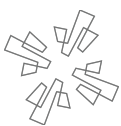
TSQR on TensorCores with error correction

Hiroyuki Ootomo, Rio Yokota

2021.03.03



Overview

Motivation

QR factorization is often used in many scientific applications. We want fast QR factorization implementation for a tall skinny matrix.

Take Home Message

- By using correction technique, we can do QR factorization efficiently without much loss of accuracy.
- We encounter two problems on error correction:
  - Floating point number specification.
  - Computation inside TensorCores.

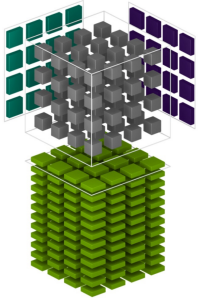


Image : https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/

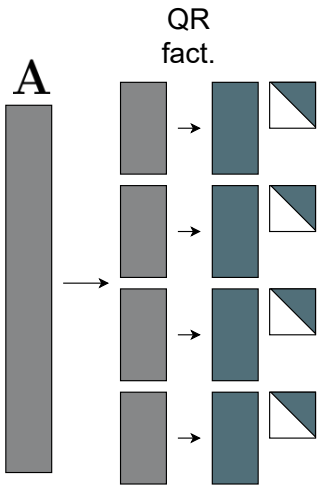
TSQR (Tall-Skinny QR)

Efficient QR factorization algorithm for tall skinny matrix on parallel computing environment

