Parallelization of QR matrix factorization using Modified Gram-Schmidt & Householder Algorithms

Abhijeet Bodas
Roll No. 190100004
IIT Bombay
190100004@iitb.ac.in

Hardik Shrivastava
Roll No. 190100053
IIT Bombay
190100053@iitb.ac.in

Shiven Barbare
Roll No. 190100110
IIT Bombay
190100110@iitb.ac.in

***Abstract*—This report focusses on the parallel implementation of QR matrix decomposition, using two popularly known algorithms, Modified Gram-Schmidt and Householder transformation. These algorithms will be implemented using CPU and GPU parallelization with the help of OpenMP and CUDA respectively. In the end, a time study will be performed with varying number of threads and the size of the matrix.**

I. Introduction

Solving linear systems is a task that has many applications in real-world problems such as signal processing, continuum stress state analysis of a body, cryptography, share market analysis, etc. For higher dimensional systems, the task of solving such a large system becomes quite cumbersome.

There are mainly two ways of solving systems of linear equations: Direct methods (like Gauss Elimination or Matrix factorization) and iterative methods (like Gauss–Seidel or Jacobi method). Matrix factorization techniques include LU factorization, Cholesky factorization and QR factorization. In this report, we will be using QR matrix decomposition technique to factorize a given square matrix into an orthogonal matrix Q and an upper triangular matrix R, which provides ease in solving the system of linear equations. We aim to implement a parallel code for the QR matrix factorization using the Householder and Modified Gram-Schmidt algorithms. We will be doing a time study with variation in the number of threads, and the dimension of the matrix. The CPU and GPU parallelization will be done using the OpenMP API and CUDA respectively. Throughout the analysis, we will be dealing only with real and square matrices.

II. QR Matrix Factorization

This matrix decomposition technique factorizes a matrix A into a product of two matrices given by

$$A = QR \tag{1}$$

where Q is an **orthogonal** matrix (i.e. $QQ^T = I$ where Q^T is the matrix transpose of Q and I is the identity matrix) and R is an **upper-triangular** matrix. Any square matrix can be expressed as a product QR as mentioned before, however special cases arise:

1) If A is invertible, then R is also invertible
2) If R has positive diagonal entries, then the pair (Q, R) is unique

III. Algorithms for QR factorization

There are several practical algorithms used to evalute the QR factorization of a matrix. Two of the popular ones are:

- Gram-Schmidt algorithm
- Householder algorithm

The implementation of both these algorithms involves a common first step. As A is a square matrix, (1) can be re-written as

$$Q^T A = R \tag{2}$$

In other words,the matrix A can be converted into an upper-triangular matrix R. The goal is to achieve this conversion by a set of matrix operations performed on matrix A, which are captured in the matrix Q^T i.e.

$$Q_n Q_{n-1} .... Q_1 A = R \tag{3}$$

where $Q^T = Q_n Q_{n-1} .... Q_1$ and $Q_1, Q_2, ..., Q_n$ represent the matrix transformations.

A. The Gram-Schmidt Algorithm

The Gram-Shchmidt algorithm is based on the Gram-Schmidt orthonormalization process that is used to convert any finite set of linearly independent vectors into a corresponding orthogonal basis in an inner product space. Consider the Euclidean space $\mathbb{R}^n$ $(n \in \mathbb{N})$ with the standard inner product. The projection of a vector $\boldsymbol{x}$ along another vector $\boldsymbol{y}$ is given by

$$\boldsymbol{x}_{||\boldsymbol{y}} = \frac{< \boldsymbol{x}, \boldsymbol{y} >}{||\boldsymbol{y}||^2} \boldsymbol{y} \tag{4}$$

where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $< \cdot, \cdot >$ is the standard inner product and $|| \cdot ||$ is the l_2 norm on $\mathbb{R}^n$.

