Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance

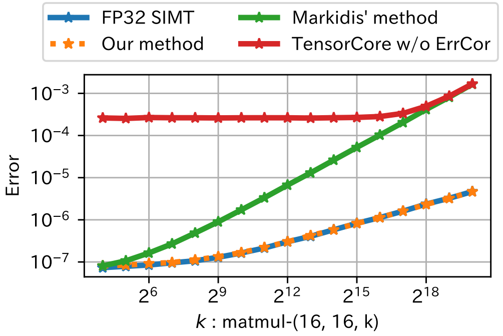
Hiroyuki Ootomo and Rio Yokota

ECP multiprecision meeting, April 21

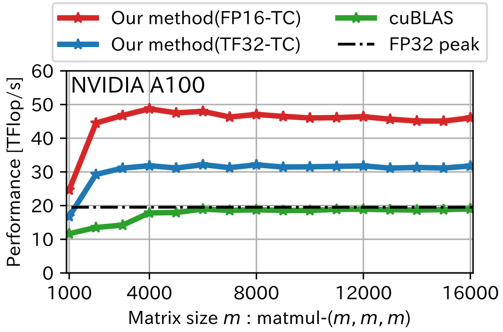
Achievements of this work

Our SGEMM emulation on Tensor Cores outperforms the theoretical peak performance of FP32 SIMT Core while achieving the same level of accuracy.

Accuracy



Throughput



Paper : `https://arxiv.org/abs/2203.03341`

Related work

1. Markidis *et al.* proposed a SGEMM emulation method on Tensor Cores.
   NVIDIA Tensor Core Programmability, Performance & Precision,
   `https://arxiv.org/abs/1803.04014`
   However, their method does not fit the accuracy of FP32 SIMT Core.
2. Feng *et al.* claimed to fix the accuracy of Markidis' method by recovering the mantissa length kept by the method.
   EGEMM-TC: accelerating scientific computing on tensor cores with extended precision, PPoPP'21
   Is it true? We have also investigated the mantissa length problem and concluded that it is not the problem of Markidis' method.

Related work : Markidis' method

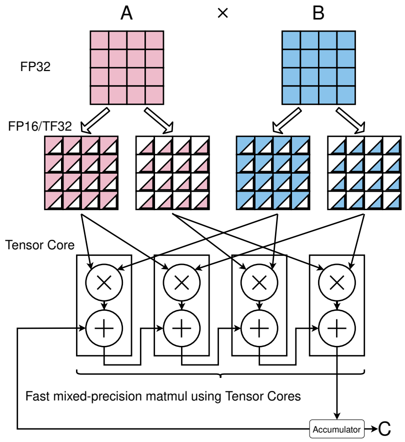
Compute a multiplitation of FP32 matrices $\mathbf{A}_{\mathsf{F32}}$ and $\mathbf{B}_{\mathsf{F32}}$.

1. Split one FP32 matrix into two FP16 matrices

$$\mathbf{M}_{\mathsf{F16}} \leftarrow \mathsf{toF16}\left(\mathbf{M}_{\mathsf{F32}}\right)$$
$$\Delta\mathbf{M}_{\mathsf{F16}} \leftarrow \mathsf{toF16}\left(\mathbf{M}_{\mathsf{F32}} - \mathsf{toF32}\left(\mathbf{M}_{\mathsf{F16}}\right)\right)$$

2. Multiply and accumulate using Tensor Core

$$\begin{aligned}\mathbf{C} \leftarrow & \left(\mathbf{A}_{\mathsf{F16}} + \Delta\mathbf{A}_{\mathsf{F16}}\right)\left(\mathbf{B}_{\mathsf{F16}} + \Delta\mathbf{B}_{\mathsf{F16}}\right) \\ & \sim \mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \Delta\mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} \\ & + \mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}} + \Delta\mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}\end{aligned}$$