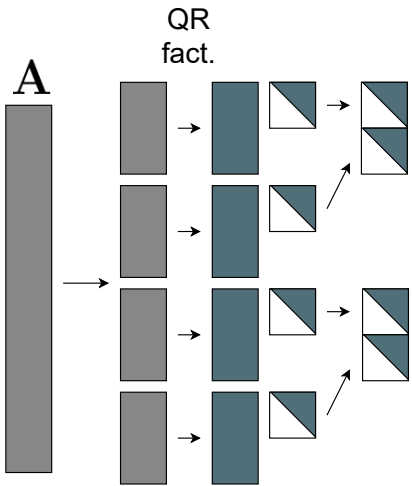
Applications

- Block iterative linear equation solver
- Block iterative eigensolver

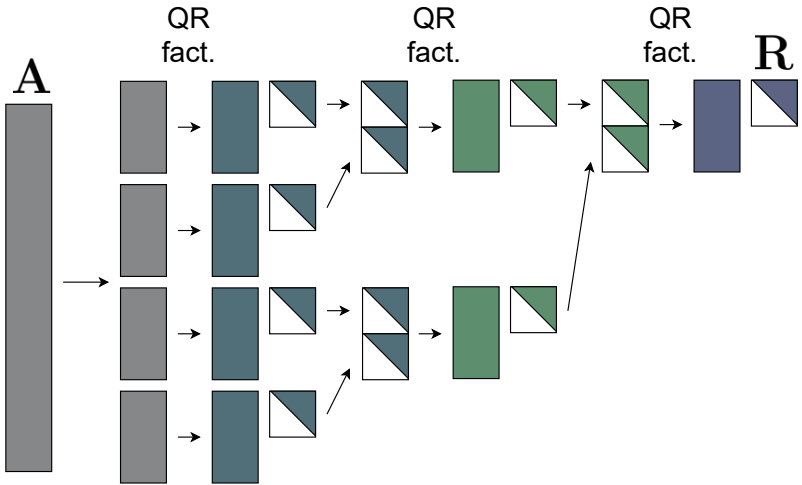
TSQR



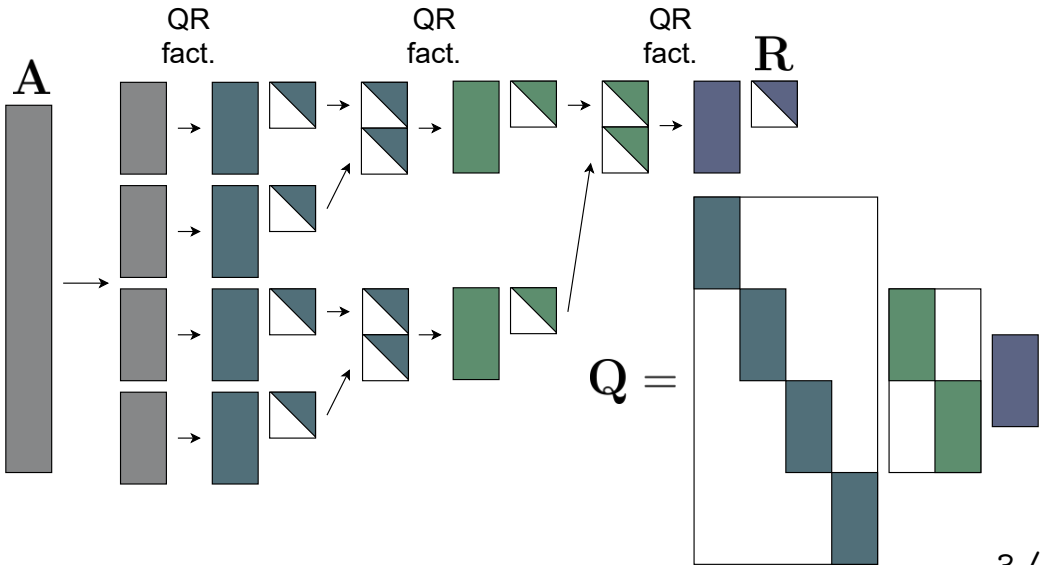
TSQR



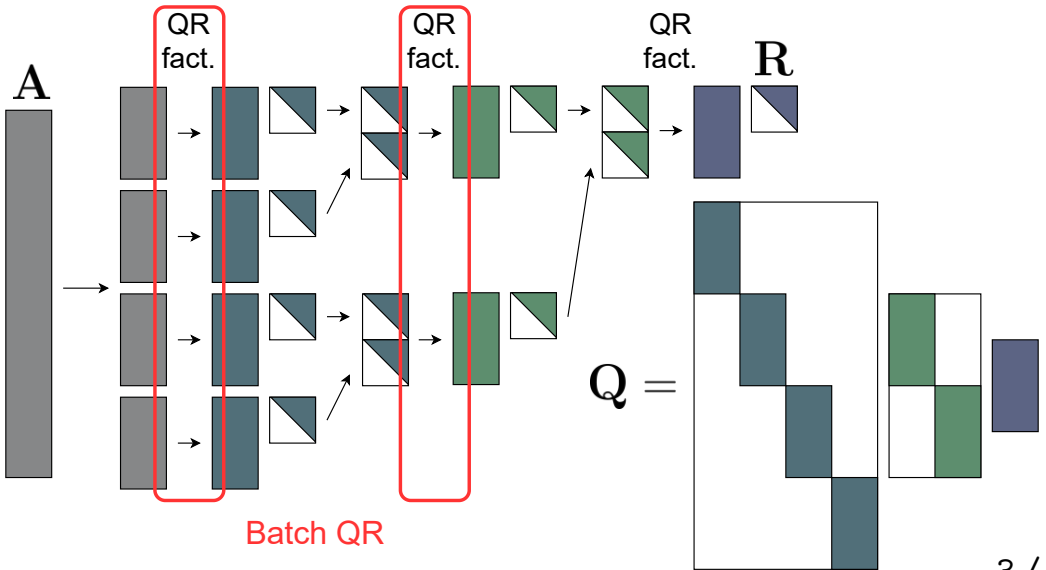
TSQR



TSQR



TSQR



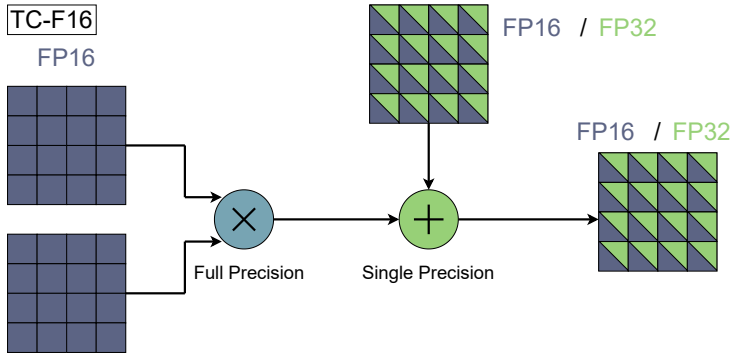
TSQR implementation using Batch QR

Compute each QR factorization in parallel using Batch QR.

