TSQR on TensorCores

Hiroyuki Ootomo, Rio Yokota

Department of Computer Science, Tokyo Institute of Technology

SC19 Denver
November 20, 2019

Tokyo Tech

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 TensorCores, we can do QR factorization fast
- By using correction technique, we can do QR factorization efficiently without much loss of accuracy

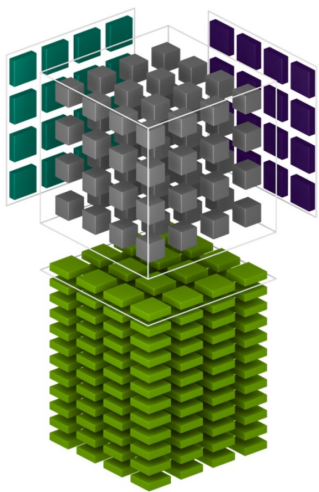
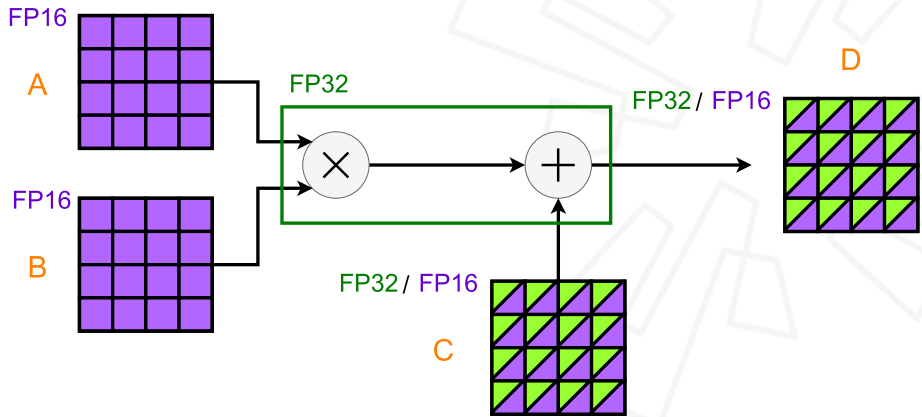


Image : https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/

TensorCore

- NVIDIA Volta & Turing Archinecture
- Mixed-precision matrix product circuit
- Peak performance of TensorCores $>$ 100 TFlop/s

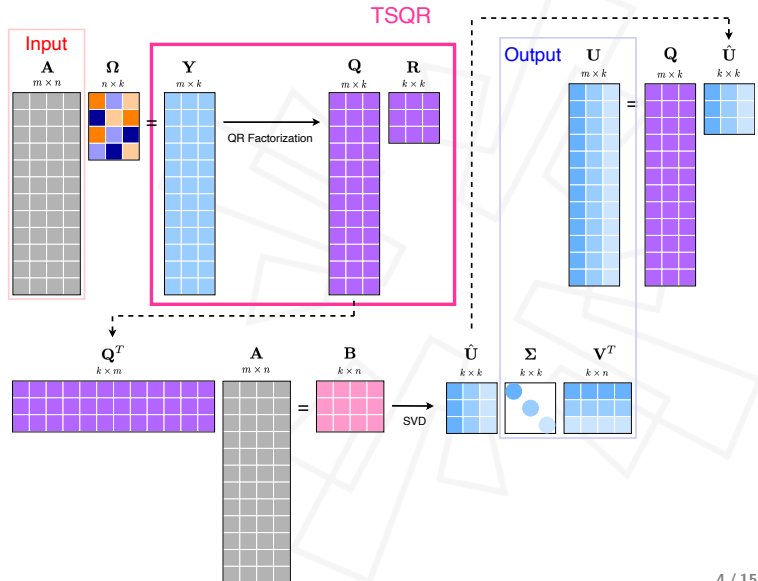


Low-rank approximation

Randomized SVD

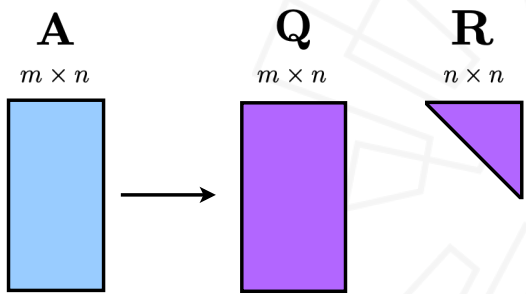
- Input $\mathbf{A} \in \mathbb{R}^{m \times n}$
- k-rank approximation
- $k \ll m, n$