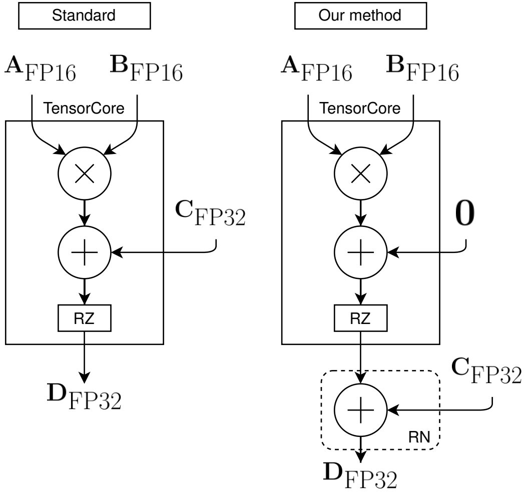


Related work

1. Markidis *et al.* proposed a SGEMM emulation method on Tensor Cores.
NVIDIA Tensor Core Programmability, Performance & Precision,
`https://arxiv.org/abs/1803.04014`
However, their method does not fit the accuracy of FP32 SIMT Core.
2. Feng *et al.* claimed to fix the accuracy of Markidis' method by recovering the mantissa length kept by the method.
EGEMM-TC: accelerating scientific computing on tensor cores with extended precision, PPoPP'21
Is it true? We have also investigated the mantissa length problem and concluded that it is not the problem of Markidis' method.

Contribution (1/2)

- We have found that the rounding for accumulator inside Tensor Cores – RZ – causes the low accuracy of Markidis' method.
- To avoid this rounding, we use FP32 SIMT Core for the accumulation outside of Tensor Cores.



Standard
$\mathbf{A}_{\mathrm{FP16}}$
$\mathbf{B}_{\mathrm{FP16}}$
TensorCore
$\mathbf{C}_{\mathrm{FP32}}$
RZ
$\mathbf{D}_{\mathrm{FP32}}$
Our method
$\mathbf{A}_{\mathrm{FP16}}$
$\mathbf{B}_{\mathrm{FP16}}$
TensorCore
$\mathbf{0}$
RZ
$\mathbf{C}_{\mathrm{FP32}}$
RN
$\mathbf{D}_{\mathrm{FP32}}$

Contributions (2/2)

- Improve the accuracy of Markidis' method
  1. Calculat expectation mantissa length
  2. Found the causes the low accuracy: rounding inside Tensor Core
  3. Develop a method to avoid this rounding.

4. Reduce the underflow probability during the error correction by scaling error correction terms
5. Reduce computational complexity by omitting negligible error correction step
6. Demonstrate that our method outperforms the FP32 SIMT Core peak performance and consumes lower consumption while the the same level accuracy.

(1/6) The expectation mantissa length of Markidis' method

[Markidis' method] $\mathbf{M}_{\mathsf{F32}} \sim \left(\mathbf{M}_{\mathsf{F16}} + \Delta\mathbf{M}_{\mathsf{F16}}\right)$ where
$\mathbf{M}_{\mathsf{F16}} \leftarrow \mathrm{toF16}\left(\mathbf{M}_{\mathsf{F32}}\right), \Delta\mathbf{M}_{\mathsf{F16}} \leftarrow \mathrm{toF16}\Big(\mathbf{M}_{\mathsf{F32}} - \mathrm{toF32}\left(\mathbf{M}_{\mathsf{F16}}\right)\Big)$
Feng *et al.* claimed that the Markidis' method can only keep 20 per 23 bits of the FP32 mantissa and this is the main cause of the low accuracy of the method.

Is it true?

Answer from our investigation

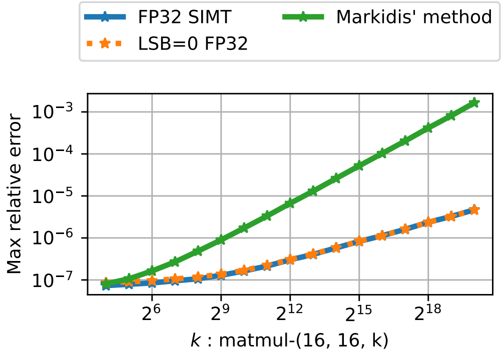
No.
We found that the expectation mantissa length kept by Markidis' method is 22.75 per 23 bit.
Furthermore, we showed that this 0.25 bits of mantissa loss is not the main cause of the low accuracy of Markidis' method.

