Parallel QR Factorization

Wangshu Li (wangshul)

Jiarui Zhang (jiaruiz)

Description

QR factorization is a decomposition of a non-singular matrix A into a product A = QR of an orthogonal matrix Q and an upper triangular matrix R

$$\overset{\mathbf{A}}{\begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \\ | & | & | \end{bmatrix}} = \underbrace{\overset{\mathbf{Q}}{\begin{bmatrix} | & | & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ | & | & | \end{bmatrix}}}_{\text{orthogonal unit vector}} \underbrace{\overset{\mathbf{R}}{\begin{bmatrix} \mathbf{e}_1^T \cdot \mathbf{a}_1 & \mathbf{e}_1^T \cdot \mathbf{a}_2 & \mathbf{e}_1^T \cdot \mathbf{a}_3 \\ 0 & \mathbf{e}_2^T \cdot \mathbf{a}_2 & \mathbf{e}_2^T \cdot \mathbf{a}_3 \\ 0 & 0 & \mathbf{e}_3^T \cdot \mathbf{a}_3 \end{bmatrix}}}_{\text{Upper Diagonal matrix}}$$

Sequential Algorithm

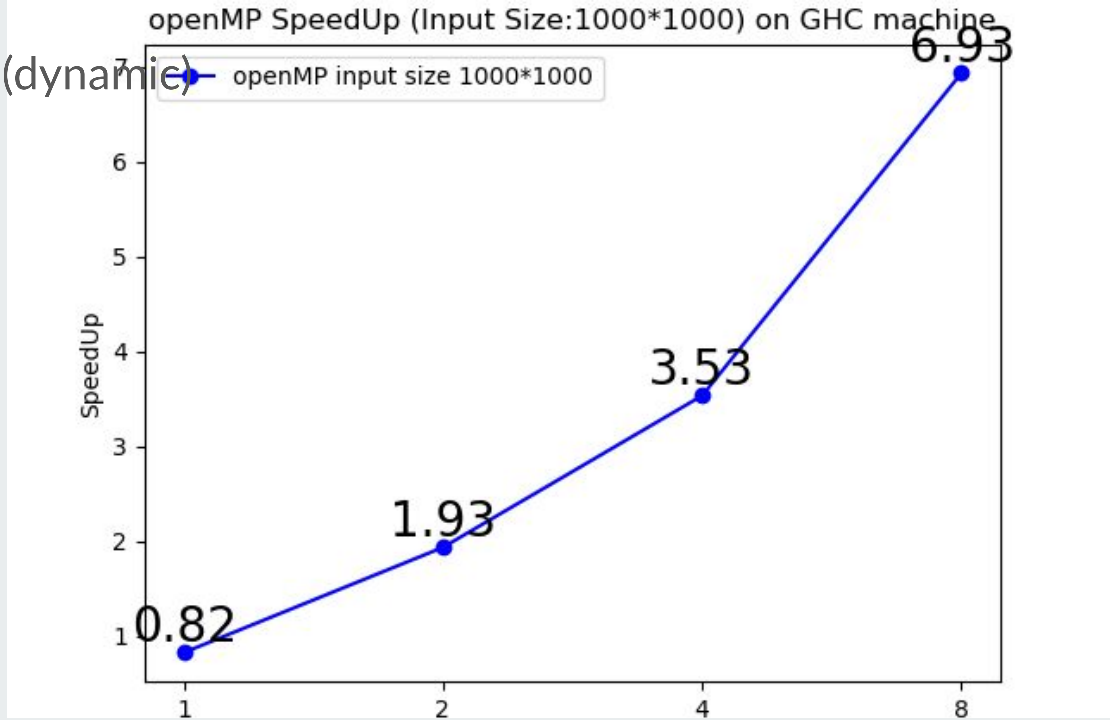
Algorithm 5: QR Decomposition.

input : A non-singular matrix $\mathbf{A}$ of size $n \times n$
output: Orthogonal matrix $\mathbf{Q}$, upper triangular form $\mathbf{U}$

```
Function QRFact(A) is
    N ← order of the square matrix A;
    u ← 0_{N×N}, R ← 0_{N×N};

    for j ← 0 to n − 1 do
        w_j ← A[:, j];

        /* Squared norm of w_j                                  */
        sum ← 0;
        for k ← 0 to n − 1 do
            sum ← sum + w_j[k] × w_j[k];
        |w_j|^2 ← sum;

        /* Compute the coefficients R[i][j]                     */
        sum ← 0;
        for i ← 0 to j − 1 do
            R[i][j] ← w_j · u[i];
            sum ← sum + R[i][j] × R[i][j];
        R[j][j] ← (|w_j|^2 − sum)^{1/2};

        /* Compute vector u[j] using fwd substitution           */
        sumvec ← 0_{1×N};
        for i ← 0 to j − 1 do
            sumvec ← sumvec + R[i][j] × u[i];
        u[j] ← (w_j − sumvec)/R[j][j];

    Q ← u^T;
    return Q, R;
```

