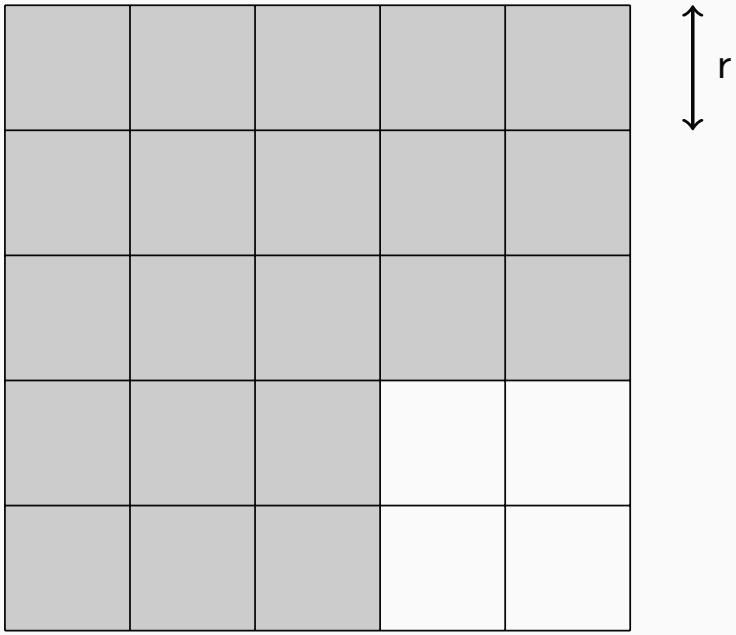
This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



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

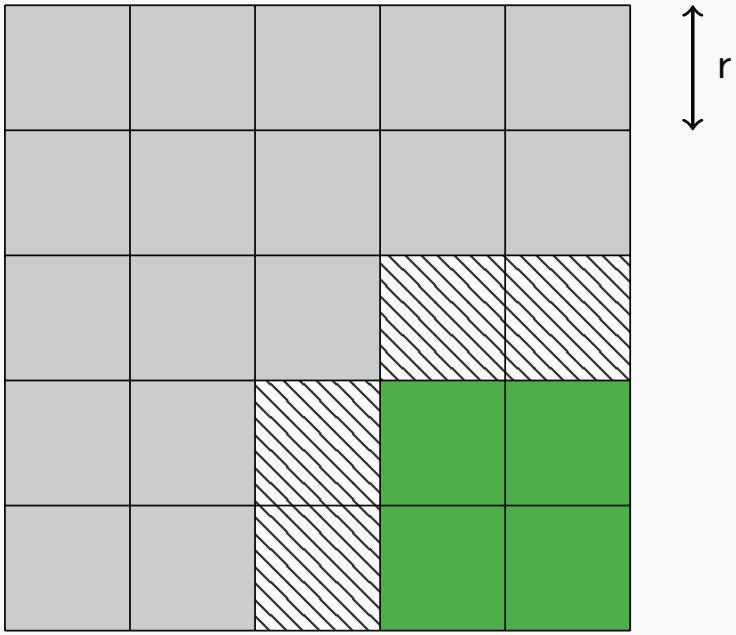
This algorithm computes $A = LU$ where $A \in \mathbb{R}^{n \times n}$, stored in precision $u = u_{16}$ or $u = u_{32}$, is partitioned into $r \times r$ blocks $\Rightarrow$ matrix–matrix (Level-3 BLAS) operations.



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]:

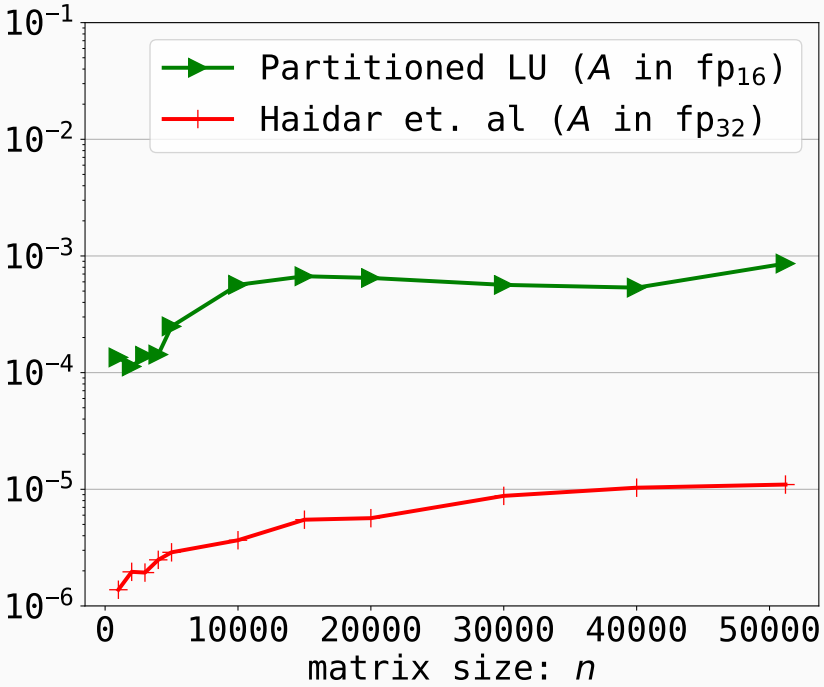
for $k = 1 : n/r$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$ (in precision u_{32}).
  for $i = k+1 : n/r$ **do**
    Solve $L_{ik}U_{kk} = A_{ik}$ for L_{ik} (in precision u_{32}).
    Solve $L_{kk}U_{ki} = A_{ki}$ for U_{ki} (in precision u_{32}).
  end for
  for $i = k+1 : n/r$ **do**
    for $j =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

Standard LU $u = u_{16}$	Tensor core LU $u = u_{32}$
nu_{16}	$2u_{16} + nu_{32}$

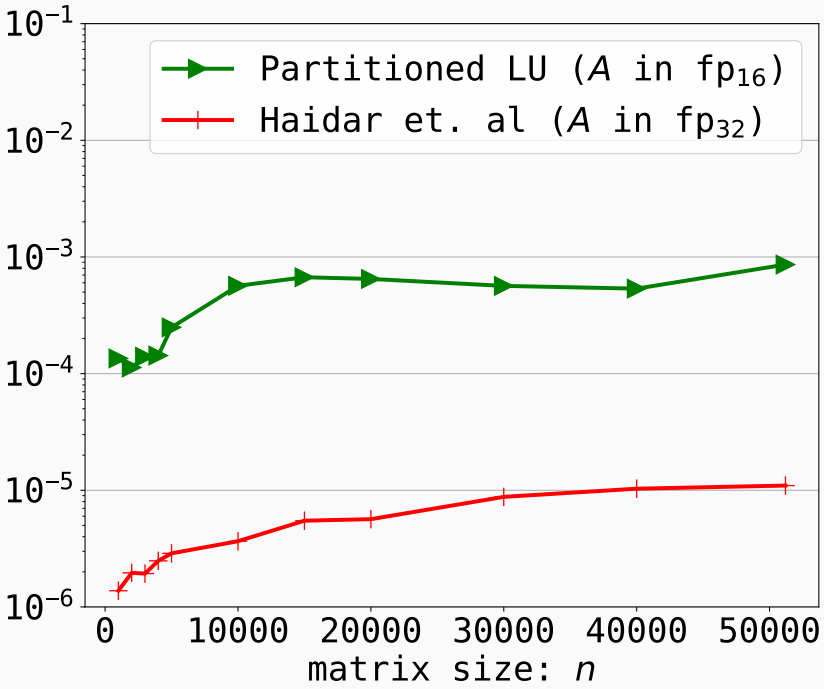


Backward error

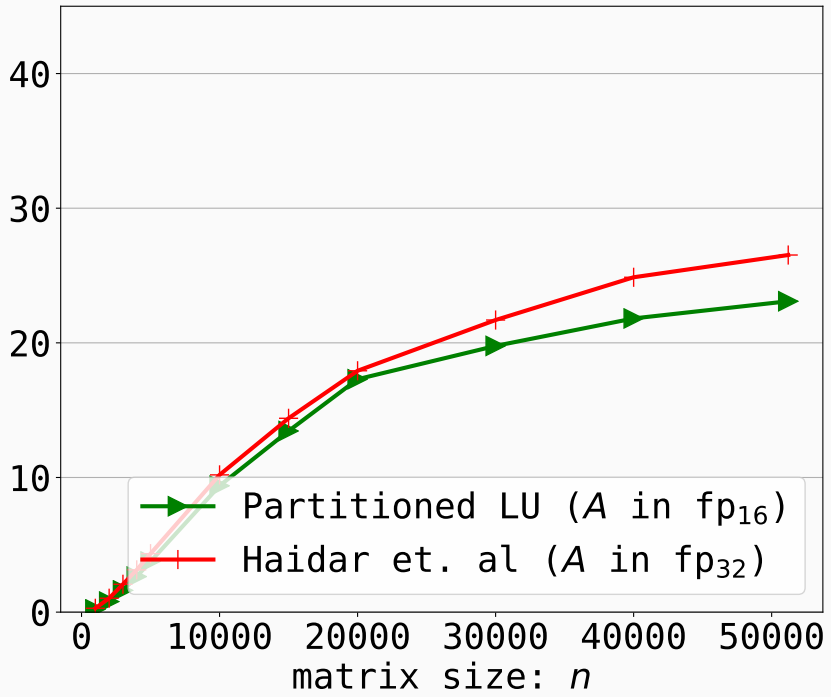
State-of-the-art mixed precision LU factorization

Backward error bounds from Blanchard et al.