(1/6) The expectation mantissa length of Markidis' method

To show that the mantissa loss is not the main cause of the low accuracy of Markidis' method, we conducted a small experiment.

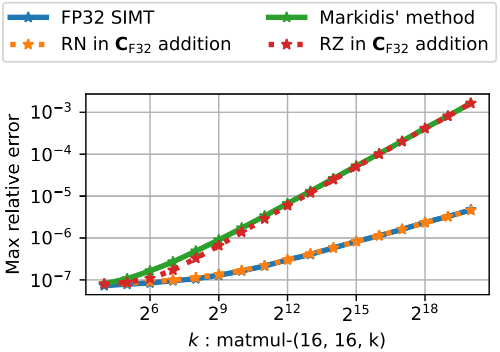
- Compared Markidis' method to a preprocessed SGEMM, which sets the LSB of mantissa to zero.
- By this preprocess, the expectation of mantissa length becomes 22.5 bits ($<$ 22.75 bits)
- The accuracy of Markidis' method is worse than this preprocessed SGEMM.

$\Rightarrow$ The 0.25 bits of mantissa loss is not the (main) cause of the low accuracy of Markidis' method.



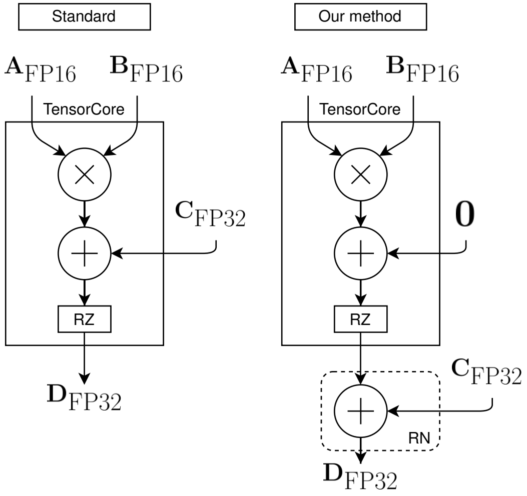
(2/6) Cause of Low Accuracy: Rounding Inside Tensor Core

- Made two types of Tensor Cores emulators which compute $\mathbf{A}_{\text{FP16}}\mathbf{B}_{\text{FP16}} + \mathbf{C}_{\text{FP32}}$ using
  1. RN in $\mathbf{C}_{\text{FP32}}$ addition.
  2. RZ in $\mathbf{C}_{\text{FP32}}$ addition.
- The accuracy of Markidis' method using "RZ in $\mathbf{C}_{\text{FP32}}$ addition" is similar to the real Tensor Cores.



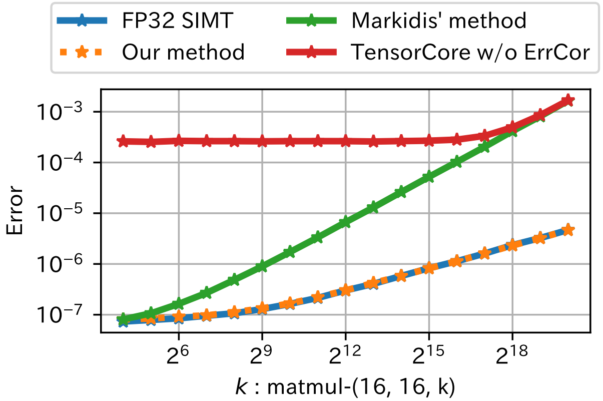
(3/6) Avoiding the Rounding Inside Tensor Core

- We use FP32 SIMT adder to compute the $\mathbf{C}_{\mathrm{FP32}}$ addition using RN.
- This method avoids the **direct** RZ for $\mathbf{C}_{\mathrm{FP32}}$ addition.