Let $\boldsymbol{A_1}, \boldsymbol{A_2}, ..., \boldsymbol{A_n}$ represent the column vectors of the $(n \times n)$ matrix A in (1) i.e. $A = (\boldsymbol{A_1}|\boldsymbol{A_2}|...|\boldsymbol{A_n})$. Using Gram-Schmidt orthonormalization, we get the following set of n orthonormal basis vectors

$$\boldsymbol{Q_1} = \frac{\boldsymbol{A_1}}{||\boldsymbol{A_1}||}$$
$$\boldsymbol{Q_2} = \frac{\boldsymbol{A_2} - < \boldsymbol{A_2}, \boldsymbol{Q_1} > \boldsymbol{Q_1}}{||\boldsymbol{A_2} - < \boldsymbol{A_2}, \boldsymbol{Q_1} > \boldsymbol{Q_1}||}$$
$$\vdots$$
$$\boldsymbol{Q_k} = \frac{\boldsymbol{A_k} - < \boldsymbol{A_k}, \boldsymbol{Q_1} > \boldsymbol{Q_1} - ... - < \boldsymbol{A_k}, \boldsymbol{Q_{k-1}} > \boldsymbol{Q_{k-1}}}{||\boldsymbol{A_k} - < \boldsymbol{A_k}, \boldsymbol{Q_1} > \boldsymbol{Q_1} - ... - < \boldsymbol{A_k}, \boldsymbol{Q_{k-1}} > \boldsymbol{Q_{k-1}}||}$$

where $k = 2, 3, ..., n$ and $\boldsymbol{Q_k}$ are the column vectors of the matrix Q i.e.

$$Q = (\boldsymbol{Q_1}|\boldsymbol{Q_2}|...|\boldsymbol{Q_n}) \tag{5}$$

The matrix R is given by

$$R = \begin{pmatrix} < \boldsymbol{A_1}, \boldsymbol{Q_1} > & < \boldsymbol{A_2}, \boldsymbol{Q_1} > & ... & < \boldsymbol{A_n}, \boldsymbol{Q_1} > \\ 0 & < \boldsymbol{A_2}, \boldsymbol{Q_2} > & ... & < \boldsymbol{A_n}, \boldsymbol{Q_2} > \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & ... & < \boldsymbol{A_n}, \boldsymbol{Q_n} > \end{pmatrix} \tag{6}$$

B. The modified Gram Schmidt Algorithm

The Gram-Schmidt algorithm, although easy to implement in code (particularly if an in-built linear algebra library is available) is not numerically stable. It produces errors in finite precision arithmetic i.e. the matrix Q is not strictly orthogonal.

To handle this, the *Modified Gram Schmidt* algorithm is commonly used in computations, and we will do so in this study too. The method is mathematically equivalent to the Gram Schmidt algorithm discussed above, with the only difference being that in the modified version, we take each vector, and modify all **forthcoming vectors** to be orthogonal to it, thus always orthogonalizing against **already computed** column vectors.

Let $A = (A_1|A_2|...|A_n)$ like before, and initialize the matrix U as $U_1 = A_1, U_2 = A_2$,..., $U_n = A_n$. We then calculate the matrix Q using the following procedure

$$\boldsymbol{Q_1} = \frac{\boldsymbol{U_1}}{||\boldsymbol{U_1}||}, \quad \boldsymbol{U_j} = \boldsymbol{U_j} - (\boldsymbol{Q_1}^T \boldsymbol{U_j})\boldsymbol{Q_1},\ 2 <= j <= n$$
$$\boldsymbol{Q_2} = \frac{\boldsymbol{U_2}}{||\boldsymbol{U_2}||}, \quad \boldsymbol{U_j} = \boldsymbol{U_j} - (\boldsymbol{Q_2}^T \boldsymbol{U_j})\boldsymbol{Q_2},\ 3 <= j <= n$$
$$\vdots$$
$$\boldsymbol{Q_n} = \frac{\boldsymbol{U_n}}{||\boldsymbol{U_n}||}$$

Once the matrix Q is found, R can be calculated similar to the above. The advantage here is Q as calculated with the *Modified Gram Schmidt* method has much smaller errors than by that calculated by the regular *Gram Schmidt* algorithm.