Implementation of BatchQR

- The size of each input matrix is equal or smaller than 256×128
- Householder QR + WY Representaion
- Compute all GEMMs and GEMVs using TensorCore
- 1 SM computes 1 QR factorization

TensorCore



Performance

	TC-FP16 [TFlop/s]	TC-TF32 [TFlop/s]
TESLA V100	125	-
A100	312	156

TF32

- 8 bit exponent
- 10 bit mantissa

Experiment

Experiment setup

Compare TSQR on TensorCore **without** error correction and cuSOLVER QR factorization

Implementation of TSQR

- Input matrix size is $m \times n$, $n \leq 128$.
- Single precision (FP32)

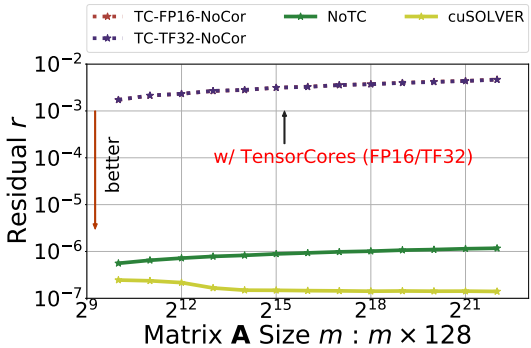
ModeName	TensorCore	ErrorCorrection
FP16_NoCor	FP16	No
TF32_NoCor	TF32	No
FP32_NoTC	No	No

Evaluation of accuracy without error correction

- Input $\mathbf{A} \leftarrow$ uniform[-1, 1]$^{m \times 128}$
- NVIDIA A100 SXM4
- CUDA 11.2

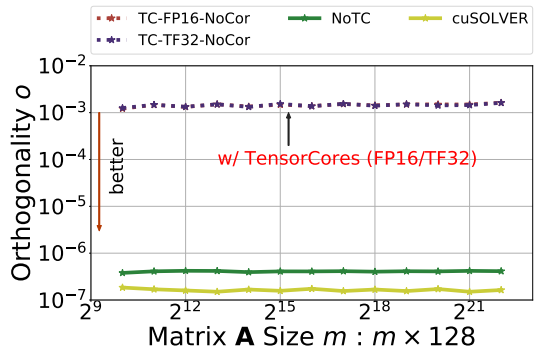
Residual

- $r = ||\mathbf{A} - \mathbf{QR}||_F / ||\mathbf{A}||_F$



Orthogonality

- $o = ||\mathbf{Q}^\top \mathbf{Q} - \mathbf{I}||_F / \sqrt{n}$



Matmul $\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP32}}\mathbf{B}_{\mathrm{FP32}}$ w/ error correction

$\mathbf{M}_{\mathrm{FP16}} \xleftarrow{\text{narrowing}} \mathbf{M}_{\mathrm{FP32}}$

FP16 FP32

Without correction

$\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP16}}\ \mathbf{B}_{\mathrm{FP16}}$

Matmul $\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP32}}\mathbf{B}_{\mathrm{FP32}}$ w/ error correction

$$\mathbf{M}_{\mathrm{FP16}} \xleftarrow{\text{narrowing}} \mathbf{M}_{\mathrm{FP32}}$$

FP16 FP32

Without correction

$$\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP16}}\ \mathbf{B}_{\mathrm{FP16}}$$

With correction

$$\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP16}}\ \mathbf{B}_{\mathrm{FP16}} + \underbrace{\left(\Delta\mathbf{A}_{\mathrm{FP16}}\ \mathbf{B}_{\mathrm{FP16}} + \mathbf{A}_{\mathrm{FP16}}\ \Delta\mathbf{B}_{\mathrm{FP16}}\right)/2^{10}}_{\text{Correction terms}}$$

where $\Delta\mathbf{M}_{\mathrm{FP16}} \xleftarrow{\text{narrowing}} \left(\mathbf{M}_{\mathrm{FP32}} - \mathbf{M}_{\mathrm{FP16}}\right) \times 2^{10}$

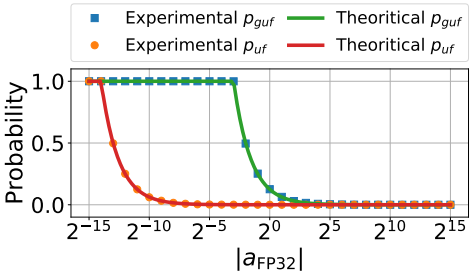
Why 2^{10} scaling?