Standard LU $u = u_{16}$	Tensor core LU $u = u_{32}$
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:

- ▲ Much more accurate compared to fp16 LU factorization.
- ▼ No reduction in memory footprint (A stored in precision u_{32}).
- ▼ Only marginally faster $\Rightarrow$ why?
  - LU factorization is traditionally a compute-bound operation...
  - With Tensor Cores, flops are 8$\times$ faster
  - Matrix is stored in fp32 $\Rightarrow$ **data movement is unchanged !**

  $\Rightarrow$ LU with tensor cores becomes memory-bound !

Reduce memory footprint and data movement

Mixed precision LU factorization algorithm from Haidar et al:

- ▲ Much more accurate compared to fp16 LU factorization.
- ▼ No reduction in memory footprint (A stored in precision u_{32}).
- ▼ Only marginally faster $\Rightarrow$ why?
  - LU factorization is traditionally a compute-bound operation...
  - With Tensor Cores, flops are 8$\times$ faster
  - Matrix is stored in fp32 $\Rightarrow$ **data movement is unchanged !**

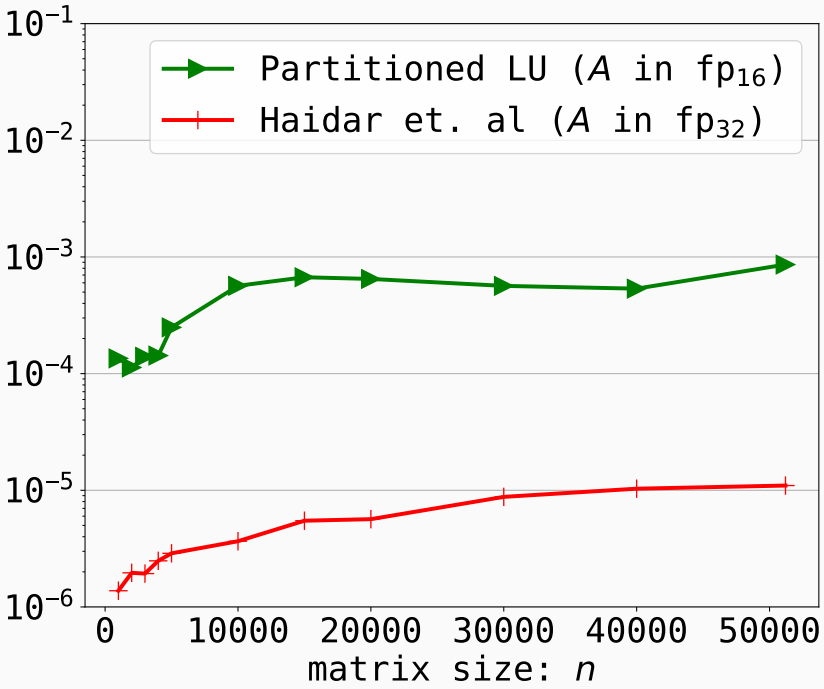
  $\Rightarrow$ LU with tensor cores becomes memory-bound !

Idea: store A in precision u_{16}, perform panel factorization in fp16 and updates TC16.

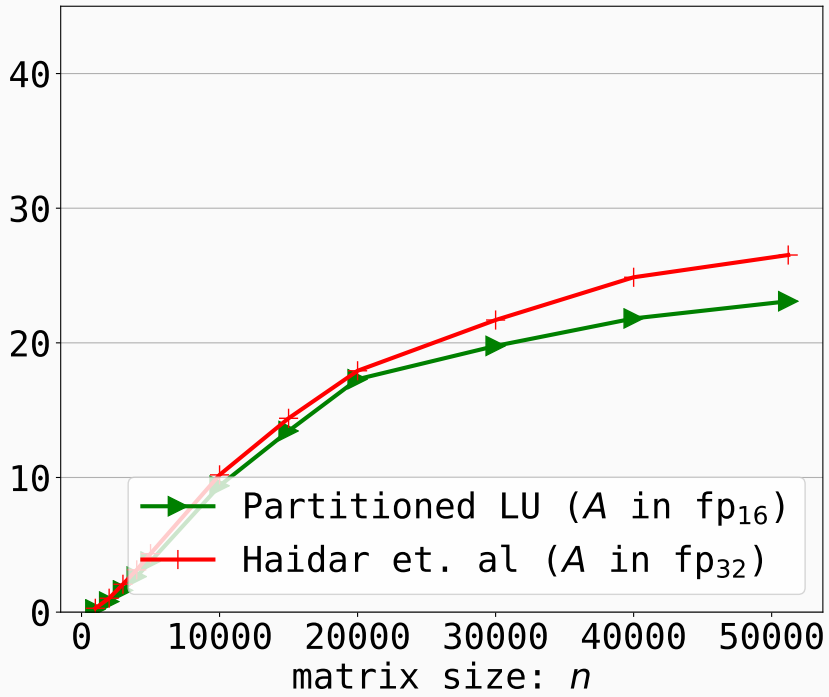
State-of-the-art mixed precision LU factorization

Backward error bounds

Standard LU $u = u_{16}$	Tensor core LU $u = u_{32}$	Tensor core LU $u = u_{16}$
nu_{16}	$2u_{16} + nu_{32}$	



Backward error

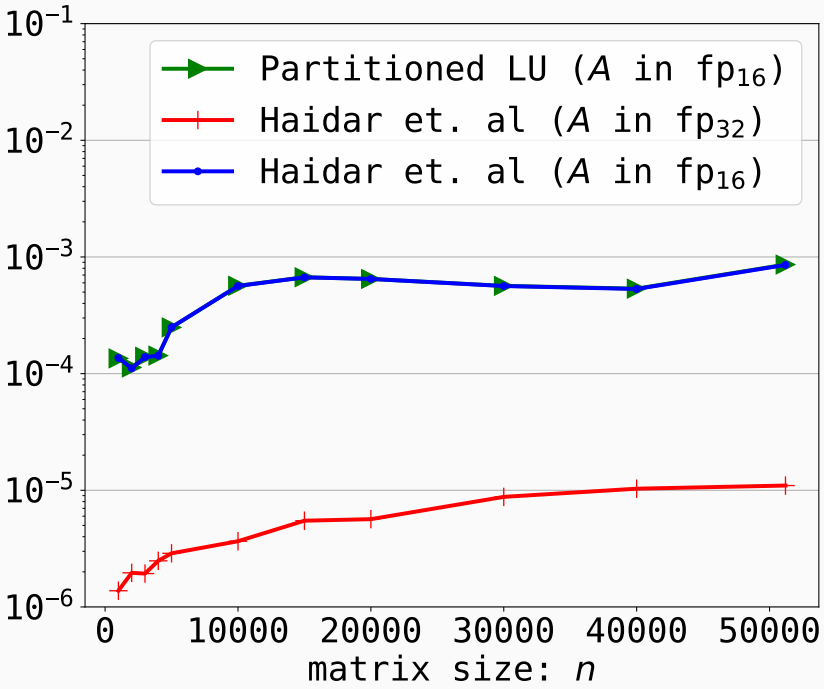


Performance (TFlop/s)