Pseudo code for Modified Gram-Schmidt can be represented as:

Algorithm 1 The Modified Gram-Schmidt Algorithm

```
for i = 1 to n do
    r_ii ← ||A_i||                 {Diagonal elements of R}
    Q_i ← (1/r_ii) A_i             {Columns of Q}
end for
for k = 1 to n; j = (k+1) to n do
    r_kj ← Q_k^T A_j               {Non-diagonal elements of R}
    A_j ← A_j − r_kj Q_k           {Update A}
end for
Q := (Q_n Q_{n−1} ... Q_1)^T
R := A
```

C. Householder transformation

The Householder transformation reflects a given vector about a mirror plane with the plane passing through origin. It can be used to transform the first column vector of the matrix and then subsequent (i, i) minors of the matrix into a standard basis vectors, converting the original matrix into a upper-triangular matrix.

The normal vector to the mirror plane are known as Householder reflectors. Using these reflectors we create orthogonal Householder matrix and multiply them to the left of original matrix A to get the upper triangular matrix R.

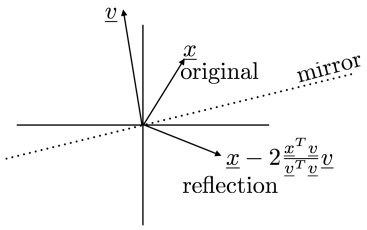

Fig. 1. Householder reflector

Assume the dimensions of A is $n \times n$. Take a column vector $\boldsymbol{x}$, of dimension $(n+1-i)$ from the (i, i) element of A such that $\boldsymbol{x} = (a_{ii} a_{(i+1)i} \dots a_{ni})^T$, where $i \in \{1, \dots, n-1\}$. Then the Householder reflector $\boldsymbol{v}$ will be

$$\boldsymbol{v} = \boldsymbol{x} - \|\boldsymbol{x}\|\boldsymbol{e_1}$$

where $\boldsymbol{e_1} = (1, 0, \dots, 0)^T$ is a standard basis vector of dimension $(n + 1 - i)$ and $\|.\|$ is the Euclidean norm.

Consider a matrix H_{i+1} of dimension $(n+1-i) \times (n+1-i)$ as,

$$H_i = I_{n+1-i} - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\|\boldsymbol{v}\| \cdot \|\boldsymbol{v}\|}$$

, where I_{n+1-i} is the identity matrix of dimension $(n + 1 - i) \times (n + 1 - i)$. This matrix H_i maps the column vector $\boldsymbol{x}$ to the standard basis vector $\|\boldsymbol{x}\| \cdot \boldsymbol{e_1}$,

$$H_i \boldsymbol{x} = \boldsymbol{x} - 2\frac{\boldsymbol{v}(\boldsymbol{v}^T \boldsymbol{x})}{\|\boldsymbol{v}\|^2} = \boldsymbol{x} - \boldsymbol{v} = \|\boldsymbol{x}\|\boldsymbol{e_1}$$

This matrix is also called as the Householder matrix.

Hence on multiplying the H_i's to the left of the (i,i) minor sub-matrices of A will map A to the upper-triangular matrix R. But H_i's have dimension $(n+1-i)\times(n+1-i)$ and A is a matrix of size $n\times n$. Consider a matrix Q_i as,

$$Q_i = \begin{bmatrix} I_{i-1} & 0 \\ 0 & H_i \end{bmatrix}$$

where I_{i-1} is an identity matrix of dimension $(i-1)\times(i-1)$. Then multiplying Q_i to the left of A is similar as multiplying the H_i's to the left of the (i,i) minor sub-matrices of A. Hence on multiplying Q_i to the left of A we get the upper-triangular matrix R and orthogonal matrix Q as,

$$Q_{n-1}Q_{n-2}\dots Q_1 A = R = Q^T A$$

$$\implies \quad Q = (Q_{n-1}Q_{n-2}\dots Q_1)^T$$

Algorithm 2 Algorithm for the Householder transformation

for $j := 1$ to $n-1$ **do**
  $\boldsymbol{x} := A(j:n, j)$;
  $\boldsymbol{v}_j := \boldsymbol{x} - \|\boldsymbol{x}\|_2 \cdot \boldsymbol{e}_1$;
  $\boldsymbol{v}_j = \boldsymbol{v}_j / \|\boldsymbol{v}_j\|_2$;
  $H := I - 2 < \boldsymbol{v}_j, \boldsymbol{v}_j^T >$;
  for $i = j$ to n **do**
    $A(j:n, j:n) = H(j:n, i:n) \cdot A(i:n, j:n)$
  end for
  $Q_j = [I(0:j-1, 0:j-1), H(j:n, j:n)]$
end for
$Q := (Q_{n-1}Q_{n-2}\dots Q_1)^T$
$R := A$

