Randomized SVD on TensorCores

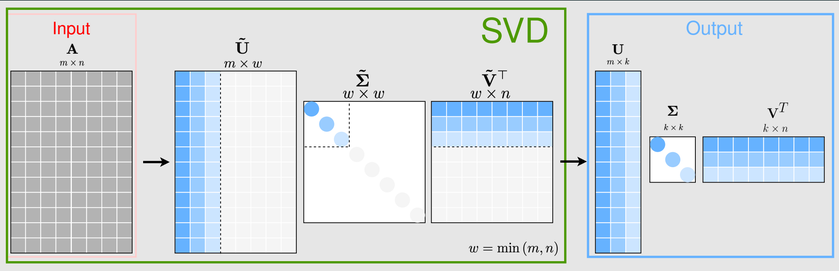
Hiroyuki Ootomo[1] , Rio Yokota[2]

[1]School of Computing, TokyoTech [2]Global Scientific Information and Computing Center, TokyoTech

Abstract

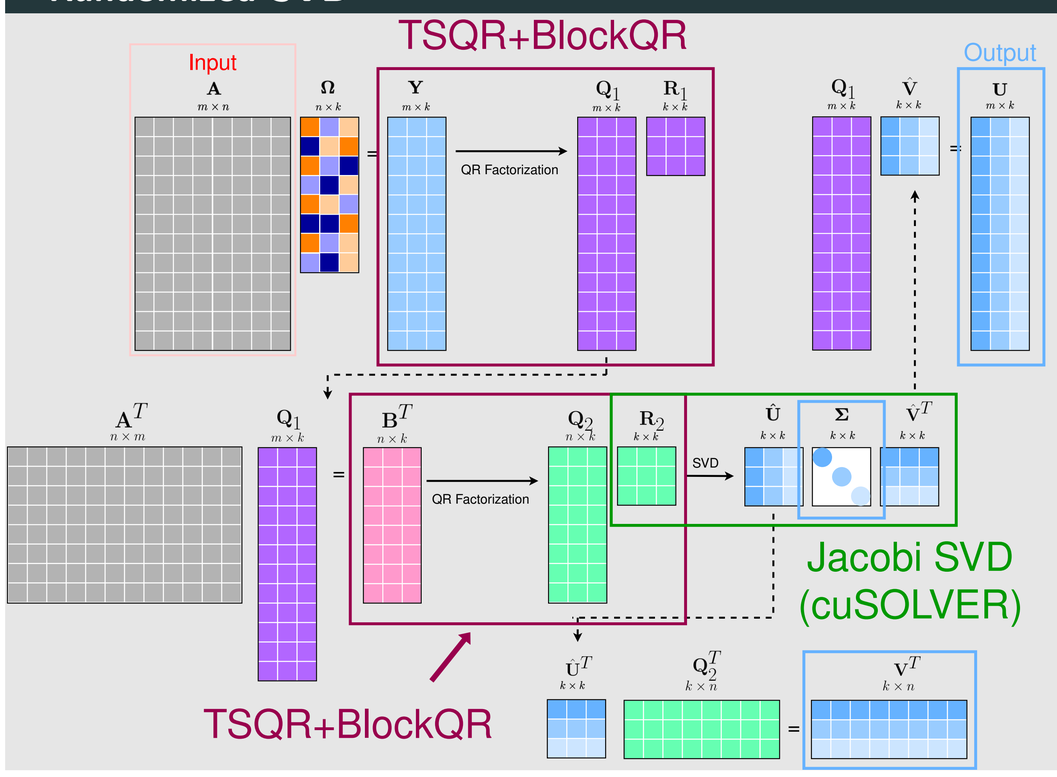
Low-rank approximation is vital and widely used for data compression, dimensiononality reduction and noise reduction. Randomized SVD is an efficient algorithm for computing low-rank approximation. This algorithm uses a QR factorization of a tall skinny matrix. We evaluate the speed, accuracy, and stability of Randomized SVD on TensorCores.

Low-rank approximation using SVD



1. $\tilde{\mathbf{U}}, \tilde{\mathbf{\Sigma}}, \tilde{\mathbf{V}}^\top \overset{SVD}{\Longleftarrow} \mathbf{A}$.
2. Reduce unnecessary singular values and singular vectors.

Randomized SVD



Randomized SVD computes low-rank approximation with specified rank k.