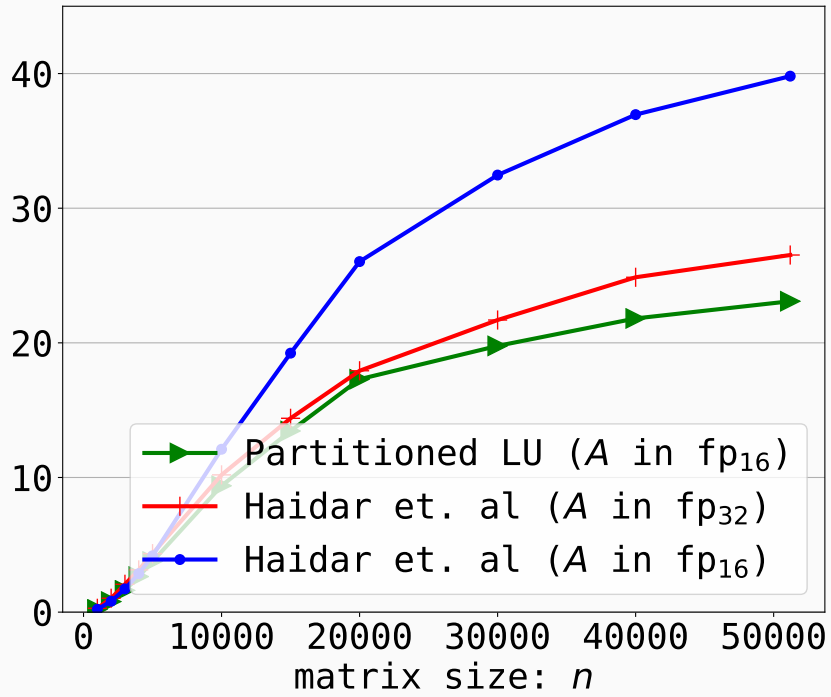
State-of-the-art mixed precision LU factorization

Backward error bounds

Standard LU $u = u_{16}$	Tensor core LU $u = u_{32}$	Tensor core LU $u = u_{16}$
nu_{16}	$2u_{16} + nu_{32}$	nu_{16}



Backward error



Performance (TFlop/s)

Storing A in fp16

Objective: Reduce data movement without significant loss of accuracy.

Storing A in fp16

Objective: Reduce data movement 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: Reduce data movement 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: Reduce data movement 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: **need to control the fp32 buffer's size!** In standard right-looking LU, need a buffer corresponding to trailing submatrix which, in the first few steps of the factorization is almost as large as the matrix itself.

Storing A in fp16

Objective: Reduce data movement 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

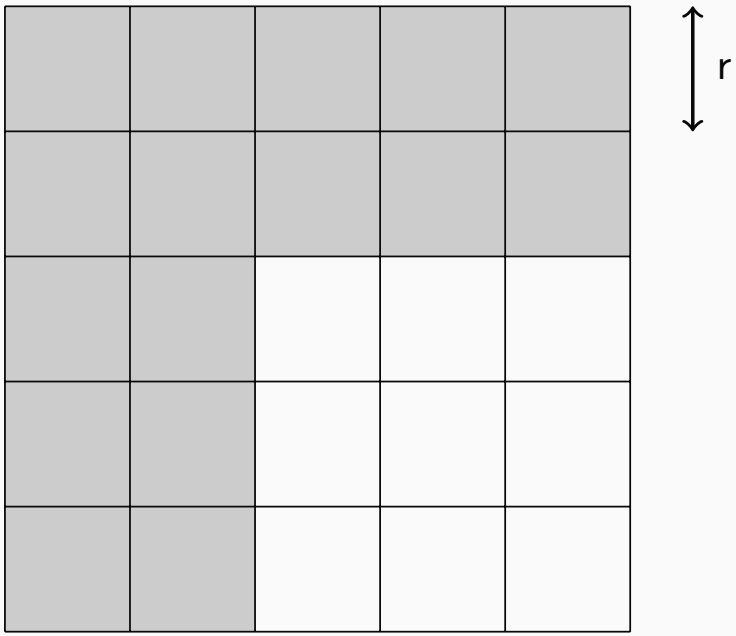
$$\begin{aligned} B_{ij} &= \mathrm{fl}_{32}(A_{ij}) \\ B_{ij} &\leftarrow B_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj} \\ A_{ij} &\leftarrow \mathrm{fl}_{16}(B_{ij}) \end{aligned}$$

Problem: **need to control the fp32 buffer's size!** In standard right-looking LU, need a buffer corresponding to trailing submatrix 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

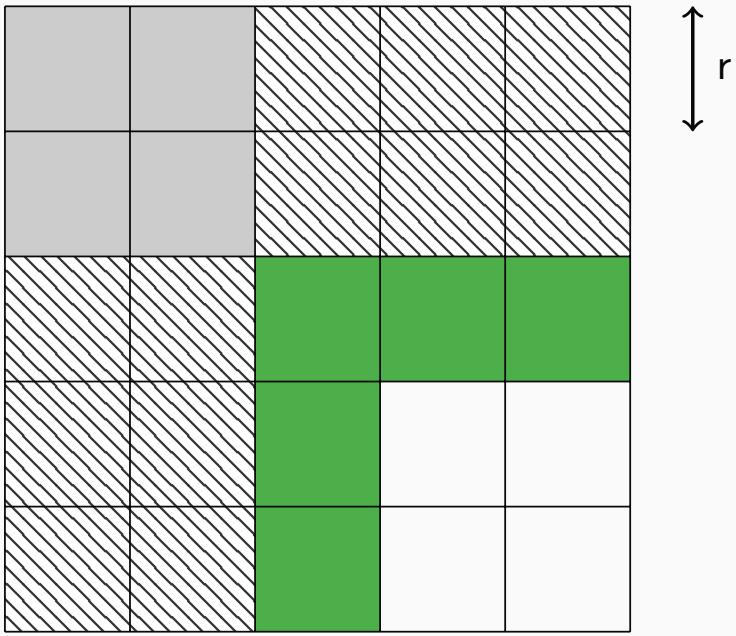
Left-looking LU factorization: at every step, the current panel is updated w.r.t the already computed factors (to the left) before being factored.



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

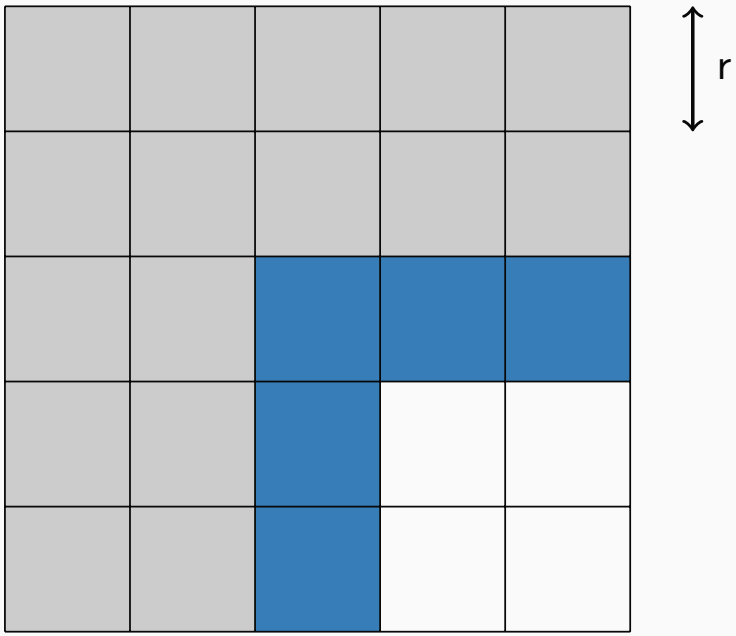
Left-looking LU factorization: at every step, the current panel is updated w.r.t the already computed factors (to the left) before being factored.



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

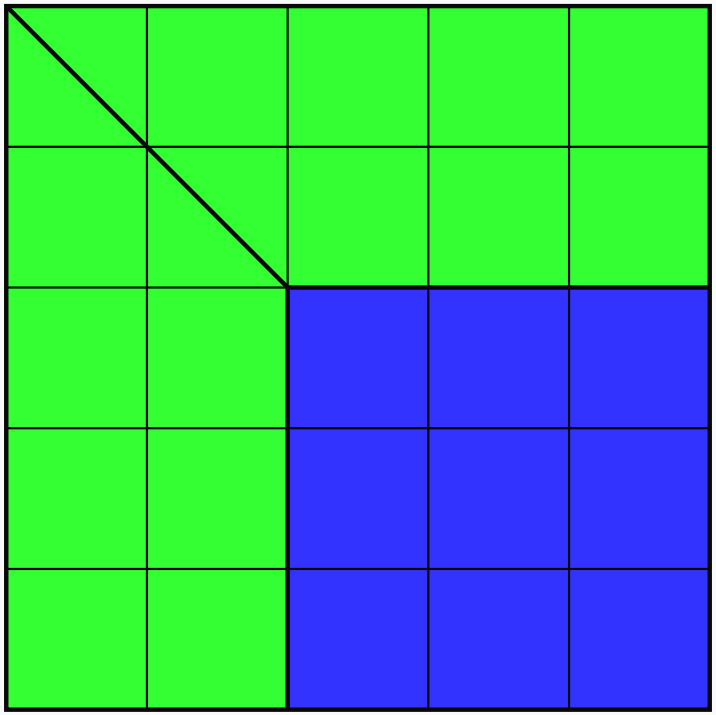
New left-looking mixed-precision LU factorization

