MIXED - PRECISION BLOCK QR DECOMPOSITION ON GPU

STUDENTS: Jaidon Lybbert, Fulin Li , Mike Pao, Alice Lin, Shashank Shivashankar

amazon Lab126

Motivation

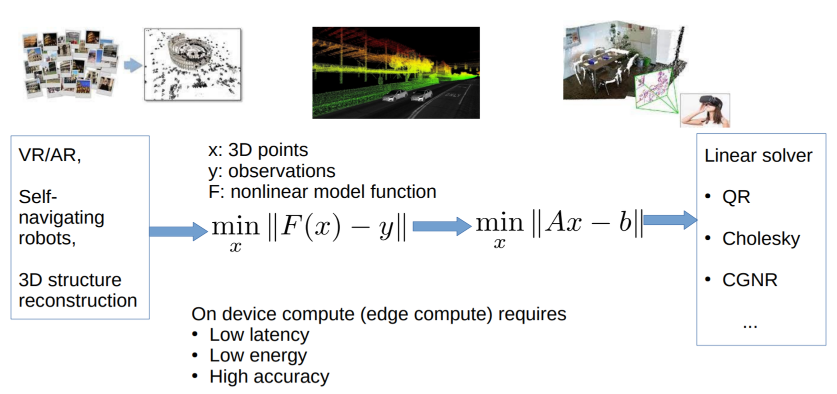
General QR Decomposition

$$A = QR$$

Our project aims to optimize QR decomposition, which factorizes a matrix into an orthogonal matrix (Q) and an upper triangular matrix (R), by developing efficient algorithms and techniques to improve its computational efficiency and precision.

Application

QR decomposition is crucial for various applications in linear algebra and numerical computation.

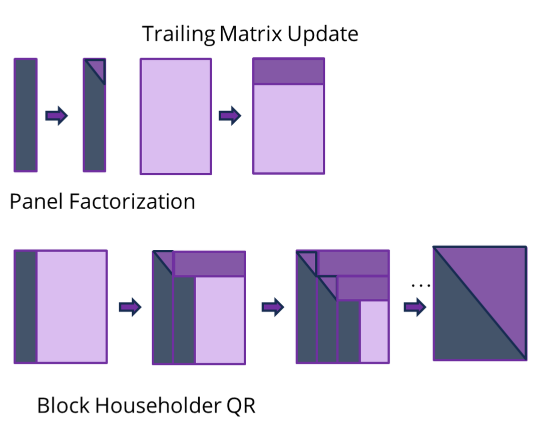


Software Design

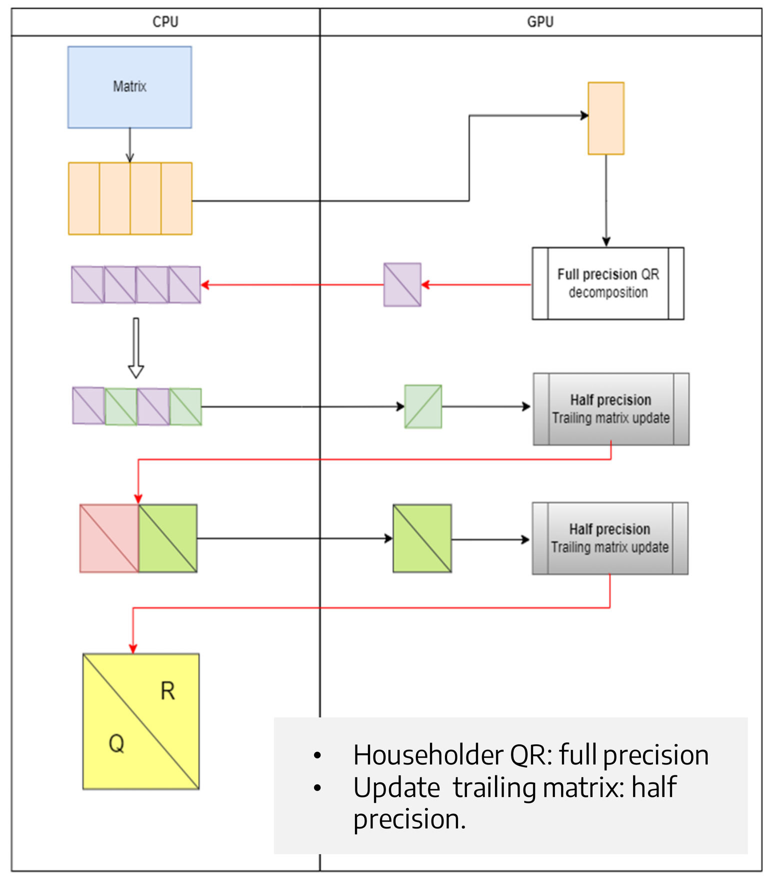
To address this challenge, our team implements several techniques to accelerate the QR decomposition process.