IV. Results

The analysis of the OpenMP implementation of the Modified Gram-Schmidt algorithm involved

- Variation of execution time of the program with the number of threads keeping the matrix size constant
- Variation of execution time of the program with the size of the matrix A keeping the number of threads fixed

The results are summarized in Figures 2 and 3 respectively.

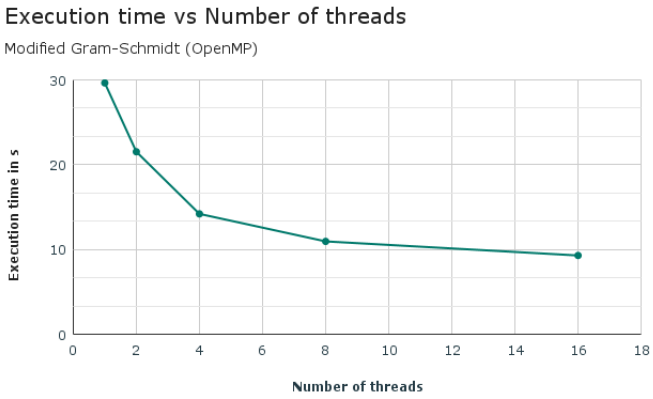

Fig. 2. Variation of execution time with number of threads ($n = 2000$)

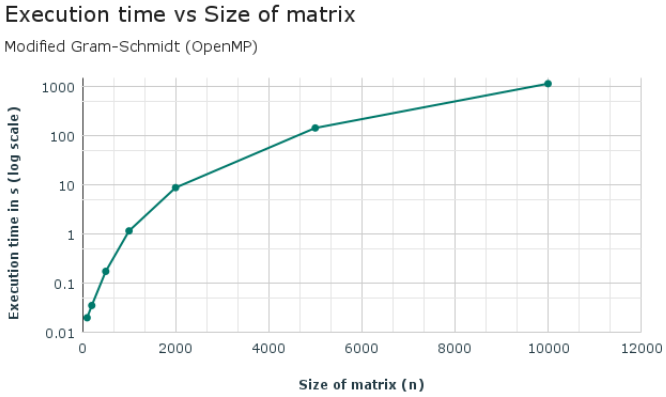

Fig. 3. Variation of execution time (log scale) with matrix size (16 threads)

The analysis of the CUDA implementation of the Modified Gram-Schmidt algorithm involved variation of the size of the matrix. The results are summarised in Figure 4

The analysis of the OpenMP implementation of the Householder transformation for QR matrix factorization is similar to that of the OpenMP implementation of Modified Gram-Schmidt algorithm. The results are summarized in Figures 6 and 5 respectively.

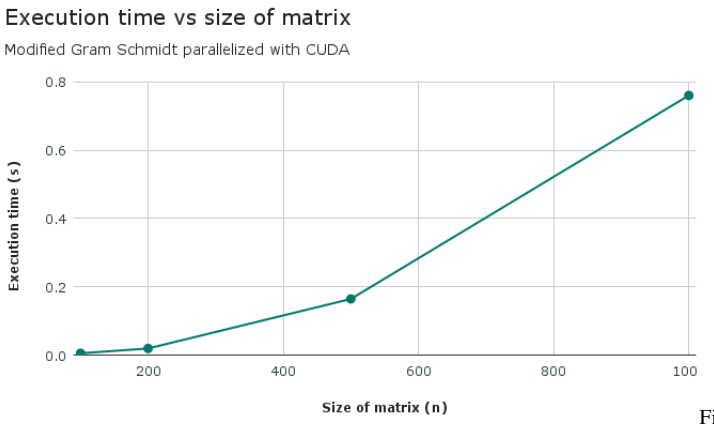

Fig. 4. Variation of execution time with size of matrix (n)

