EPFL

HPC for numerical methods and data analysis

Fall Semester 2024

Prof. Laura Grigori

Assistant: Mariana Martínez Aguilar

Session 5 – October 8, 2024

TSQR Factorization

Exercise 1 CGS and MGS

If we recall what a QR factorization is, given a matrix $A \in \mathbb{R}^{m\times n}$, with $m \geq n$, its QR factorization is

$$A = \hat{Q}\hat{R} = \begin{bmatrix} Q & \tilde{Q} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = QR,$$

where $\hat{Q} \in \mathbb{R}^{m\times m}$ orthogonal and $\hat{R} \in \mathbb{R}^{m\times n}$ upper triangular. Note that A can be seen as a map $W : \mathbb{R}^n \to \mathbb{R}^m$. Recall that computing the QR decomposition using CGS is numerically unstable. An alternative algorithm, which is mathematically equivalent is the Modified Gram-Schmidt algorithm (MGS). Define Q_{j-1} as the matrix we get at the j-th step, $Q_{j-1} = \begin{bmatrix} q_1 & q_2 & \dots & q_{j-1} \end{bmatrix}$ and P_j as the projector onto the subspace orthogonal to the column space $\text{col}(Q_{j-1})$. Then we can write this as:

$$P_j = I - Q_{j-1}Q_{j-1}^* = (I - q_{j-1}q_{j-1}^*)\dots(I - q_1q_1^*).$$

a) How many synchronizations do we need for each vector w_j in this case? Why?

b) Why is this more stable than CSG?

c) Make the necessary modifications to your last week's code to implement MGS. Compare both codes by computing $\|I - QQ^\top\|$.

Exercise 2 TSQR

Remember that communication refers to messages between processors. In the recent years we've seen trends causing floating point to become faster than communication. This is why it's important to minimize communication when dealing with parallelism. The TSQR, "Tall Skinny QR" algorithm is a communication avoiding algorithm for matrices with many more rows than columns. In this exercise we are going to assume we're using $P = 4$ processors and the matrix we want to factorize is $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. The computation can be expressed as a product of intermediate

orthonormal factors in a binary tree like structure. We scatter row wise our matrix A along 4 processors $A_1, A_2, A_3, A_4 \in \mathbb{R}^{m/4 \times n}$. At the leaves of the binary tree, we perform in parallel 4 local QR factorizations to get $A_1 = \hat{Q}_1^{(2)}\hat{R}_1^{(2)}, A_2 = \hat{Q}_2^{(2)}\hat{R}_2^{(2)}, A_3 = \hat{Q}_3^{(2)}\hat{R}_3^{(2)}, A_4 = \hat{Q}_4^{(2)}\hat{R}_4^{(2)}$. Here $\hat{Q}_l^{(2)} \in \mathbb{R}^{m/4 \times m/4}$ and $\hat{R}_l^{(2)} \in \mathbb{R}^{m/4 \times n}$. In block structure we get

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} = \begin{bmatrix} \hat{Q}_1^{(2)} & & & \\ & \hat{Q}_2^{(2)} & & \\ & & \hat{Q}_3^{(2)} & \\ & & & \hat{Q}_4^{(2)} \end{bmatrix} \begin{bmatrix} \hat{R}_1^{(2)} \\ \hat{R}_2^{(2)} \\ \hat{R}_3^{(2)} \\ \hat{R}_4^{(2)} \end{bmatrix}$$