$\Delta\mathbf{M}_{\mathrm{FP16}} \xleftarrow{\text{narrowing}} \mathbf{M}_{\mathrm{FP32}} - \mathbf{M}_{\mathrm{FP16}}$ underflow can be observed.
Multiplying by 2^{10} can prevent it, where 10 is a mantissa length of FP16.

Underflow probability

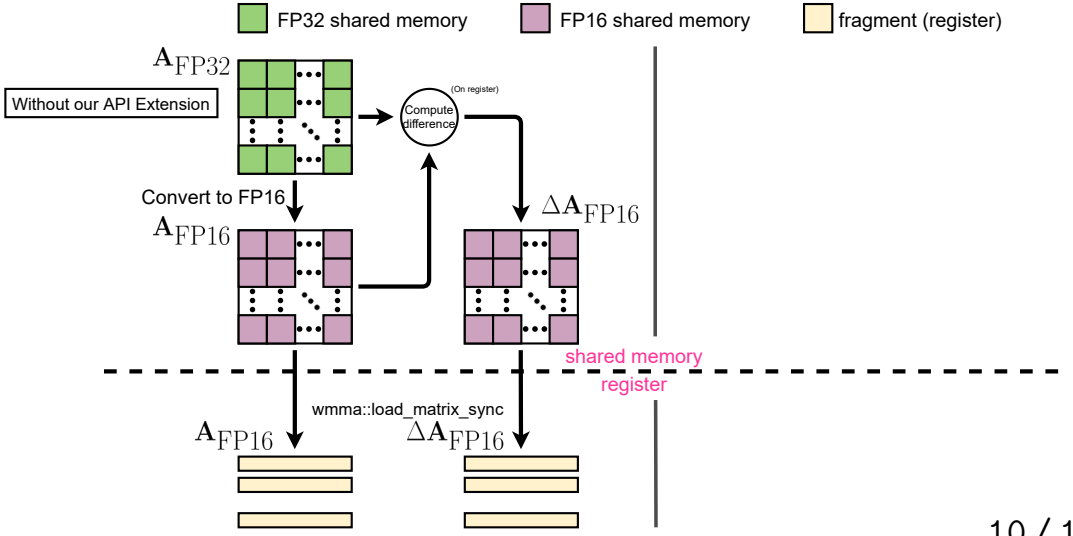
- Compute $\Delta\mathbf{a}_{\mathrm{FP16}} \xleftarrow{\text{narrowing}} \mathbf{a}_{\mathrm{FP32}} - \mathbf{a}_{\mathrm{FP16}}$

- p_{uf} :
  Underflow probability of $\Delta\mathbf{a}_{\mathrm{FP16}}$
- p_{guf} :
  p_{uf} + gradual underflow probability