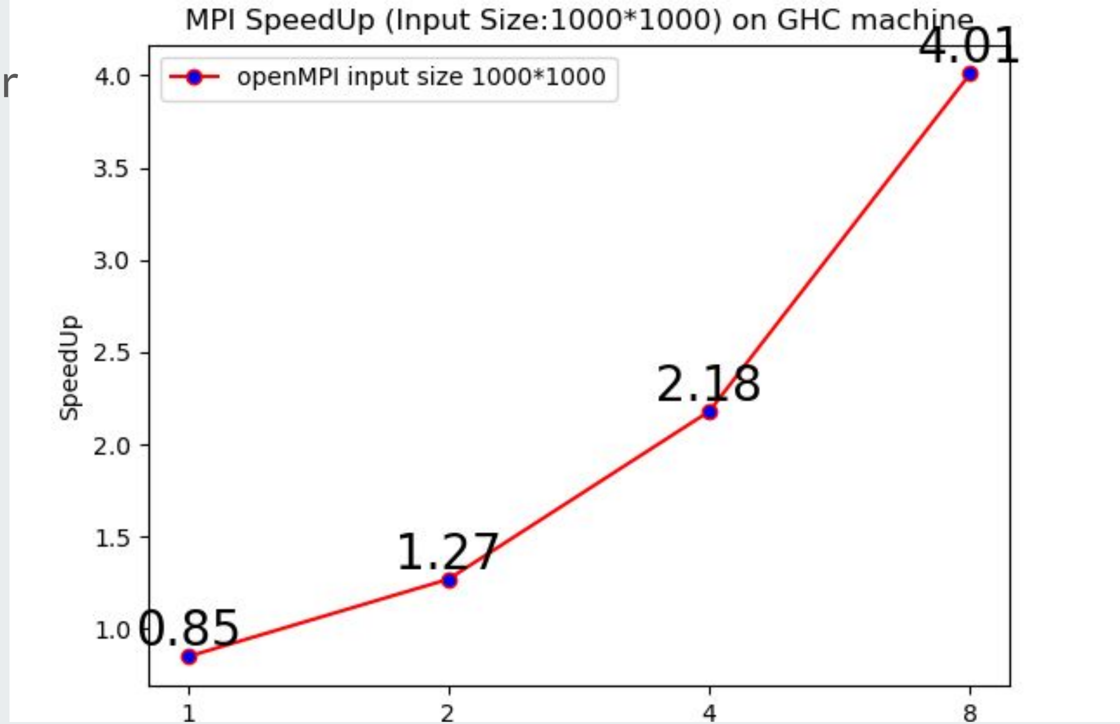
OpenMP

- #pragma omp parallel for schedule (dynamic)
  - Parallelize the outer loop
- #pragma omp barrier
  - Synchronize between threads
- Almost linear speedup



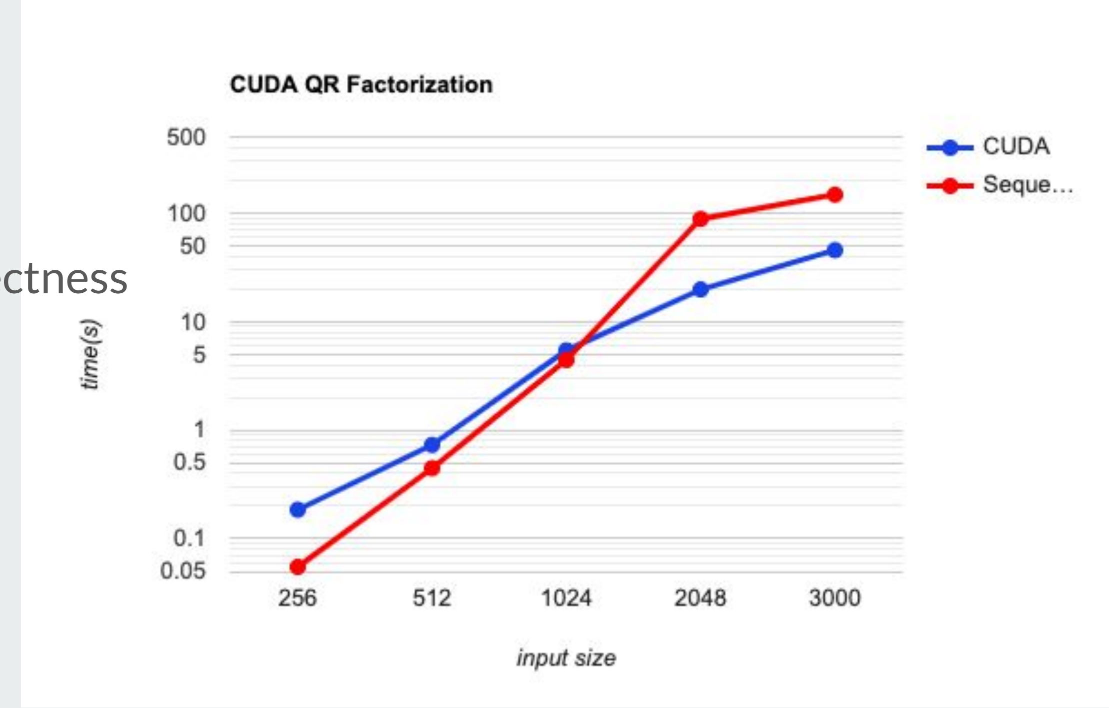
OpenMPI

- Each process works for 1/N number of total columns
  - Parallelism over columns
- Processes send its columns to all processes with higher pid
  - High communication cost
- Use a 2-d buffer for receiving
  - Workflow pipeline



CUDA

- Each thread works for a single cell
    - Parallelism over cells
- Each column launches the kernel
    - Guarantee dependency/correctness
- Shared block memory
    - Reduce I/O latency
- Synchronization within kernel
- Bad when input size is small
    - Data dependency



Analytics

- Data dependency due to the nature of QR factorization
  - Each column needs previous columns for computation
- OpenMP has the best performance
  - Shared address space model has minimal communication cost
  - No synchronization is required between threads because there is no data race
- CUDA and MPI have worse performance
  - Too many kernel functions when parallelizing over cells
  - CUDA is not appropriate for dependent workloads
  - OpenMPI has high communication cost