Block QR

Split the input matrix into smaller block matrices and perform QR decomposition on each block.

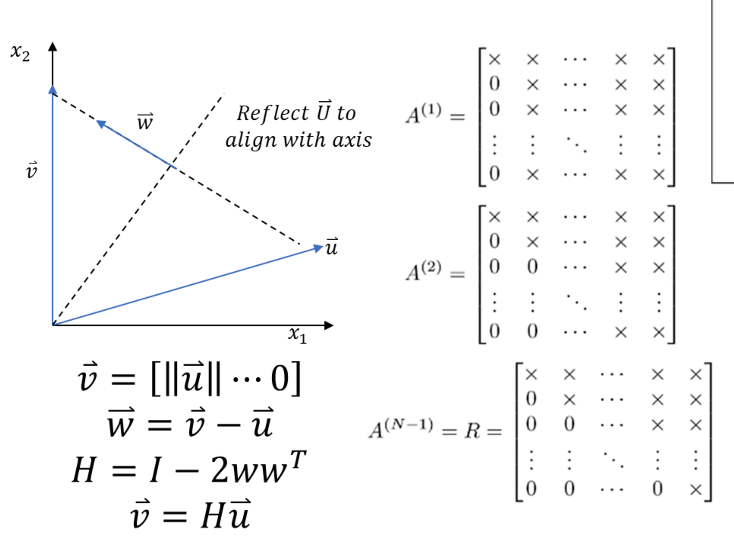


Mixed Precision



- Householder QR: full precision
- Update trailing matrix: half precision.

Householder QR



- Iterate over columns
  - Compute householder matrix to zero column below diagonal
  - Update matrix A by A = HA
- After iterating over all columns A = R

WY Transformation

Combine multiple householder transformations into a single matrix via the WY-representation of matrix products before doing the matrix update.

$Q \in R^{M\times M} = Q_1 Q_2 \cdots Q_j \cdots Q_n$
where $Q_j = I_m - \beta_j w_j w_j^T$
and the factors β_j, w_j *are stored as*

$V \in R^{M\times n} = [w_1 w_2 \cdots w_j \cdots w_n]$
$B \in R^{n} = [\beta_1 \beta_2 \cdots \beta_j \cdots \beta_n]$

the W and Y factors such that $Q = I_m - WY^T$ *can be caculated from V, and B.*

Requirements

Targets

- ✓ Implement a fast and correct version of mixed precision QR using GPU.
- ✓ Implementation should use the GPU's half precision data type for the matrix multiplications and single precision for remaining operations.
- ✓ The computing accuracy and speed to be higher than that of a naive implementation on an x86 CPU architecture.
- ✓ Integration into an Open-Source Package.

Hardware Constraints

NVIDIA RTX 2080	
# of SMs	46
Threads/SM	1024
Blocks/SM	16
TC FMA dimension	16x16x16
TC / SM	8
SMEM	64KB
RF	4 * 64kB

Software Dependencies

- **CUDA Toolkit:**
  - Version: CUDA Toolkit 9.1 or higher
  - Provides development tools and libraries for GPU programming
- **NVIDIA GPU Driver:**
  - Required for NVIDIA RTX 2080 hardware compatibility

CUDA programming

Procedures

Minimize communication across PCIE bus

Child kernels (CUDA dynamic parallelism)

Host: Host Code — Use RAM

Device: Block QR Kernel — Use VRAM; Householder QR Kernel — Use SMEM; Matrix Update Kernel

1. Divide the matrix to N blocks. N is a power of 2.
2. Send every block to the GPU core, apply full precision **Householder QR** in every block. — Parallel computing in multiple GPU cores.
3. Combine the panel Q matrices from each block into a single matrix Q using the **WY transformation**.
4. Get all QR deposition of every block, then recursively **update the trailing matrix.** (Requires large amounts of computation) — To enhance efficiency, half precision is used to accelerate the process.