Recall that $\hat{R}_l^{(2)} \in \mathbb{R}^{m/4 \times n}$ are tall and skinny upper triangular matrices, hence they can be written as

$$\hat{R}_l^{(2)} = \begin{bmatrix} R_l^{(2)} \\ 0 \end{bmatrix},$$

where $R_l^{(2)} \in \mathbb{R}^{n \times n}$. In the second level of the binary tree we combine the upper triangular matrices $R_1^{(2)}$ with $R_2^{(2)}$ and $R_3^{(2)}$ with $R_4^{(2)}$ to get block structured matrices. We perform the QR factorization in parallel to get the following structured matrices

$$\begin{bmatrix} R_1^{(2)} \\ R_{(2)}^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{Q}_{11}^{(1)} & \hat{Q}_{12}^{(1)} \\ \hat{Q}_{21}^{(1)} & \hat{Q}_{22}^{(2)} \end{bmatrix} \begin{bmatrix} R_1^{(1)} \\ 0 \end{bmatrix} \qquad \begin{bmatrix} R_3^{(2)} \\ R_{(4)}^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{Q}_{33}^{(1)} & \hat{Q}_{34}^{(1)} \\ \hat{Q}_{43}^{(1)} & \hat{Q}_{44}^{(2)} \end{bmatrix} \begin{bmatrix} R_3^{(1)} \\ 0 \end{bmatrix}.$$

This can be written as

$$\begin{bmatrix} \hat{R}_1^{(2)} \\ \hat{R}_2^{(2)} \\ \hat{R}_3^{(2)} \\ \hat{R}_4^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{Q}_{11}^{(1)} & & \hat{Q}_{12}^{(1)} & & & & & \\ & I & & & & & & \\ \hat{Q}_{21}^{(1)} & & \hat{Q}_{22}^{(1)} & & & & & \\ & & & I & & & & \\ & & & & \hat{Q}_{33}^{(1)} & & \hat{Q}_{34}^{(1)} & \\ & & & & & I & & \\ & & & & \hat{Q}_{43}^{(1)} & & \hat{Q}_{44}^{(1)} & \\ & & & & & & & I \end{bmatrix} \begin{bmatrix} R_1^{(1)} \\ 0 \\ R_3^{(1)} \\ 0 \end{bmatrix}.$$

Finally at the root of the tree we compute the last QR factorization

$$\begin{bmatrix} R_1^{(1)} \\ R_3^{(1)} \end{bmatrix} = \begin{bmatrix} \hat{Q}_{11}^{(0)} & \hat{Q}_{13}^{(0)} \\ \hat{Q}_{31}^{(0)} & \hat{Q}_{33}^{(0)} \end{bmatrix} \begin{bmatrix} R_1^0 \\ 0 \end{bmatrix}$$

Which can be written as

$$\begin{bmatrix} R_1^{(1)} \\ 0 \\ R_{(3)}^{(1)} \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{Q}_{11}^{(0)} & & \hat{Q}_{13}^{(0)} & \\ & I & & \\ \hat{Q}_{31}^{(0)} & & \hat{Q}_{33}^{(0)} & \\ & & & I \end{bmatrix} \begin{bmatrix} R_1^{(0)} \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

$\hat{Q}$ is represented implicitly as the product of the intermediate orthogonal matrices.

a) Let $Q \in \mathbb{C}^{a \times b}$ and $S \in \mathbb{C}^{b \times c}$ be matrices such that their columns form an orthonormal set. Show that the columns of QS also form an orthonormal set.

b) Let $A \in \mathbb{R}^{m \times n}$ with $m > n$. Write down the dimensions of all the intermediate matrices and their sub blocks.

c) Show that the columns of the resulting $\hat{Q}$ are orthogonal. How is this matrix computed?

d) Suppose that you are given the implicit representation of Q as a product of orthogonal matrices. This representation is a tree of sets of Householder vectors, $\{\hat{Q}^d_{r,k}\}$. Here, k indicates the processor number (both where it is computed and where it is stored), and d indicates the level in the tree. How would you get $\hat{Q}$ explicitly? We only need the "thin" $\hat{Q}$, meaning only its first n columns. Note that we can do this by applying the intermediate factors $\{\hat{Q}^d_{r,k}\}$ to the first n columns of the $m \times m$ identity matrix. Write a Python script using MPI that does this (assume you are using 4 processors). (*Hint: looking at the graphical representation of parallel TSQR might help.*)

e) *Optional:* consider more processors. How would you change your code? Do you have a restriction on the number of processors you can use?