Standard
$\mathbf{A}_{\mathrm{FP16}}$
$\mathbf{B}_{\mathrm{FP16}}$
TensorCore
$\mathbf{C}_{\mathrm{FP32}}$
RZ
$\mathbf{D}_{\mathrm{FP32}}$
Our method
$\mathbf{A}_{\mathrm{FP16}}$
$\mathbf{B}_{\mathrm{FP16}}$
TensorCore
$\mathbf{0}$
RZ
$\mathbf{C}_{\mathrm{FP32}}$
RN
$\mathbf{D}_{\mathrm{FP32}}$

(3/6) Avoiding the Rounding Inside Tensor Core

With this approach, we improved the accuracy of Markidis' method.



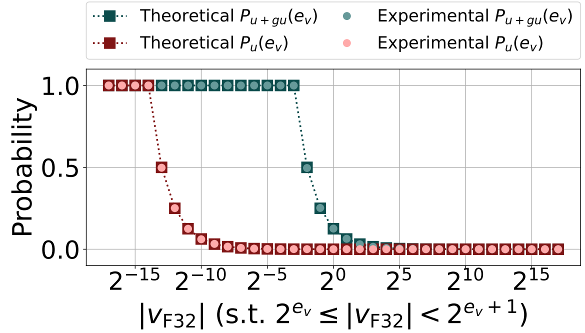
(4/6) Reducing Underflow Probability with Scaling

One computation underflows with high probability in Markidis' method.

$$\mathbf{M}_{\mathrm{F16}} \leftarrow \mathrm{toF16}\left(\mathbf{M}_{\mathrm{F32}}\right)$$

$$\Delta\mathbf{M}_{\mathrm{F16}} \leftarrow \underbrace{\mathrm{toF16}\left(\mathbf{M}_{\mathrm{F32}} - \mathrm{toF32}\left(\mathbf{M}_{\mathrm{F16}}\right)\right)}_{\text{This computation}}$$

We have investigated the underflow and gradual underflow probabilities.



- P_{u+gu} :
  The underflow and gradual underflow probability.
- P_u :
  The underflow probability.

(4/6) Reducing Underflow Probability with Scaling

- Markidis' method

$$\begin{aligned}
\mathbf{M}_{\mathsf{F16}} &\leftarrow \mathsf{toF16}\left(\mathbf{M}_{\mathsf{F32}}\right) \\
\Delta\mathbf{M}_{\mathsf{F16}} &\leftarrow \mathsf{toF16}\left(\mathbf{M}_{\mathsf{F32}} - \mathsf{toF32}\left(\mathbf{M}_{\mathsf{F16}}\right)\right) \\
\mathbf{C} &\sim \mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \Delta\mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}} + \Delta\mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}
\end{aligned}$$

- Our prototype method scales the $\Delta\mathbf{M}_{\mathsf{F16}}$ computation.

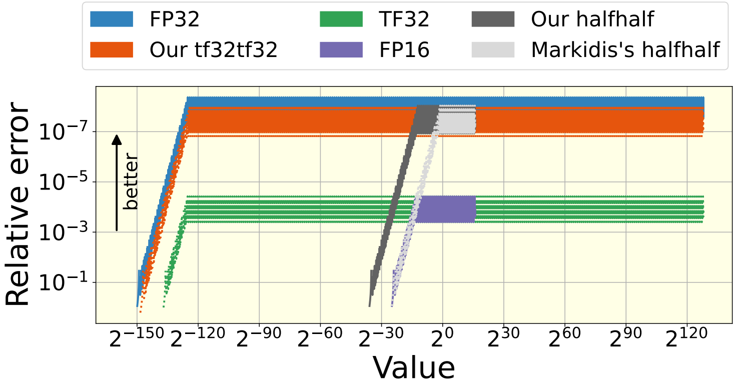
$$\begin{aligned}
\mathbf{M}_{\mathsf{F16}} &\leftarrow \mathsf{toF16}\left(\mathbf{M}_{\mathsf{F32}}\right) \\
\Delta\mathbf{M}_{\mathsf{F16}} &\leftarrow \mathsf{toF16}\left(\left(\mathbf{M}_{\mathsf{F32}} - \mathsf{toF32}\left(\mathbf{M}_{\mathsf{F16}}\right)\right) \times 2^{11}\right) \\
\mathbf{C} &\sim \mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \left(\Delta\mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}\right) / 2^{11} + \Delta\mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}} / 2^{11\times 2}
\end{aligned}$$

The "11" comes from the mantissa length of FP16 with implicit one bit, $10 + 1 = 11$.

