Chapter 1

Introduction

The QR factorization is one of the fundamental matrix decompositions with applications throughout scientific computing and data analysis. For matrices with many more rows than columns,they are called “tall-and-skinny matrices”. These matrices generally occur in eg. over determined systems image processing, finding nullspaces of polynomial matrices. Their coloumn to row ratio is generally 1000:1. For the course project we have attempted to implement a ”tall skinny QR” algorithm [1].

Tool Used:

1. Software
    - OpenCL [2]
    - Intel MKL [3]
2. Hardware
    - Intel i3 second gen.
    - AMD Radeon HD5450

Chapter 2

Algorithm

The matrix **A** is tall, as a result the vectors need to compute the qr will be very long and since the process of computing norms uses only single threads, the cess will be time consuming. The idea is to split the a matrix as shown below and compute multiple QR parallely uisng the concept of work groups.

$$A = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix}$$

As a result of this splitting the **A** matrix the **Q**'s generated will have to be stacked as shown below to get the **Q** for the iteration.

$$\mathbf{Q} = \left(\begin{array}{c|c|c|c} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c|c} Q_{01} & \\ \hline & Q_{23} \end{array}\right) \cdot \begin{pmatrix} Q_{0123} \end{pmatrix}$$

Since the **Q** matrix will be square the multiplication of **Q**'s from the i^{th} and the $(i+1)^{th}$ iteration will be sparse, this multiplication will be done using the intel MKL library for sparse maultiplication. The intel MKl library has well defined routines for sparse multiplication.

The **R**'s can be stacked one above the other and can be passed as uasual to the next iteration. The advantage of doing this is that the size of **Q** matrix is always tall and the **R** from the last iteration will be the final **R**, unlike the search for **Q** will requires multiplication of **Q** from the previous and the present iteration.

$$\mathbf{QR}\left(\frac{R_{00}}{R_{10}}\right) = [Q_{01}][R_{01}]$$

An overview of the total flow for the algorithm.

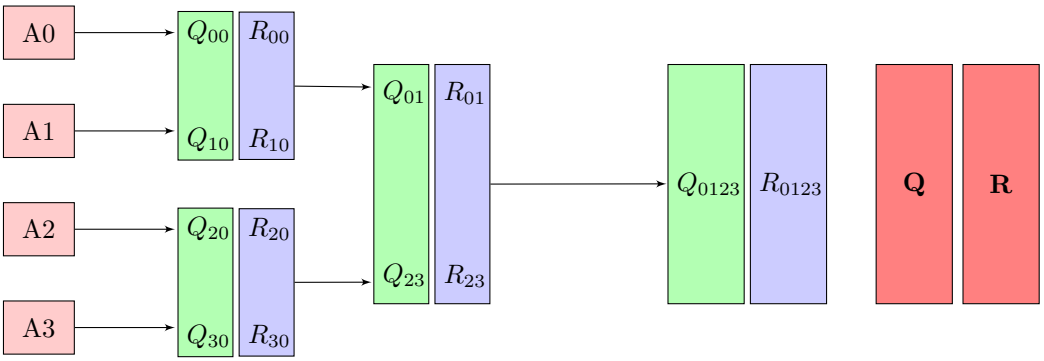

Figure 2.1: Parallel QR Decomposition on GPU using OpenCL

Chapter 3

Challenges

1. Implement a Block based parallel QR.
2. Single block-QR algo for non-square matrices.
3. Integrate OpenCL and Intel MKL.

Bibliography

[1] James Demmel, Laura Grigori, Mark Frederick Hoemmen, Julien Langou *"Communication-optimal parallel and sequential QR and LU factorizations - Technical Report No. UCB/EECS-2008-89"*, Aug 2008.

[2] Khronos OpenCL Working Group, *"The OpenCL Specification"*, version 1.1.

[3] Intel MKL Documentation - *"software.intel.com/en-us/articles/intel-math-kernel-library-documentation"*

[4] Benedict R. Gaster, Lee Howes, David Kaeli, Perhaad Mistry and Dana Schaa *"Heterogeneous Computing with OpenCL"*.

[5] Aaftab Munshi, Benedict R. Gaster, Timothy G. Mattson, James Fung and Dan Ginsburg *"OpenCL Programming Guide"*.