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

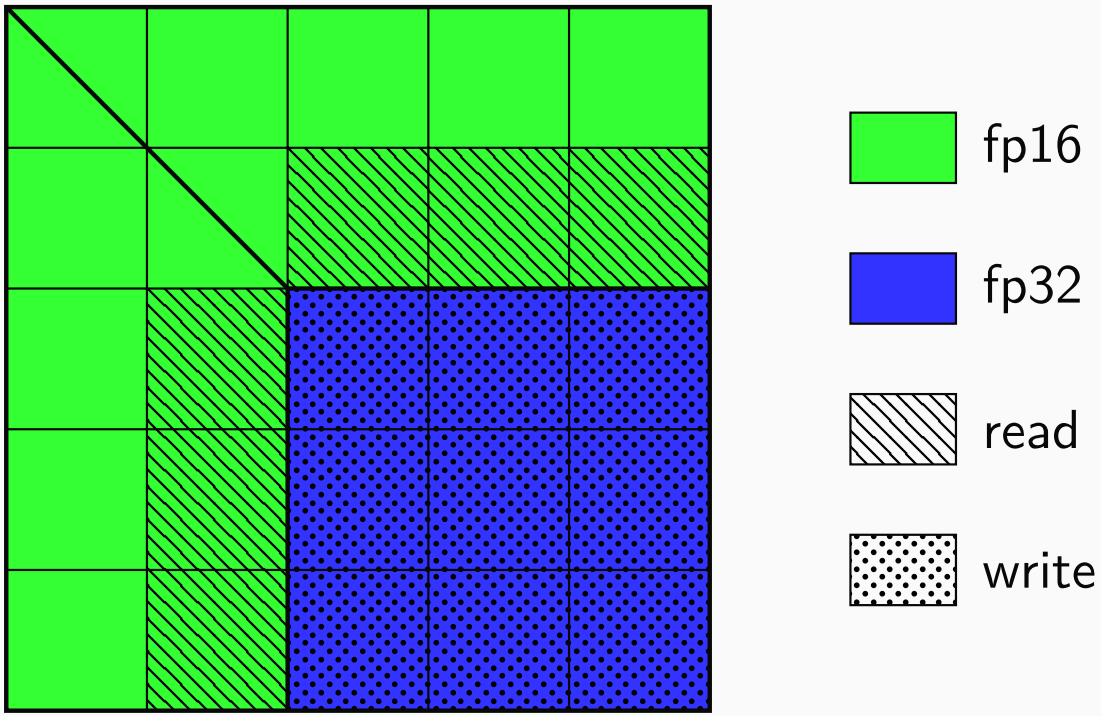
Data movement analysis

Matrix after 2 steps:



Data movement analysis

Matrix after 2 steps:

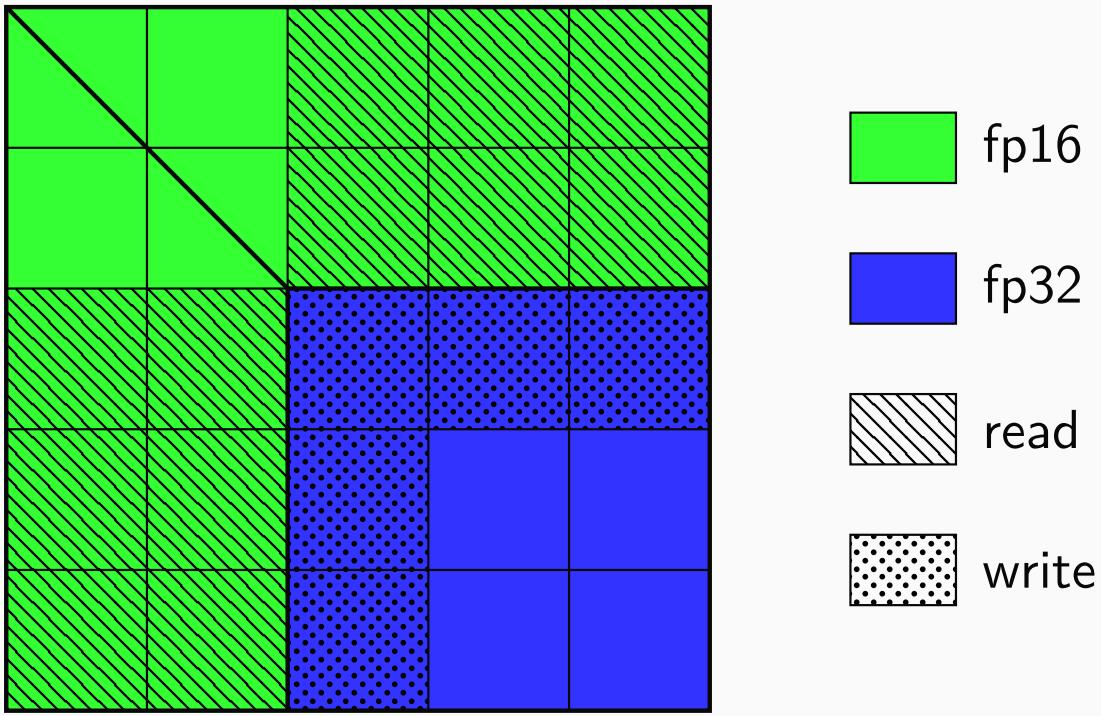


Volume of data movement for an $n \times n$ matrix with $r \times r$ blocks:

- RL algorithm: $n^3/3r$ fp32 entries $+$ $O(n^2)$ fp16 entries
  $\rightarrow$ $4n^3/3r + O(n^2)$ bytes

Data movement analysis

Matrix after 2 steps:



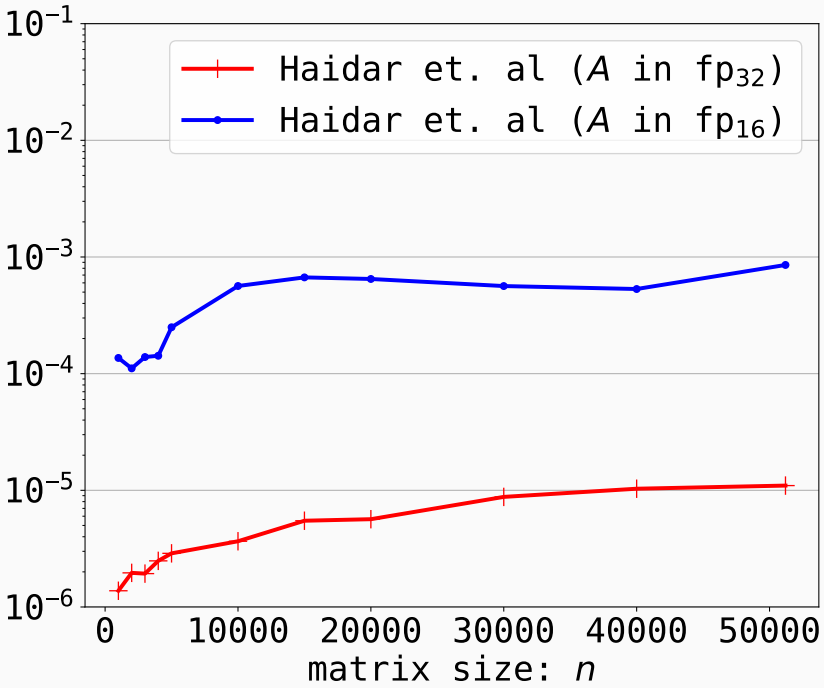
Volume of data movement for an $n \times n$ matrix with $r \times r$ blocks:

- RL algorithm: $n^3/3r$ fp32 entries + $O(n^2)$ fp16 entries
  $\rightarrow 4n^3/3r + O(n^2)$ bytes
- LL algorithm: $n^3/3r$ fp16 entries + $O(n^2)$ fp32 entries
  $\rightarrow 2n^3/3r + O(n^2)$ bytes

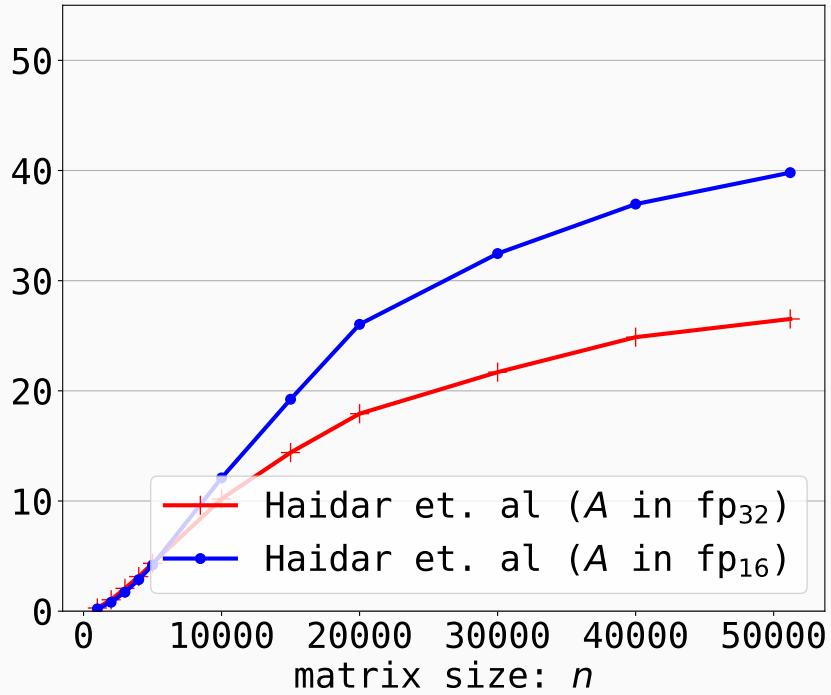
$\Rightarrow$ For large n, **volume of data movement is halved!**