Algorithm

Algorithm 1 Calculate $A = QR$ using Householder reflections

```
for k = 1 to n do
    u = A_{k:m,k}
    v_k = sign(u_1)||u||_2 e_1 + u
    v_k = v_k / ||v_k||_2
    A_{k:m,k:n} = A_{k:m,k:n} − 2 v_k (v_k^T A_{k:m,k:n})
end for
```

Algorithm 2 Calculate W, Y from the factored form of Q: V and B

```
Y = w_1
W = β_1 w_1
for j = 2 to r do
    z = β_j (I_m − W Y^T) w_j
    W = [W|z]
    Y = [Y|w_j]
end for
```

Algorithm 3 Block Householder QR Decomposition

```
Q = I_m
λ = 1
k = 0
while λ ≤ n do
    τ ← min(λ + r − 1, n)
    k = k + 1
    A_{λ:m,λ:τ} ← Householder_qr(A_{λ:m,λ:τ})
    W_k, Y_k ← WY_transform(V_k)
    A_{λ:m,τ+1:n} = (I − W_k Y_k^T)^T A_{λ:m,τ+1:n}
    Q_{:,λ:m} = Q_{:,λ:m}(I − W_k Y_k^T)
    λ = τ + 1
end while
```

Performance results

Error criteria

$$\|QR - A\| \leq m \cdot 2^{-23}\|A\|$$
$$\|Q^H Q - I\| \leq m \cdot 2^{-23}$$
$$\|L\| \leq m \cdot 2^{-23}$$

* L: the trapezoidal submatrix below the main diagonal of R

https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets

Input Matrix A: Random Float Matrix
Matrix Sizes: Various Sizes

By utilizing GPU parallel computing, we can perform several tasks at once, leading to significant speedup compared to traditional sequential CPU processing.

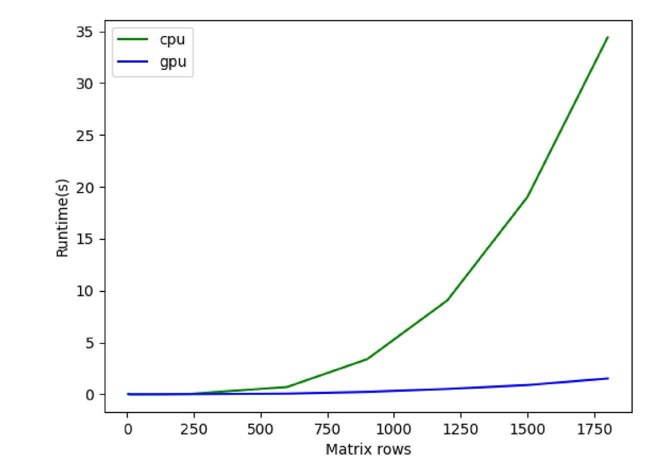


Figure 1:
Runtimes of QR decomposition

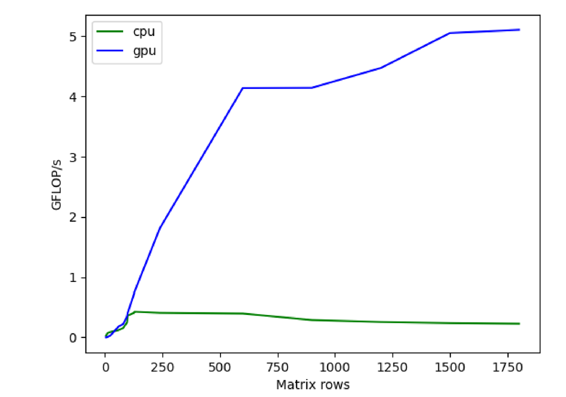


Figure 2:
Sustained performance of QR decomposition

Conclusion

- By implementing the block QR decomposition in parallel on a GPU, we saw a speedup of **over 10x** compared to the sequential CPU implementation for large matrices, meeting our success criteria for execution time
- Our performance bottleneck is in the construction of matrix Q, which takes **about 80%** of the execution time, this *can be accelerated* on the GPU

Future Work, References, and Acknowledgments

- Implement tiled QR to enhance the parallelism and performance of the QR decomposition process.
- Investigate and incorporate any other relevant advancements or optimizations in the field to enhance the overall algorithm.

References

- Zhang, S., Baharlouei, E., & Wu, P. (2020). High Accuracy Matrix Computations on Neural Engines: A Study of QR Factorization and its Applications. Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 17–28. https://doi.org/10.1145/3369583.3392685 | PDF
- Bouwmeester, H., Mathias Jacquelin, Langou, J., & Yves, R. (2011). Tiled QR factorization algorithms. arXiv.org. https://arxiv.org/pdf/1104.4475.pdf

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

ADVISORS: TONG QIN, NATHAN KUTZ

SPONSORS: AMAZON LAB 126