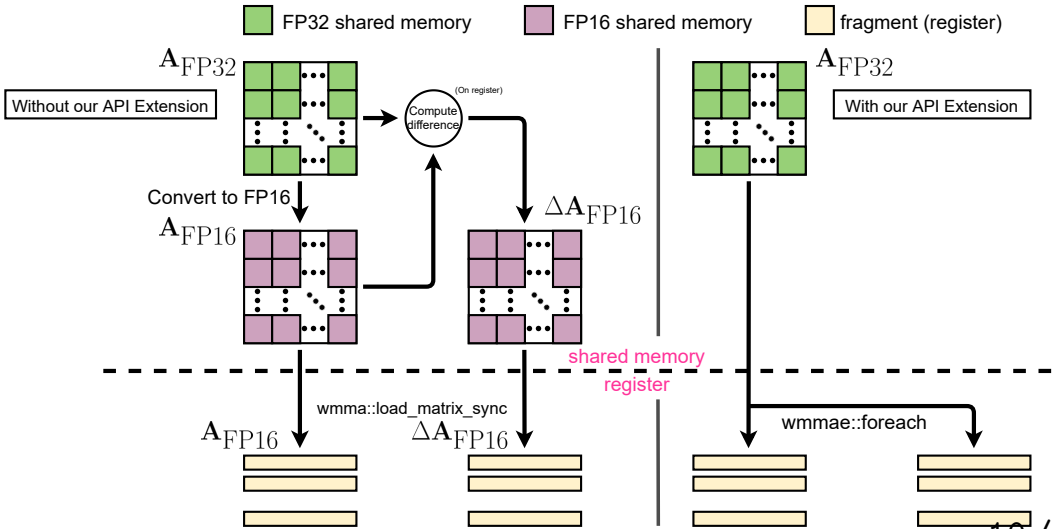
WMMA API (TesnorCore API) Extension

- Reduce shared memory footprint



WMMA API (TesnorCore API) Extension

- Reduce shared memory footprint



Experiment

Experiment setup

Compare TSQR on TensorCore with error correction and cuSOLVER QR factorization

Implementation of TSQR

- Input matrix size is $m \times n$, $n \leq 128$.
- Single precision (FP32)

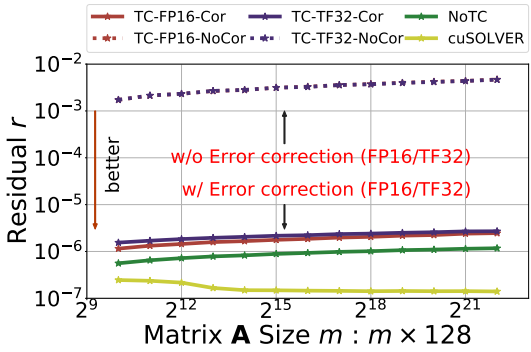
ModeName	TensorCore	ErrorCorrection
FP16_Cor	FP16	Yes
FP16_NoCor	FP16	No
TF32_Cor	TF32	Yes
TF32_NoCor	TF32	No
FP32_NoTC	No	No

Evaluation accuracy

- Input $\mathbf{A} \leftarrow \text{uniform}[-1, 1]^{m \times 128}$
- NVIDIA A100 SXM4
- CUDA 11.2

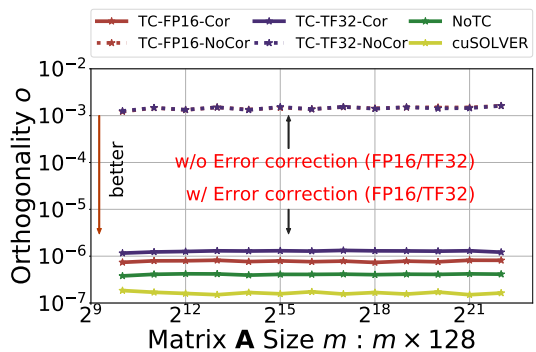
Residual

- $r = ||\mathbf{A} - \mathbf{QR}||_F / ||\mathbf{A}||_F$



Orthogonality

- $o = ||\mathbf{Q}^\top \mathbf{Q} - \mathbf{I}||_F / \sqrt{n}$



Why the accuracy is not good compared to FP32_NoTC?

Mantissa length of FP32(24bit) > mantissa length of FP16 x2 (22bit)

Up to 2 bits of loss occurs.

The exponent length of FP16 is smaller than FP32

When a_{FP32} is smaller than minimum value of FP16,

- $a_{\mathrm{FP16}} = \mathrm{FP16}(a_{\mathrm{FP32}}) = 0$
- $\Delta a_{\mathrm{FP16}} = \mathrm{FP16}(a_{\mathrm{FP32}} - \mathrm{FP32}(a_{\mathrm{FP16}}))$.

It can only hold 11 bits of mantissa at most.

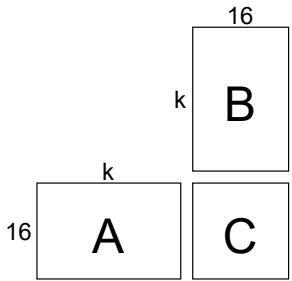
Why the accuracy is not good compared to FP32_NoTC?