1. $\mathbf{\Omega} \Leftarrow \mathrm{Rand}(n, k)$
2. $\mathbf{Y} \Leftarrow \mathbf{A} \times \mathbf{\Omega}$
3. $\mathbf{Q}, \_ \Leftarrow \mathrm{qr}(\mathbf{Y})$
4. $\mathbf{B} \Leftarrow \mathbf{Q}^T \times \mathbf{A}$
5. $\hat{\mathbf{U}}, \Sigma, \mathbf{V} \Leftarrow \mathrm{svd}(\mathbf{B})$
6. $\mathbf{U} \Leftarrow \mathbf{Q} \times \hat{\mathbf{U}}$



QR Factorization

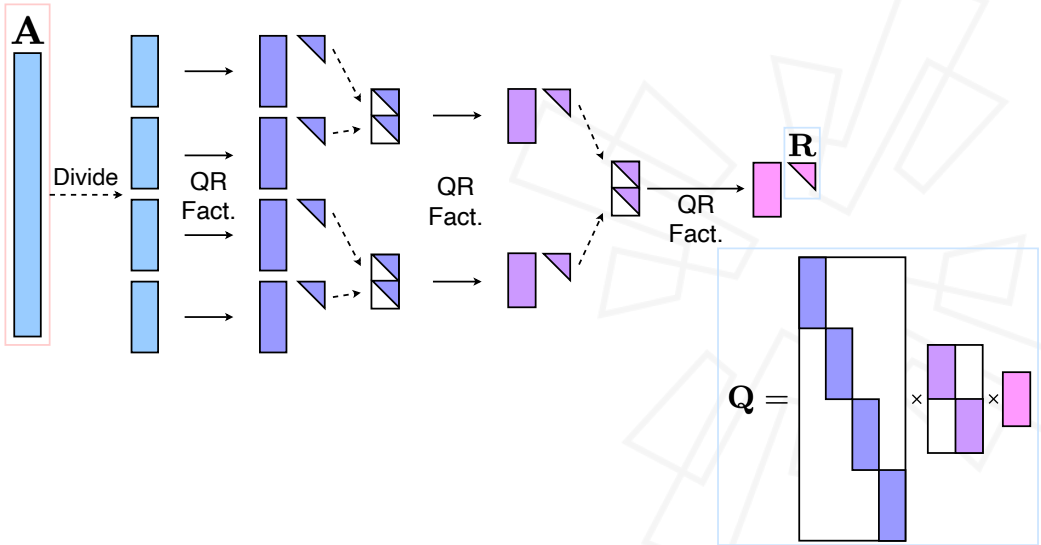
QR Factorization is a factorization of a matrix $\mathbf{A}$ into a product of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$



TSQR (Tall-Skinny QR)

Efficient algorithm for tall skinny matrix

TSQR



QR Factorization using Householder reflector

Algorithm QR Factorization using Householder reflector (Standard)

Require: $m, n \in \mathbb{N}, \mathbf{A} \in \mathbb{R}^{m \times n}$

Ensure: $\mathbf{Q} \in \mathbb{R}^{m \times m}, \mathbf{R} \in \mathbb{R}^{m \times n}$ s.t. $\mathbf{QR} = \mathbf{A}$

1: $\mathbf{Q}' \Leftarrow \mathbf{I}$, $\mathbf{R} \Leftarrow \mathbf{A}$
2: **for** $i \Leftarrow 0$ **to** $n - 1$ **do**
3: $\quad \mathbf{u} \Leftarrow [0 \ \cdots \ 0 \ \mathbf{R}_{i,i} \cdots \mathbf{R}_{m-1,i}]^T$
4: $\quad \mathbf{u}_i \Leftarrow \mathbf{u}_i \pm |\mathbf{u}|$
5: $\quad \mathbf{R} \Leftarrow \mathbf{R} - 2\mathbf{u}\left(\mathbf{u}^T\mathbf{R}\right)/||\mathbf{u}||^2$ // ←Outer, Mat-Vec product
6: $\quad \mathbf{Q} \Leftarrow \mathbf{Q} - 2\mathbf{u}\left(\mathbf{u}^T\mathbf{Q}\right)/||\mathbf{u}||^2$ // ←Outer, Mat-Vec product
7: **end for**
8: $\mathbf{Q} \Leftarrow \mathbf{Q}'^T$