New left-looking mixed-precision LU factorization

Backward error bounds		
Right-looking $u = u_{32}$	Right-looking $u = u_{16}$	New left-looking $u = u_{16}$
$2u_{16} + nu_{32}$	nu_{16}	



Backward error

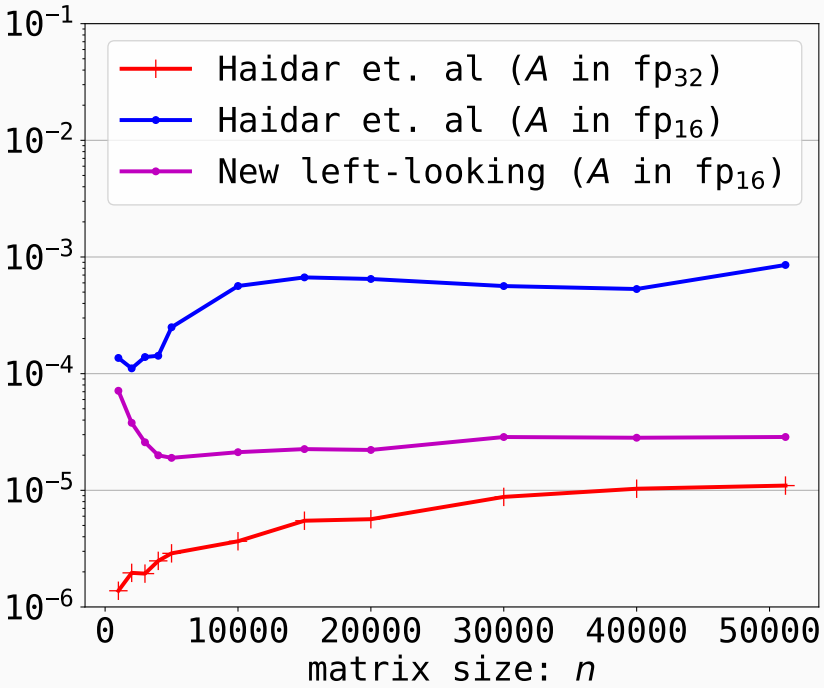


Performance (TFlop/s)

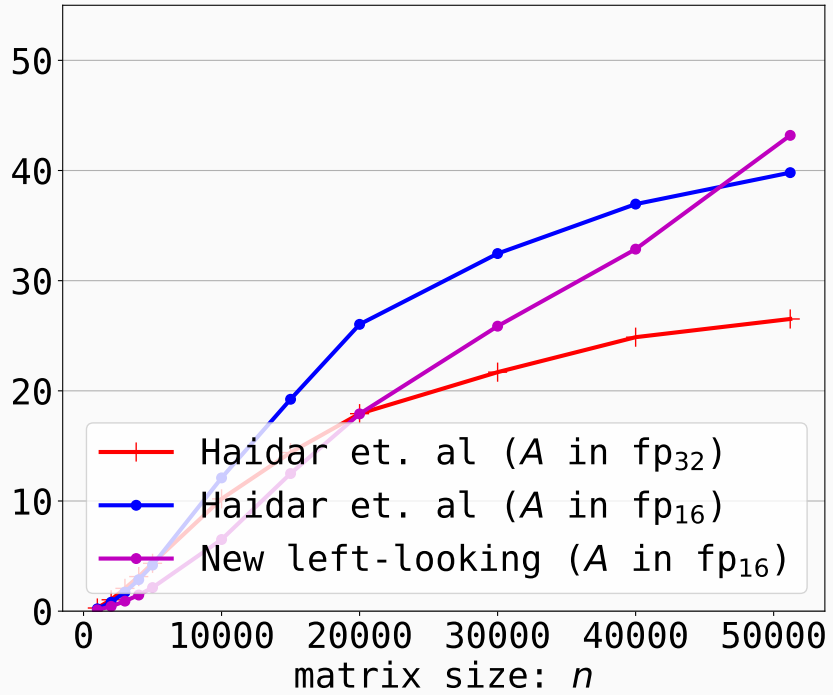
New left-looking mixed-precision LU factorization

Backward error bounds

Right-looking $u = u_{32}$	Right-looking $u = u_{16}$	New left-looking $u = u_{16}$
$2u_{16} + nu_{32}$	nu_{16}	$2u_{16} + nu_{32}$



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:

- 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: Switch panel precision to u_{16} and use fp16 or TC16 arithmetic for 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: Switch panel precision to u_{16} and use fp16 or TC16 arithmetic for the panel factorization.

- ▼ 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, 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?

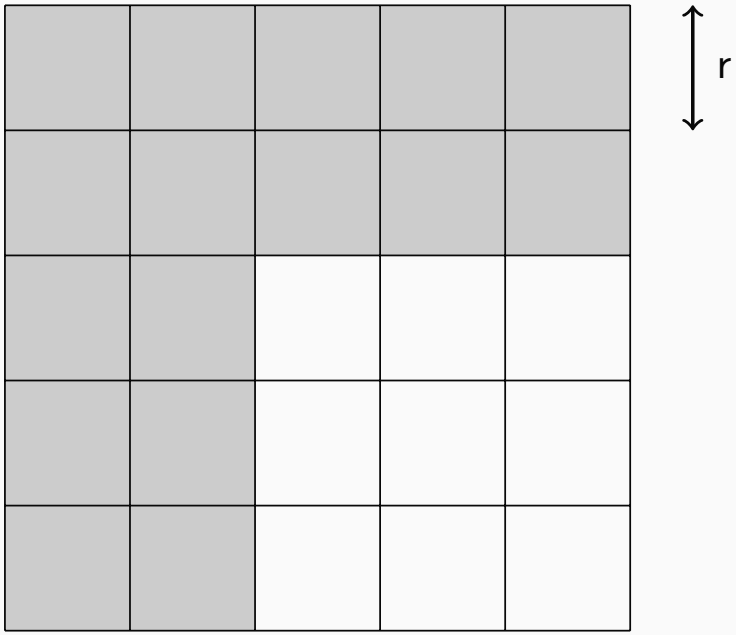
Problems:

- 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:

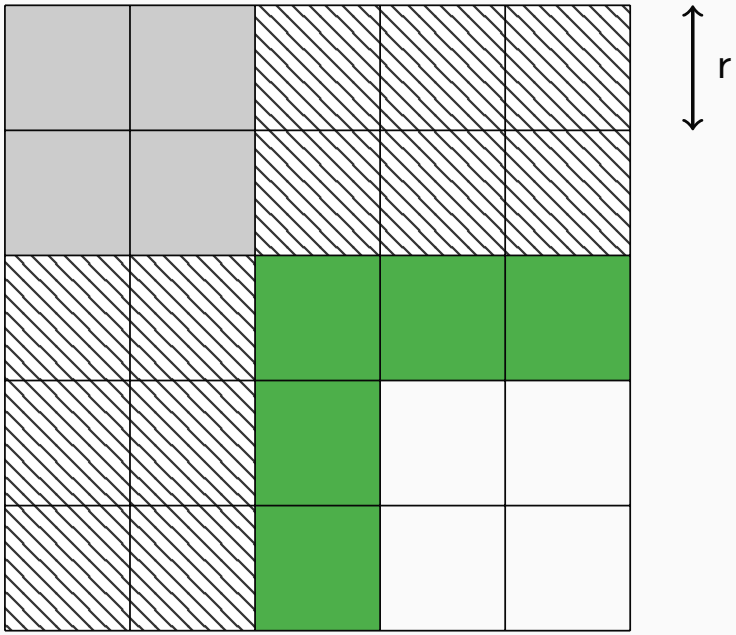
- Partition the panels into a smaller blocks of size $s < r$.
- Apply the left-looking algorithm to panel factorization (TC32). Use the same fp32 buffer for outer and inner updates.
- Perform the inner panel factorization in precision u_{32}.