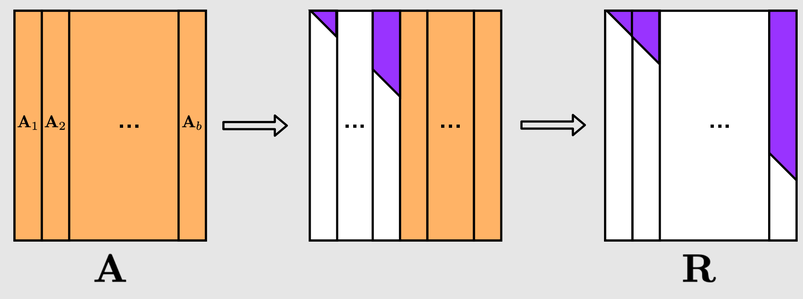
Algorithm

1. $\mathbf{Y} \Leftarrow \mathbf{A\Omega}$
2. $\mathbf{Q}_1\mathbf{R}_1 \Leftarrow \mathrm{QR}(\mathbf{Y})$
3. $\mathbf{B}^T \Leftarrow \mathbf{A}^T\mathbf{Q}_1$
4. $\mathbf{Q}_2\mathbf{R}_2 \Leftarrow \mathrm{QR}(\mathbf{B}^T)$
5. $\hat{\mathbf{U}}\mathbf{\Sigma}\hat{\mathbf{V}}^T \Leftarrow \mathrm{SVD}(\mathbf{R}_2)$
6. $\mathbf{U} \Leftarrow \mathbf{Q}_1\hat{\mathbf{V}}$
7. $\mathbf{V} \Leftarrow \mathbf{Q}_2\hat{\mathbf{U}}$

QR factorization

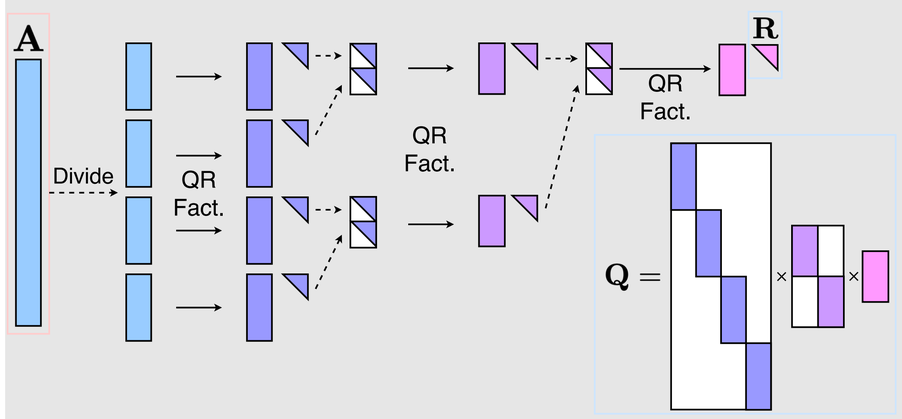
- TSQR + BlockQR
- cuSOLVER

BlockQR



1. Divide input matrix $\mathbf{A}$ into $\mathbf{A}_1, \mathbf{A}_2$ by column.
2. Compute
$$\mathbf{Q}_1, \mathbf{R}_1 \Leftarrow \mathrm{QR}(\mathbf{A}_1)$$
$$\mathbf{R}_{11} \Leftarrow \mathbf{Q}_1^\top \mathbf{A}_1$$
$$\mathbf{Q}_2, \mathbf{R}_2 \Leftarrow \mathrm{QR}(\mathbf{A}_1 - \mathbf{Q}_1\mathbf{R}_{11}).$$
3. Merge matrices.
$$\mathbf{Q} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}.$$
$$\mathbf{R} = \begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_{11} \\ \mathbf{O} & \mathbf{R}_2\end{bmatrix}.$$
4. Compute this two divided QR recursively.

TSQR on TensorCore[1]



Algo 1. TSQR

1. Divide the input matrix $\mathbf{A}$.
2. Calculate QR decomposition for each subdivided matrix to get $\mathbf{R}$s and $\mathbf{Q}$s.
3. Merge consecutive $\mathbf{R}$ blocks.
4. Repeat 2-3 until there is only one $\mathbf{R}$.
5. Calculate $\mathbf{Q}$ from $\mathbf{Q}$s which are calculated in 2-4.

Implementation

- **Step 2-4**
  Some QR decompositions can be calculated in parallel.
  $\Rightarrow$ Batched QR
- **Step 5**
  Calculate matrix multiplication implicitly using batched matmul.

Batched QR

To compute each QR factorization, we use HouseholderQR.