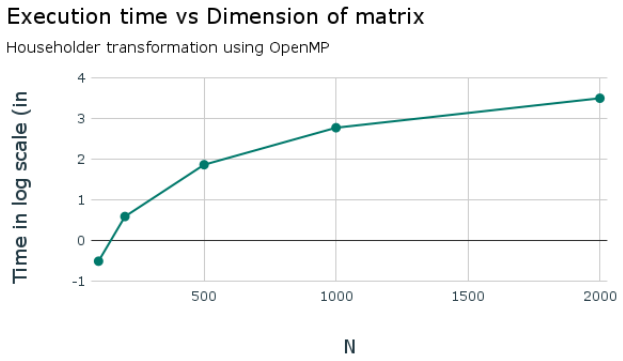

Fig. 5. Execution time vs size of matrix (N) (for threads = 16)

V. Hardware and environment specifications

This report is a collaboration between three students and hence the hardware specifications are different for different parallel implementations of algorithms.

A. Modified Gram-Schmidt: OpenMP

- System Model: HP Pavilion Notebook
- System Type: x64-based PC
- Processor: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, 2304 Mhz, 4 Core(s), 8 Logical Processor(s)

B. Modified Gram-Schmidt: CUDA

- Environment: Google Colaboratory
- CPU: Intel(R) Xeon(R) CPU @ 2.20GHz
- GPU: Nvidia Tesla P100
- CUDA version: v11.2

C. Householder transformation: OpenMP

- OS Name: Microsoft Windows 10 Home Single Language
- System Type: x64-based PC
- Processor: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz, 2400 Mhz, 4 Core(s), 8 Logical Processor(s)

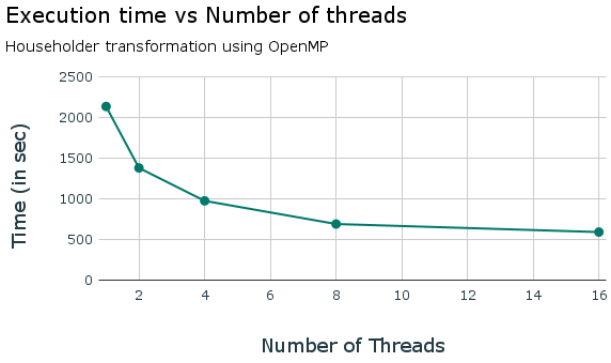

Fig. 6. Variation of execution time with number of threads ($N = 1000$)

VI. Conclusions

The results of the time study of different implementations led to the following conclusions:

- The execution time varies inversely with the number of threads for both the algorithms.
- The complexity of the Modified Gram-Schmidt algorithm is approximately $\mathcal{O}(n^3)$.
- The QR decomposition method using Householder transformation is computationally expensive as it involves matrix multiplication within an iterative loop. This makes the overall time complexity of the algorithm to be around $\mathcal{O}(n^4)$. When dealing with the matrix size of $n \geq 500$ in matrix multiplication, memory latency becomes a significant issue which is countered by an optimization technique known as *Loop Blocking*.

It was observed that the CUDA implementation for the Modified Gram Schmidt algorithm was marginally faster than the OpenMP implementation, but both had the same order of magnitude of execution time.

VII. Acknowledgements

We would like to thank Prof. Sivasubramaniam Goplakrishnan for giving us an opportunity to work on such an amazing project. We would also like to thank all our peers in this course who helped us with appropriate guidance at different times.

References

[1] Prof. Michael T. Heath, Department of Computer Science,University of Illinois at Urbana-Champaign, "Parallel Numerical Algorithms", Lecture Slides - CS 554 / CSE 512

[2] Aaron Schlegel, "QR Decomposition with the Gram-Schmidt Procedure", https://rpubs.com/aaronsc32/qr-decomposition-gram-schmidt

[3] Dianne P. O'Leary, Peter Whitman, "Parallel QR Factorization by Householder and modified Gram-Schmodt algorithms", Parallel Computing 16, North Holland, 1990