QR Factorization using Householder reflector

Algorithm QR Factorization using Householder reflector (Improved)

Require: $m, n \in \mathbb{N}, \mathbf{A} \in \mathbb{R}^{m \times n}$

Ensure: $\mathbf{Q} \in \mathbb{R}^{m \times m}, \mathbf{R} \in \mathbb{R}^{m \times n}$ s.t. $\mathbf{QR} = \mathbf{A}$

1: $\mathbf{Q}' \Leftarrow \mathbf{I}$, $\mathbf{R} \Leftarrow \mathbf{A}$
2: **for** $i \Leftarrow 0$ **to** $n - 1$ **do**
3: $\quad \mathbf{u} \Leftarrow [0 \ \cdots \ 0 \ \mathbf{R}_{i,i} \cdots \mathbf{R}_{m-1,i}]^T$
4: $\quad \mathbf{u}_i \Leftarrow \mathbf{u}_i \pm |\mathbf{u}|$
5: $\quad \mathbf{H} \Leftarrow \mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{|\mathbf{u}|^2}$ // ←Using TensorCores
6: $\quad \mathbf{R} \Leftarrow \mathbf{HR}$ // ←Mat-Mat product using TensorCores
7: $\quad \mathbf{Q}' \Leftarrow \mathbf{HQ}'$ // ←Mat-Mat product using TensorCores
8: **end for**
9: $\mathbf{Q} \Leftarrow \mathbf{Q}'^T$

Matrix product correction using TensorCores

FP16 FP32

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

Without correction

$$\mathbf{Q}_{\text{FP32}} \leftarrow \mathbf{H}_{\text{FP16}}\ \mathbf{Q}_{\text{FP16}}$$

$$\mathbf{R}_{\text{FP32}} \leftarrow \mathbf{H}_{\text{FP16}}\ \mathbf{R}_{\text{FP16}}$$

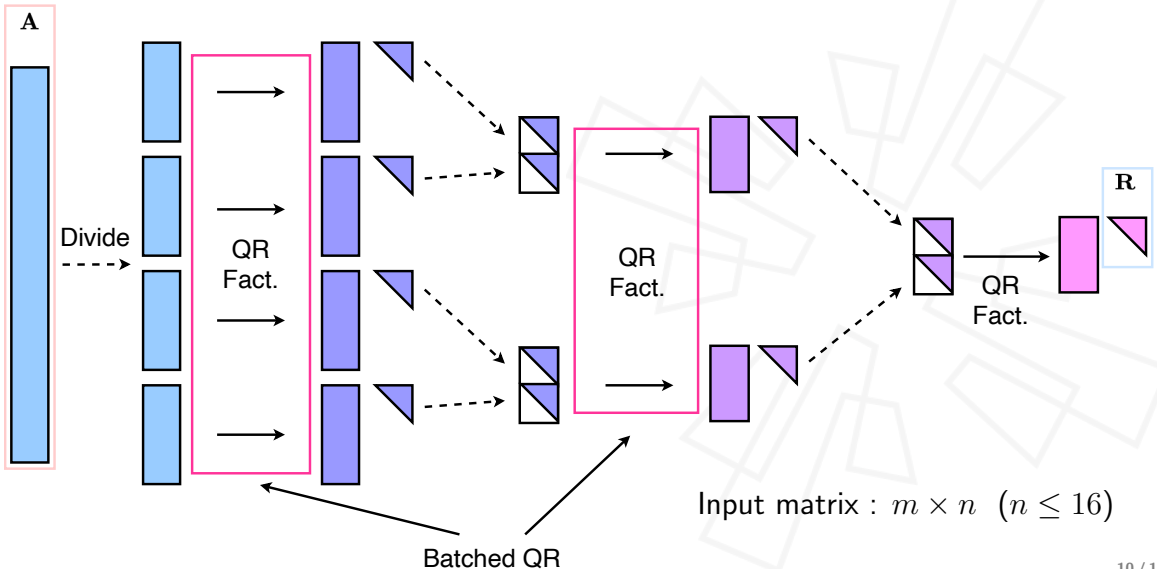
With correction

$$\mathbf{Q}_{\text{FP32}} \leftarrow \mathbf{H}_{\text{FP16}}\ \mathbf{Q}_{\text{FP16}} + \Delta\mathbf{H}_{\text{FP16}}\ \mathbf{Q}_{\text{FP16}} + \mathbf{H}_{\text{FP16}}\ \Delta\mathbf{Q}_{\text{FP16}}$$