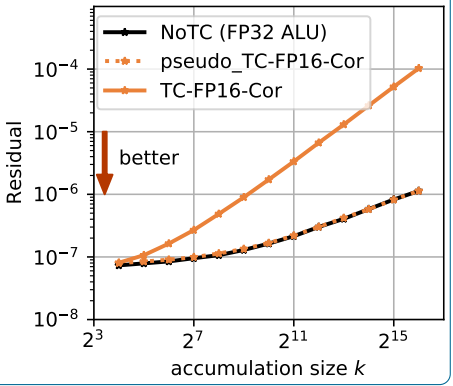
- TensorCore uses RZ

Evaluation of matmul

pseudo_TC : $\mathrm{FP32}(\mathbf{A}_{\mathrm{FP16}}) \times \mathrm{FP32}(\mathbf{B}_{\mathrm{FP16}}) + \mathrm{FP32}(\mathbf{C}_{\mathrm{FP16}})$ on FP32 ALU

- Using pseudo_TC with error correction
- $\mathbf{A}_{i,j}, \mathbf{B}_{i,j} \Leftarrow \mathrm{Uniform}[-1.\mathrm{f}, 1.\mathrm{f}]$

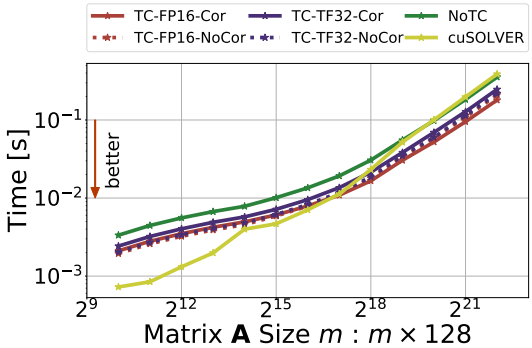




Evaluation performance

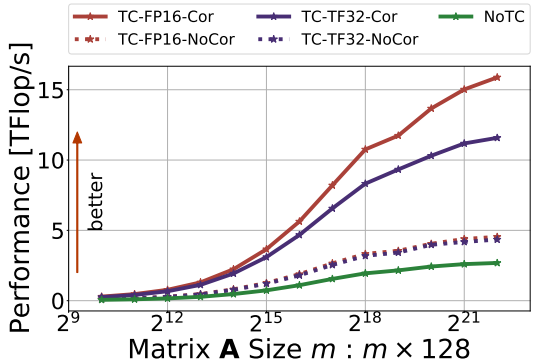
- Input $\mathbf{A} \leftarrow \text{uniform}[-1, 1]^{m \times 128}$
- NVIDIA A100 SXM4
- CUDA 11.2

Time



- We used latest cuSOLVER version

Performance

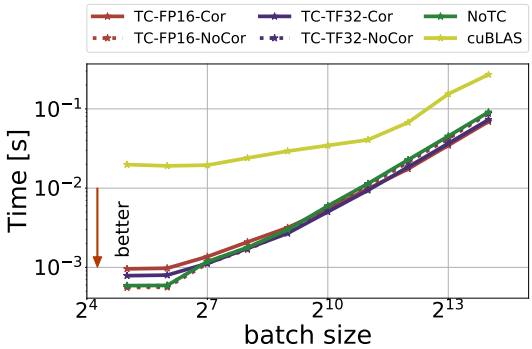


- Error correction needs 3x ops.

Evaluation performance (Batch QR of 256 x 128 matrices)

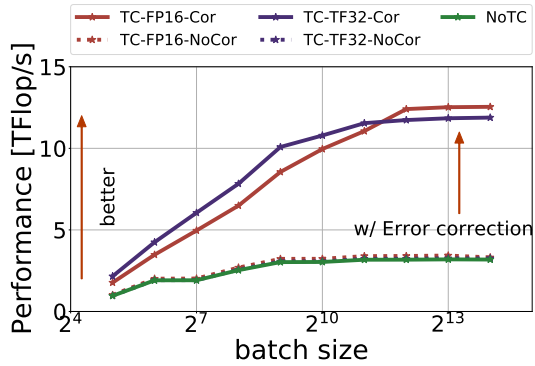
- $\mathbf{A}_i \leftarrow \text{uniform}[-1, 1]^{256\times128}$
- NVIDIA A100 SXM4
- CUDA 11.2

Time



- `cublasSgeqrfBatched`
- Memory bandwidth

Performance



- Error correction needs 3x ops.

Conclusion

Conclusion

- Using TensorCores and correction technique, we can calculate TSQR efficiently without much loss of accuracy
- We face two problems, one is from the floating point number specification and the other one is from computation inside TensorCores

Source code

- TSQR on TensorCore : `https://github.com/enp1s0/tsqr-tc`
- WMMA API Extension : `https://github.com/wmmae/wmma_extension`
- HMMA.F32.F32 : `https://github.com/wmmae/hmma.f32.f32`