Where to use TensorCores

- Calculate $\mathbf{H}$ (Algo 2. line 5)
- Update $\mathbf{Q}, \mathbf{R}$ (Algo 2. line 6, 7)
- Batched Matmul

Algo 2. Householder QR

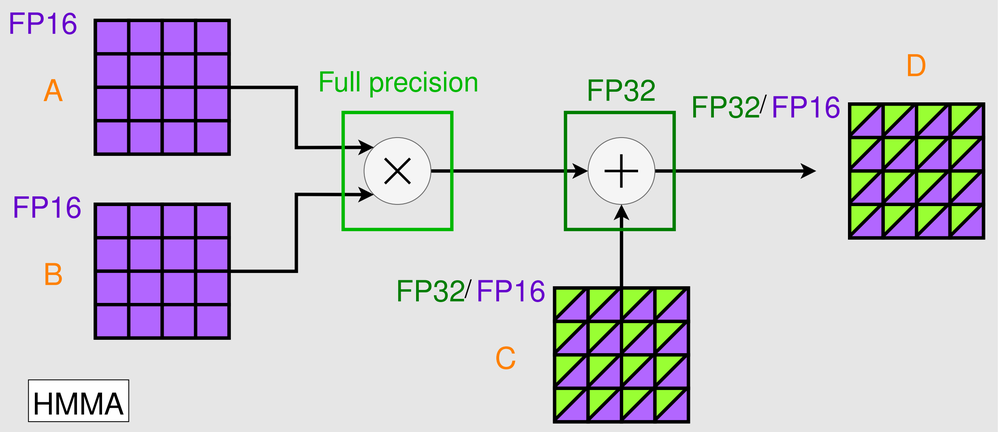
Require: $m, n \in \mathbb{N}, \mathbf{A} \in \mathbb{R}^{m\times n}$
Ensure: $\mathbf{Q} \in \mathbb{R}^{m\times m}, \mathbf{R} \in \mathbb{R}^{m\times n}$

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: $\quad \mathbf{H} \Leftarrow \mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{|\mathbf{u}|^2}$
6: $\quad \mathbf{R} \Leftarrow \mathbf{HR}$
7: $\quad \mathbf{Q}' \Leftarrow \mathbf{HQ}'$
8: **end for**
9: $\mathbf{Q} \Leftarrow \mathbf{Q}'^T$

[1] Hiroyuki Ootomo, Rio Yokota "TSQR on TensorCores", SC'19 Research Poster

FP32 matmul with accuracy correction on TensorCores

TensorCore



- Mixed precision matrix product and addition circuit

Accuracy correction of FP32 matrix product

$$\mathbf{A}_{\mathrm{FP16}} \leftarrow \mathbf{A}_{\mathrm{FP16}}, \mathbf{B}_{\mathrm{FP16}} \leftarrow \mathbf{B}_{\mathrm{FP16}}$$

- Without correction
$$\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP16}}\mathbf{B}_{\mathrm{FP16}}$$
- With correction
$$\mathbf{C}_{\mathrm{FP32}} \leftarrow \mathbf{A}_{\mathrm{FP16}}\mathbf{B}_{\mathrm{FP16}} + \underbrace{\Delta\mathbf{A}_{\mathrm{FP16}}\mathbf{B}_{\mathrm{FP16}} + \mathbf{A}_{\mathrm{FP16}}\Delta\mathbf{B}_{\mathrm{FP16}}}_{\text{Correction term}}$$
where $\Delta\mathbf{M}_{\mathrm{FP16}} \xleftarrow{\text{Convert to FP16}} \mathbf{M}_{\mathrm{FP32}} - \mathbf{M}_{\mathrm{FP16}}$

Test environment

- Intel Xeon CPU E5-2630 v3 @ 2.40GHz x2
- NVIDIA Tesla V100-PCIE-16GB
- CUDA 10.1

Implemantation

- Input/Output type is FP32
- SVD oeprations using `gesvdj` subroutine of cuSOLVER
- Implement TSQR with different configuratios to be used within the QR factorizations (BlockQR) of Randomized SVD

	TensorCore	Correction
TC-noCor	YES	NO
TC-Cor	YES	YES
noTC	NO	NO

- Implement Randomized SVD using QR factorizations subroutine of cuSOLVER for comparison

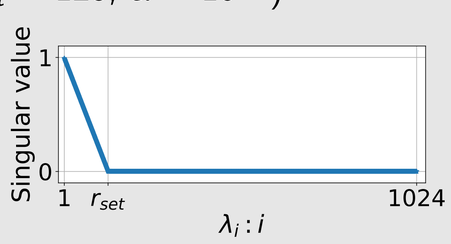
Evaluation of Randomized SVD on TensorCores