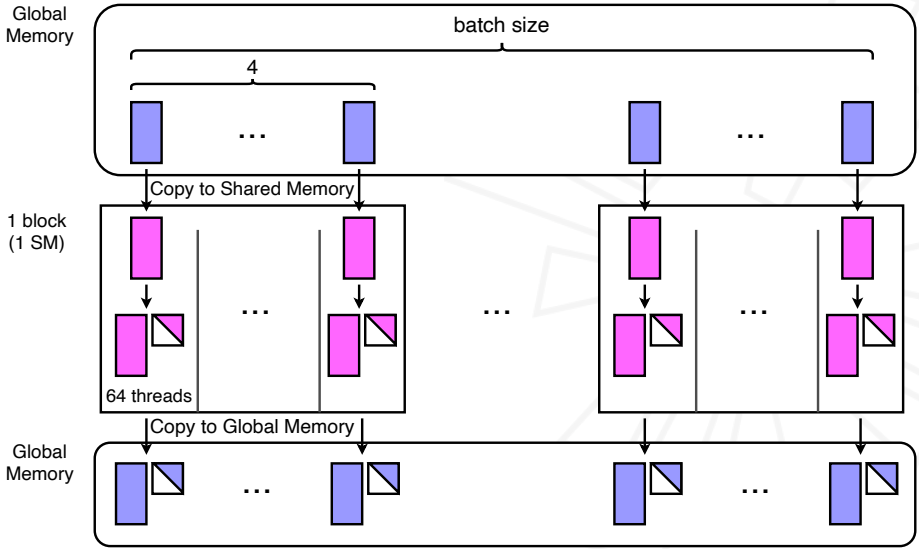
$$\mathbf{R}_{\text{FP32}} \leftarrow \mathbf{H}_{\text{FP16}}\ \mathbf{R}_{\text{FP16}} + \underbrace{\Delta\mathbf{H}_{\text{FP16}}\ \mathbf{R}_{\text{FP16}} + \mathbf{H}_{\text{FP16}}\ \Delta\mathbf{R}_{\text{FP16}}}_{\text{Correction terms}}$$

where $\Delta\mathbf{M}_{\text{FP16}} \xleftarrow{\text{narrowing}} \mathbf{M}_{\text{FP32}} - \mathbf{M}_{\text{FP16}}$

Implementation of TSQR



Implementation of batched QR

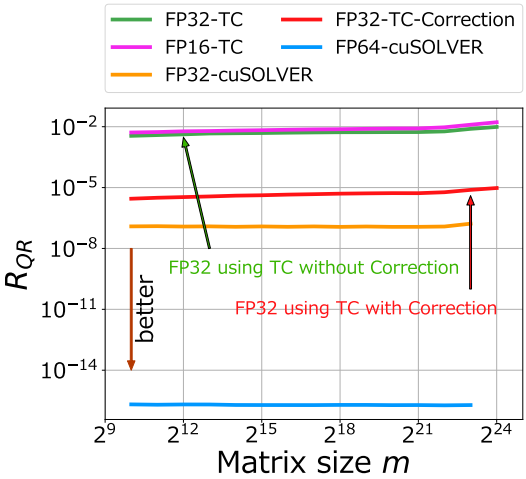


Evaluation of accuracy

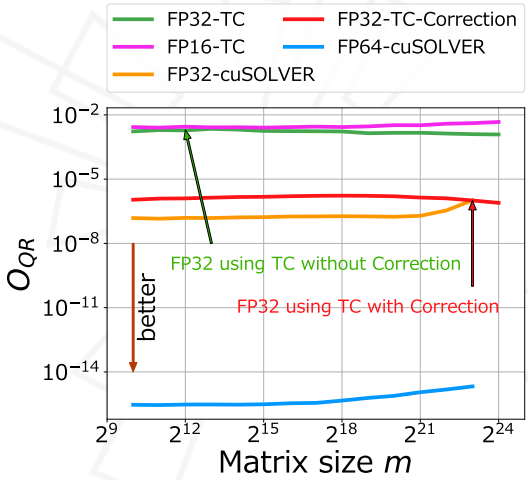
Intel Xeon CPU E5-2630 x2, NVIDIA Tesla V100 PCIE