(4/6) Reducing Underflow Probability with Scaling

The comparison of representation accuracy and range.

- **Markidis' halfhalf** : Represents one FP32 using two FP16s
- **Our halfhalf** : Markidis' halfhalf + scaling.
- **Our tf32tf32** : Our halfhalf using TF32 (e8m10) instead of FP16.



(5/6) Reducing computational complexity

- Our prototype method

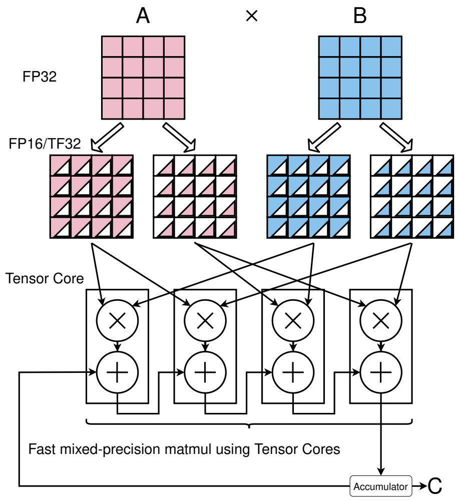
$$\mathbf{C} \sim \mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \left(\Delta\mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}\right)/2^{11} \underset{\sim\sim\sim}{+\Delta\mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}/2^{11\times 2}}$$

- Our final method omits "$\underset{\sim\sim\sim}{+\Delta\mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}/2^{11\times 2}}$" since the effect of this error correction computation is negligible to the FP32 23 bits of mantissa.

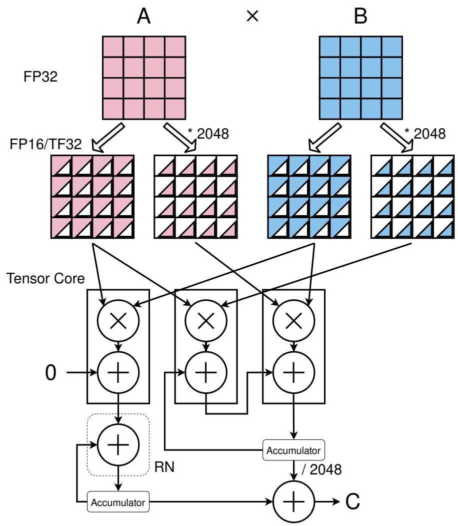
$$\mathbf{C} \sim \mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \left(\Delta\mathbf{A}_{\mathsf{F16}}\mathbf{B}_{\mathsf{F16}} + \mathbf{A}_{\mathsf{F16}}\Delta\mathbf{B}_{\mathsf{F16}}\right)/2^{11}$$

Summary: Comparison of our method and Markidis' method

Markidis' method



Our method



(6/6) Evaluation

We have incorporated our method into NVIDIA CUTLASS 2.5.[1] to use the high performance functions, e.g. memory blocking strategies and thread allocations.

Labels

- `cutlass_halfhalf` : Uses FP16 Tensor Cores
- `cutlass_tf32tf32` : Uses TF32 Tensor Cores

Evaluation environment

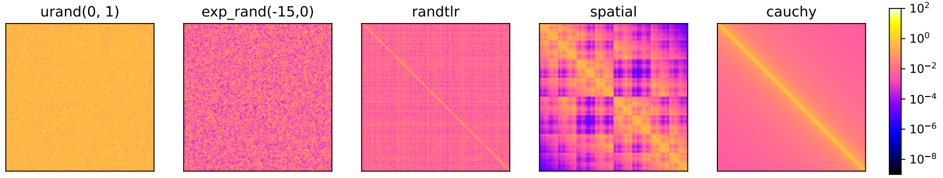
- NVIDIA A100 40GB SXM4
- CUDA 11.3

[1]NVIDIA includes 3xTF32 SGEMM emulation into CUTLASS 2.8 independently from us.