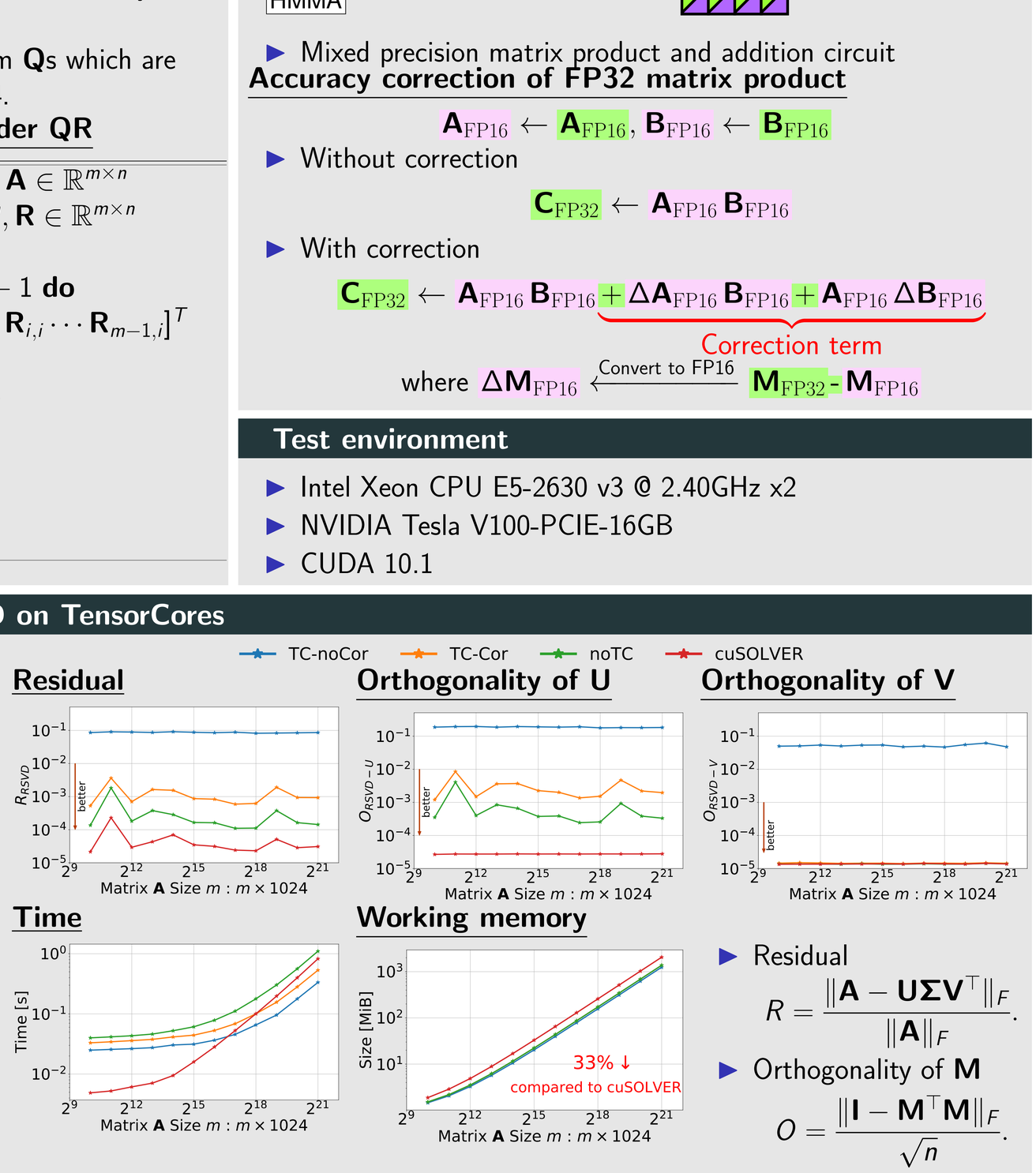
Input matrix

Random $m \times 1024$ FP32 matrix $\mathbf{A}$ whose singular values are calculated by eq (1).

$$\lambda_i = (1-\alpha)\max\left(1 - \frac{i-1}{r_{set}}, 0\right) + \alpha \quad (1)$$

$(r_{set} = 128, \alpha = 10^{-2})$





- Residual
$$R = \frac{\|\mathbf{A} - \mathbf{U\Sigma V}^\top\|_F}{\|\mathbf{A}\|_F}.$$
- Orthogonality of $\mathbf{M}$
$$O = \frac{\|\mathbf{I} - \mathbf{M}^\top\mathbf{M}\|_F}{\sqrt{n}}.$$

Conclusion

- Using TensorCores and correction techniques, our approach can calculate Randomized SVD without much loss of accuracy.
- Our approach provides 1.5x faster performance and reduces working memory by 33% compared to cuSOLVER.

Future work

- Investigate the effect of using FP16 to compute the Randomized SVD in depth.
- Experiment with different singular value decay patterns.
- Apply the Randomized SVD using TensorCores to some applications and investigate stability and performance.