New doubly partitioned left-looking algorithm



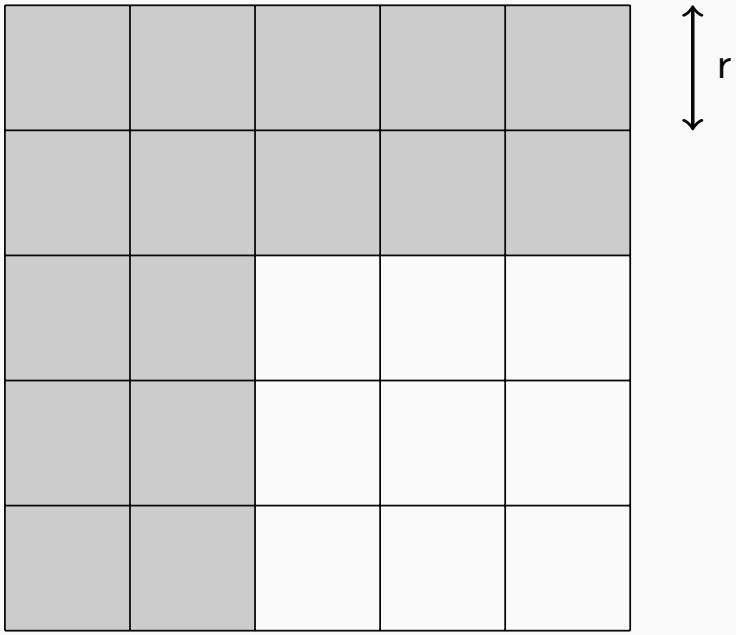
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij}U_{jk}$.
      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



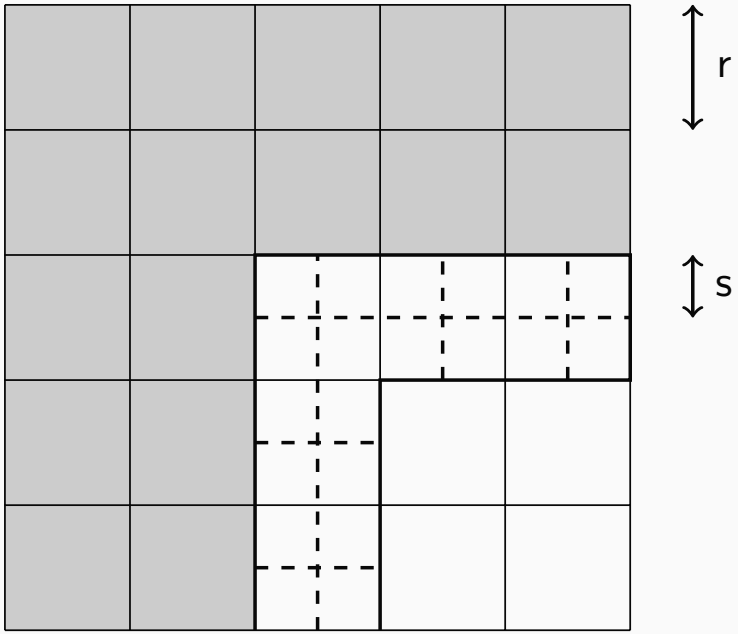
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij}U_{jk}$.
      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



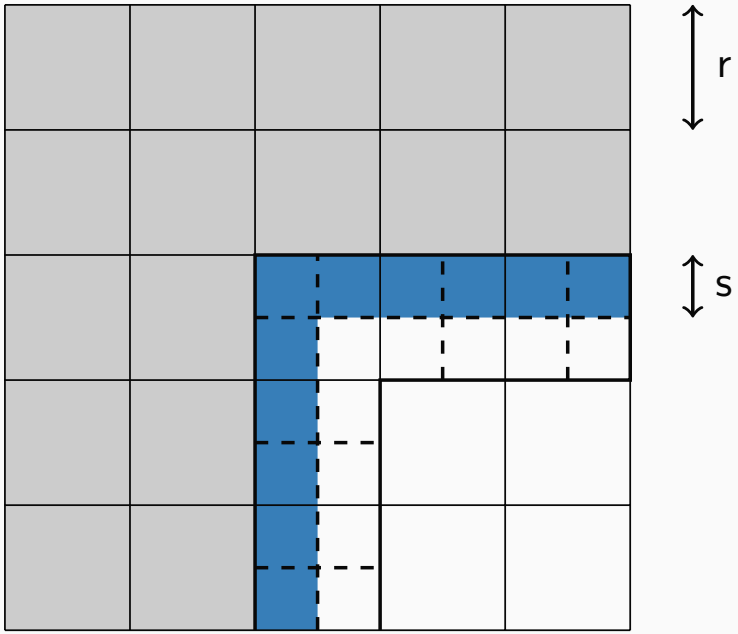
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij}U_{jk}$.
      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



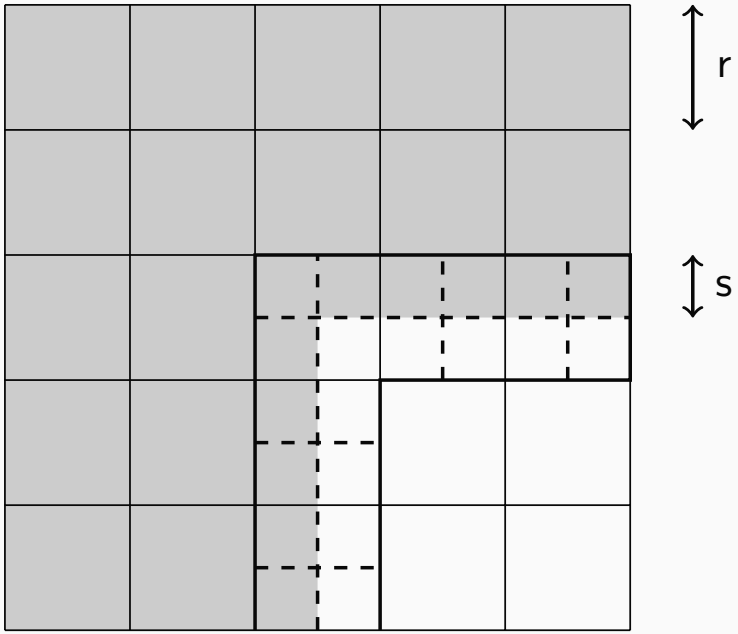
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij}U_{jk}$.
      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



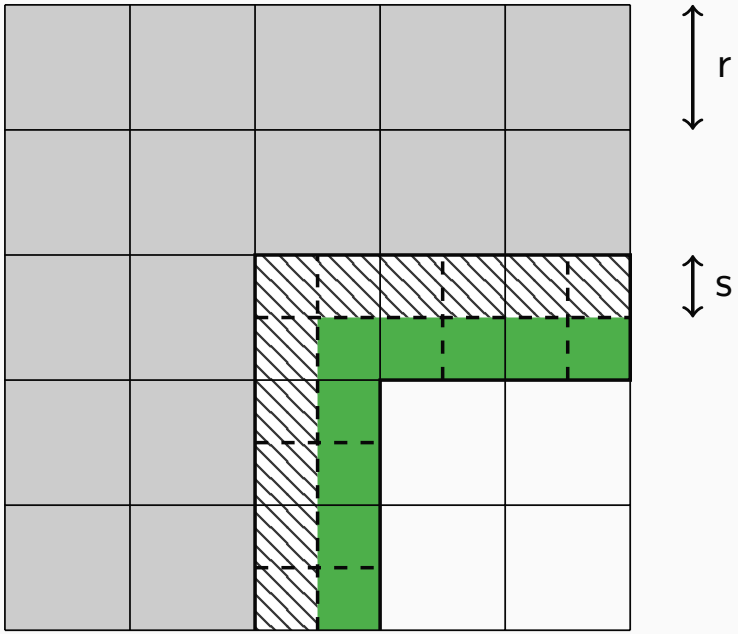
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij}U_{jk}$.
      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