Accuracy evaluation (1/2)

We evaluate our implementation using various exponent distribution matrices.

Test matrices



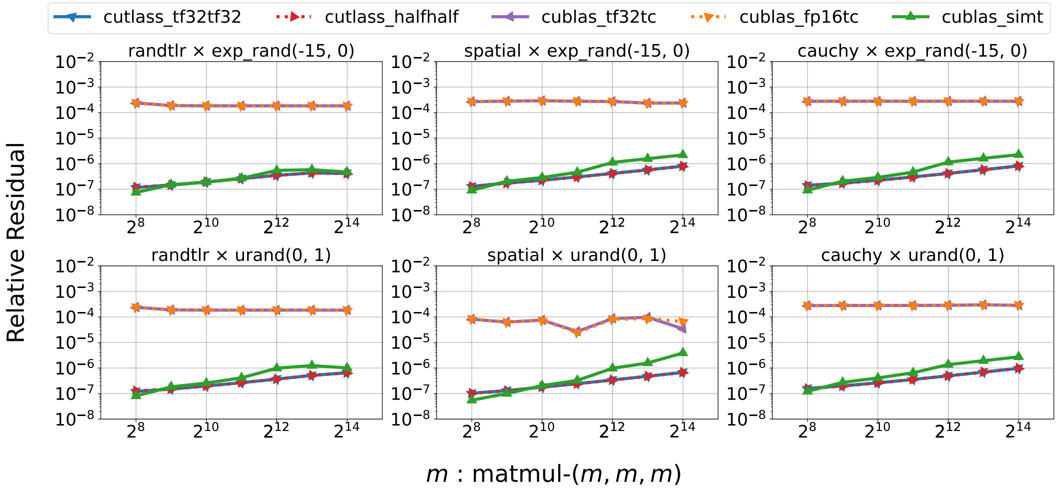
Error evaluation

$$\text{RelativeResidual} = ||\mathbf{C}_{\text{ref}} - \mathbf{C}||_F / ||\mathbf{C}_{\text{ref}}||_F,$$

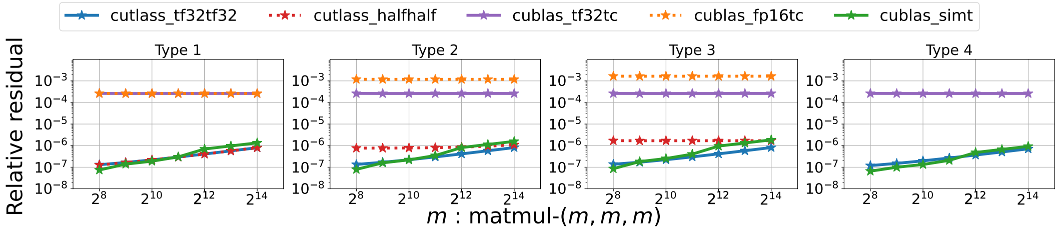
where $||\cdot||_F$ is Frobenius norm and $\mathbf{C}_{\text{ref}}$ is a reference computation result in FP64.

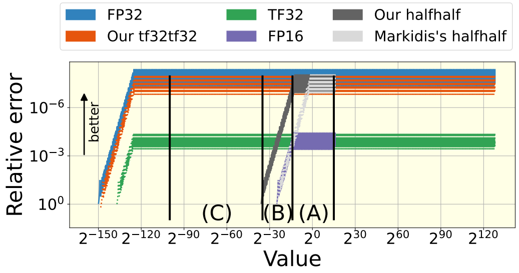
Accuracy evaluation (1/2)

Test matrices



Accuracy evaluation (2/2)