Input : $m \times 16$

Residual



Orthogonality

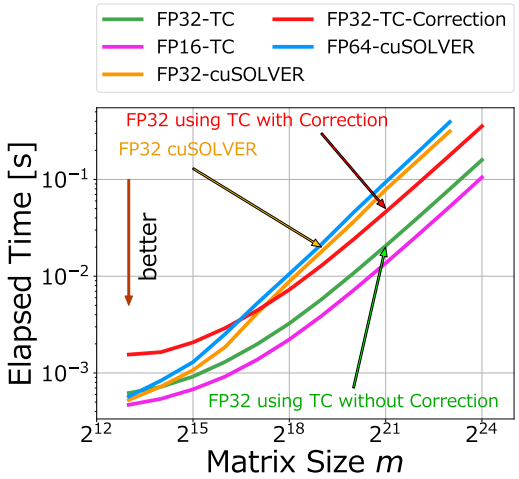


Evaluation of performance

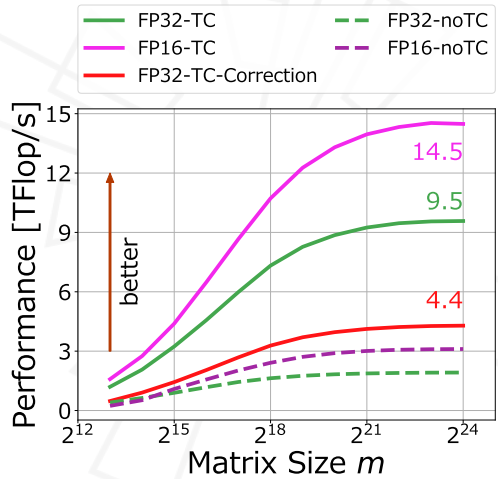
Intel Xeon CPU E5-2630 x2, NVIDIA Tesla V100 PCIE

Input : $m \times 16$

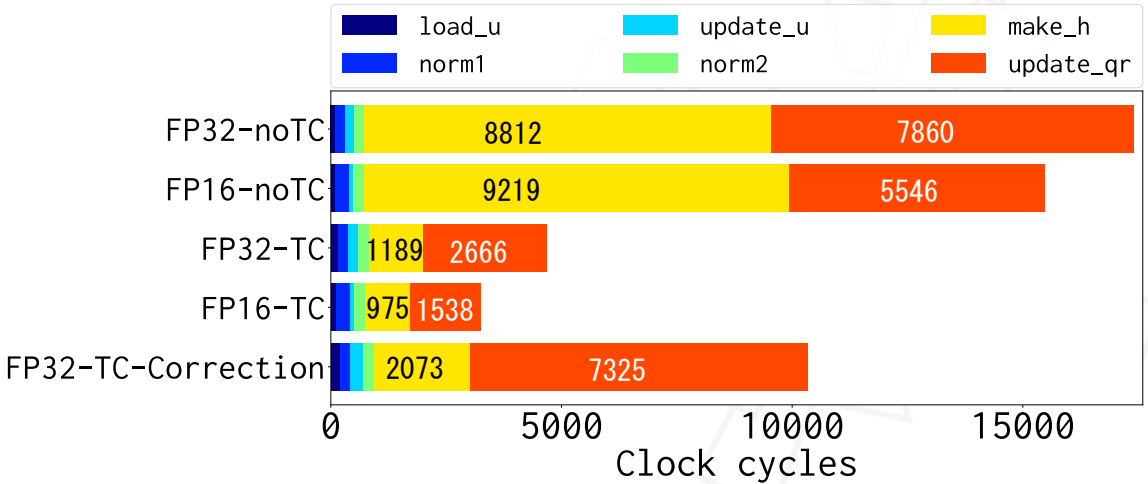
Computing time



Flop/s



Computing time distribution of different steps inside Householder QR



Conclusion and future work

Conclusion

- Using TensorCores and correction technique, we can calculate TSQR efficiently without much loss of accuracy
- Our approach provides more than 3x faster performance compared to cuSOLVER

Future work

- Implement TSQR for input matrix with larger column size
- Implement Randomized SVD

Question?

https://github.com/enp1s0/tsqr-gpu