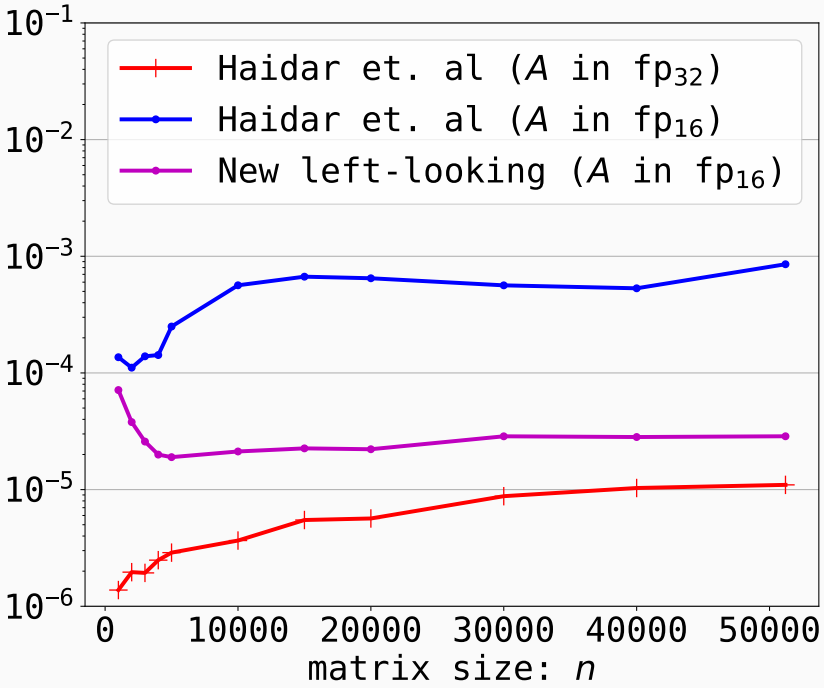
New doubly partitioned left-looking algorithm



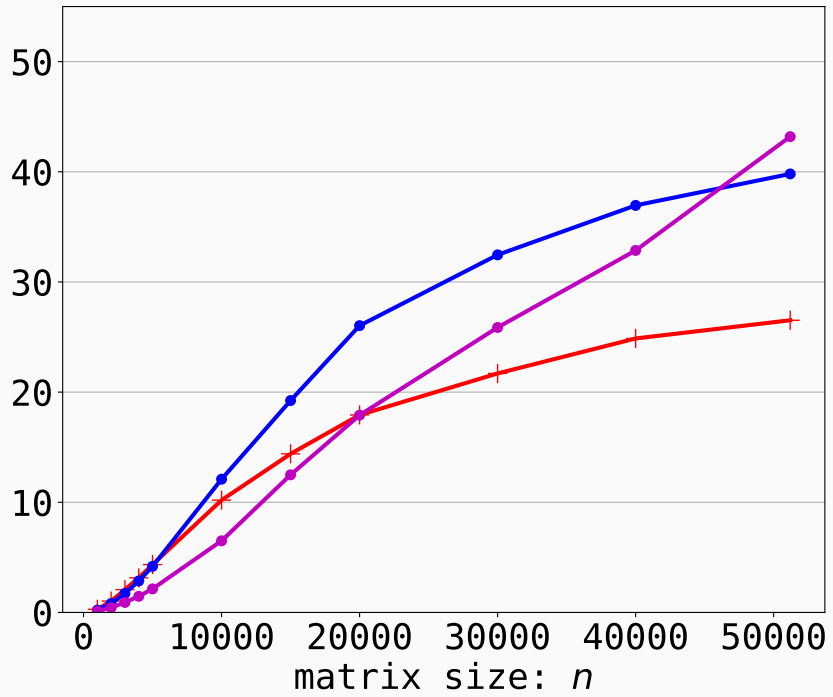
for $k = 1 : n/r$ **do**
  for $i = k : n/r$ **do**
    Convert to fp32:
    $B_{ik} \leftarrow \mathrm{fl}_{32}(A_{ik})$ and
    $B_{ki} \leftarrow \mathrm{fl}_{32}(A_{ki})$.
    for $j = 1 : k - 1$ **do**
      Using tensor cores (TC32):
      Update $B_{ik} \leftarrow B_{ik} - L_{ij}U_{jk}$.
      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.	New left-looking	
	1-lvl	
$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$	



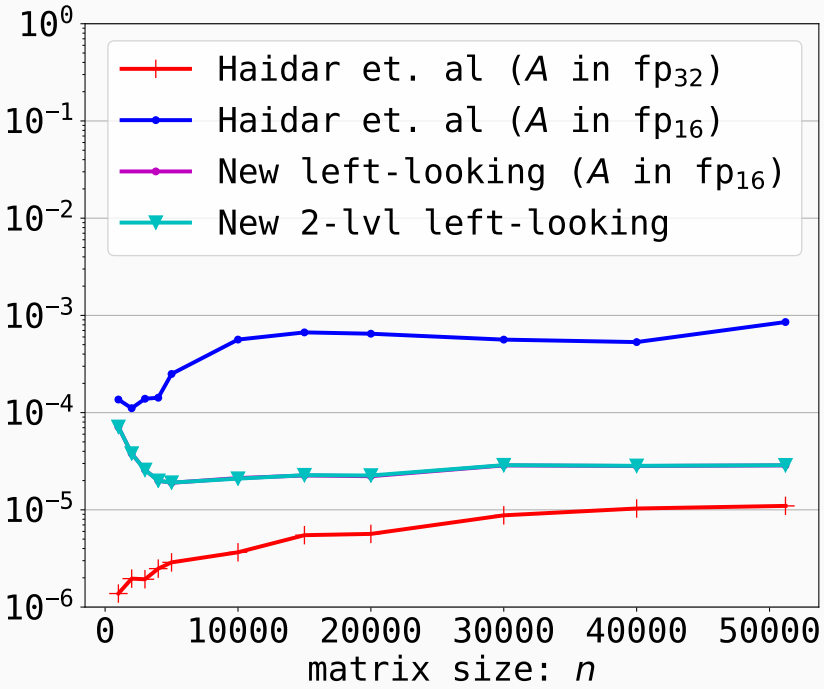
Backward error



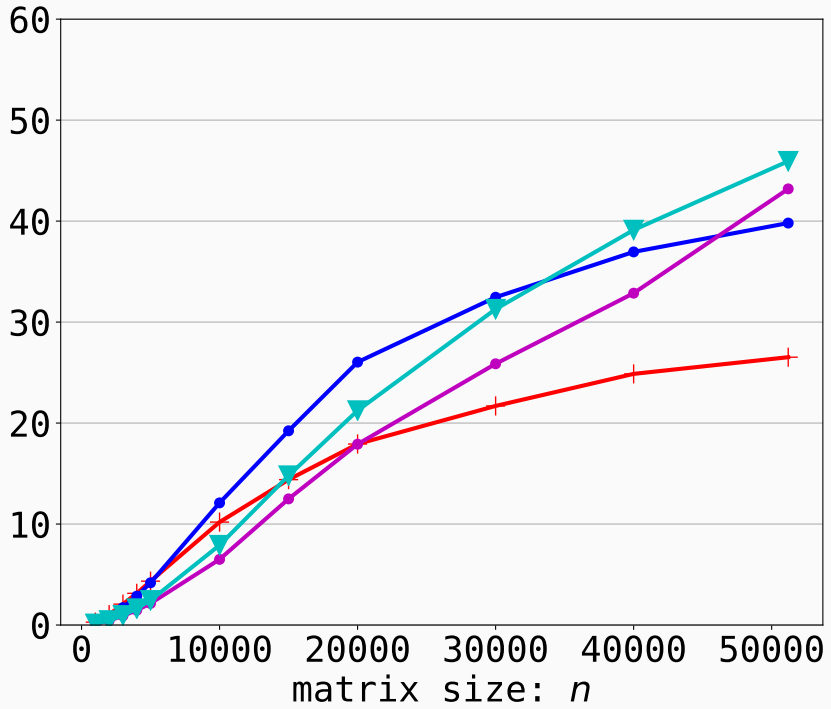
Performance (TFlop/s)

New doubly partitioned left-looking algorithm

Backward error bounds		
Haidar et al.	New left-looking	
	1-lvl	2-lvl, fp32 inner
$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$



Backward error

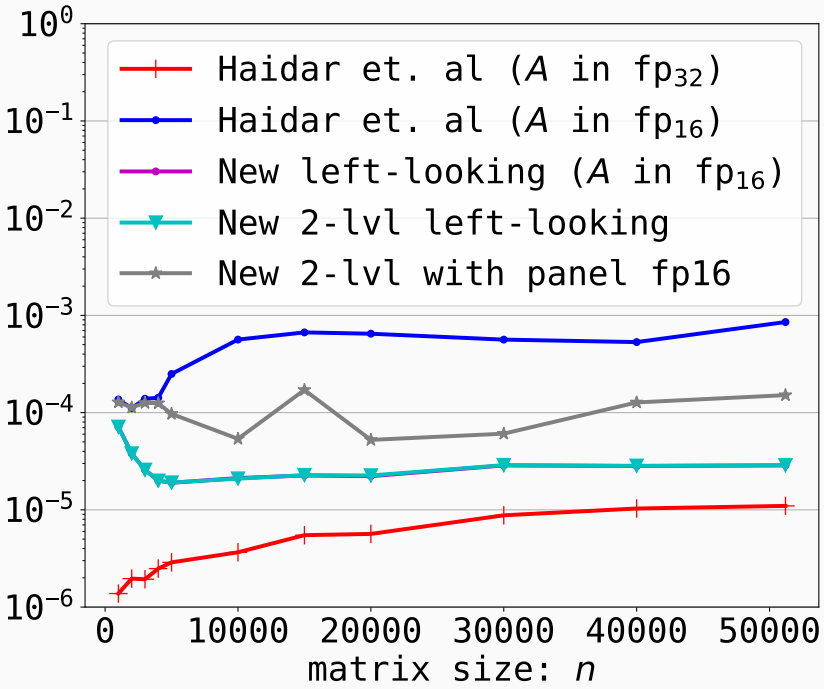


Performance (TFlop/s)