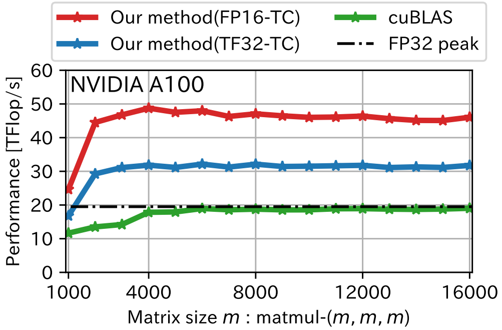
Type 1 : (A) $\times$ (A)

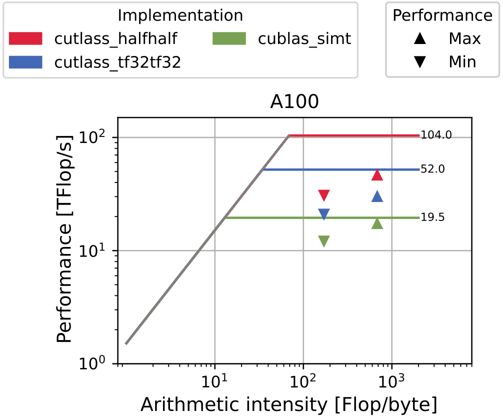
Type 2 : (A) $\times$ (B)

Type 3 : (B) $\times$ (B)

Type 4 : (A) $\times$ (C), (B) $\times$ (C), (C) $\times$ (C)

Throughput evaluation

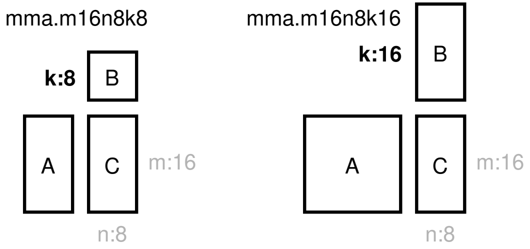




- We have also measured the power consumption of each method and found thet our method consumes lower power compared to cuBLAS SGEMM.

To improve the throughput of our implementations...

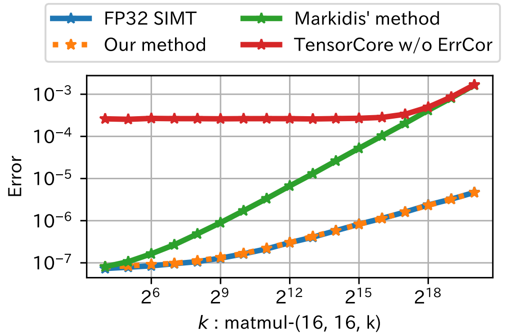
1. Reduce the shared memory bank conflict. Currently, the shared memory layout (skew) is not suitable to this error correction method and a lot of shared memory bank conflicts occur.
2. Use `mma.m16n8k16` instruction instead of `mma.m16n8k8` for `cutlass_halfhalf`. By using this instruction, we can reduce the latency. However, `mma.m16n8k16` has an additional RZ between first half 8 accumulations and latter half 8 accumulations. We need to carefully investigate the effect.



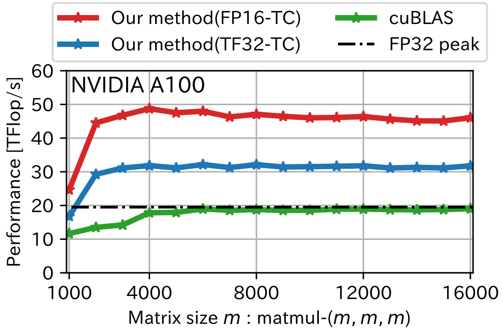
Conclusion

We have improved Markidis' error correction method and demonstrated that our SGEMM emulation on Tensor Cores outperforms the theoretical peak performance of FP32 SIMT Core while achieving the same level of accuracy.

Accuracy



Throughput



Open problem

1. We haven't done the theoretical error analysis of avoiding RZ in our method. Although the RZ for adding $\mathbf{C}_{\text{FP32}}$ is avoided, the rounding for $\mathbf{A}_{\text{FP16}} \cdot \mathbf{B}_{\text{FP16}}$ is still RZ.