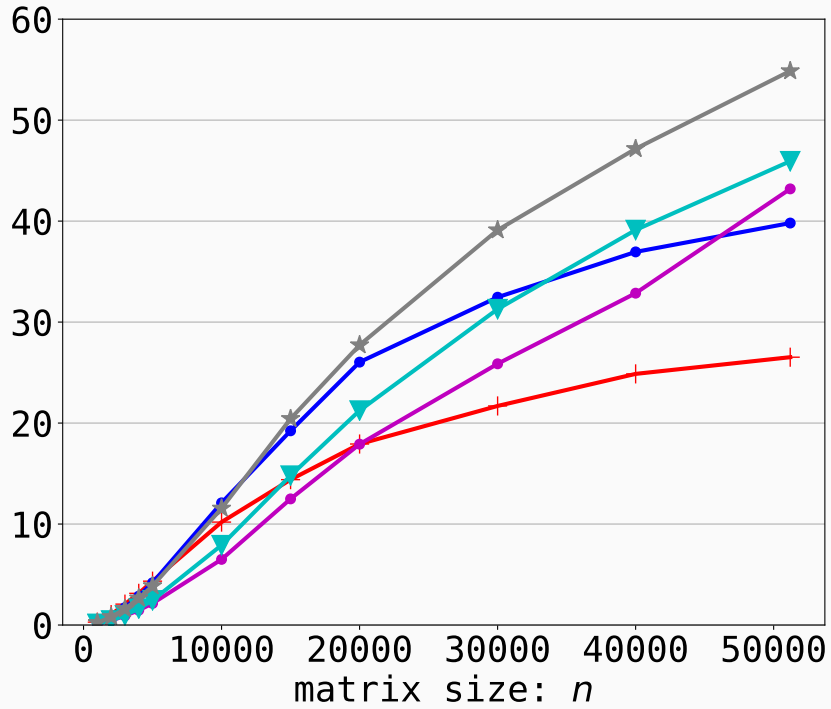
New doubly partitioned left-looking algorithm

Backward error bounds

Haidar et al.	New left-looking		
	1-lvl	2-lvl, fp32 inner	2-lvl, fp16 inner
$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$	$2u_{16} + nu_{32}$	$su_{16} + nu_{32}$



Backward error

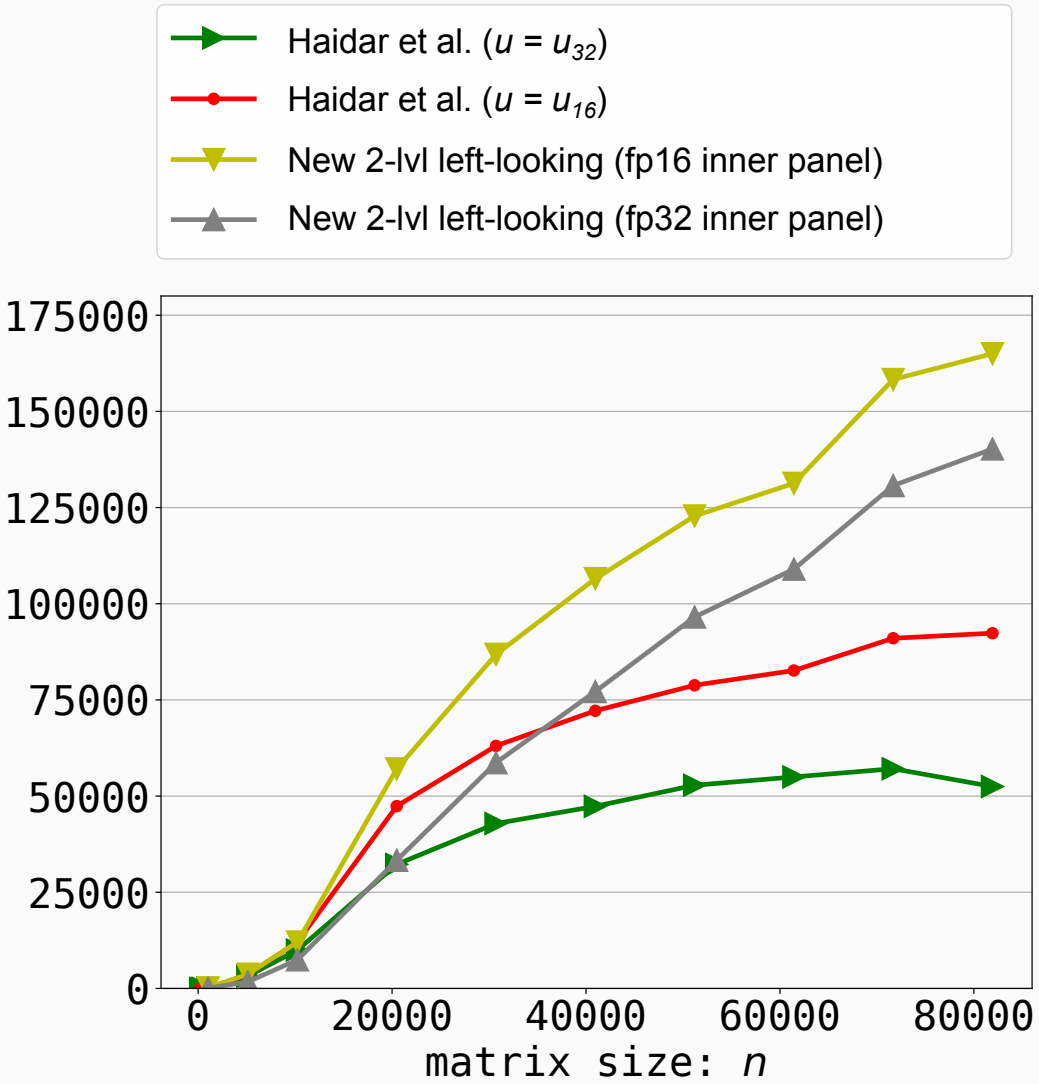


Performance (TFlop/s)

Inner panel factorization in fp32 or fp16?

$\Rightarrow$ **no clear winner**, different tradeoffs

Performance on A100



Up to 3.5× faster (170 TFLOPS)

Impact on iterative refinement

- LU-based IR with mixed precision LU factorization (using tensor cores) and **fp64 target accuracy**
- Matrices from Fasi & Higham[3] with specified $\kappa(A)$

LU fact. algorithm	LU fact. TFLOPS	$\kappa(A) = 100$ #iter	$Ax = b$ TFLOPS
Haidar et al. ($u = u_{32}$)	48.1	4	27.4
Haidar et al. ($u = u_{16}$)	77.9	6	29.1
New (fp32 inner)	82.9	4	35.7
New (fp16 inner)	118.1	4	**41.8**

[3] Fasi & Higham, *Matrices with tunable infinity-norm condition number and no need for pivoting in LU factorization*, SIMAX 2021.

Impact on iterative refinement

- LU-based IR with mixed precision LU factorization (using tensor cores) and **fp64 target accuracy**
- Matrices from Fasi & Higham[3] with specified $\kappa(A)$

LU fact. algorithm	LU fact. TFLOPS	$\kappa(A) = 100$ #iter	$\kappa(A) = 100$ $Ax = b$ TFLOPS	$\kappa(A) = 1000$ #iter	$\kappa(A) = 1000$ $Ax = b$ TFLOPS
Haidar et al. ($u = u_{32}$)	48.1	4	27.4	5	25.5
Haidar et al. ($u = u_{16}$)	77.9	6	29.1	43	6.7
New (fp32 inner)	82.9	4	35.7	5	**32.8**
New (fp16 inner)	118.1	4	**41.8**	7	30.2

More accurate factorization $\Rightarrow$ IR converges faster!

Up to 5× speedup

[3] Fasi & Higham, *Matrices with tunable infinity-norm condition number and no need for pivoting in LU factorization*, SIMAX 2021.

Conclusions and future work

New mixed precision LU factorization based on

- left-looking algorithm with fp32 buffers of controlled size
- doubly partitioned mixed precision panel factorization with tensor cores

Compared with standard algorithm with LU factors in fp32:

- Half the memory footprint
- Half the data movement
- Up to 2× faster on V100 and 3.5× faster on A100

Compared with standard algorithm with LU factors in fp16:

- Significantly more accurate ⇒ up to 5× faster convergence with IR

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)