Project Proposal: Parallel QR Factorization

Jiarui Zhang (jiaruiz), Wangshu Li (wangshul)

1. Project Web Page URL

https://liwangshu.github.io/QRparallel/

2. Summary

We aim to implement several parallel versions of matrix QR factorization, measure and compare their performance, and analyze the best-performed implementation.

3. Background

In linear algebra, a QR factorization is a decomposition of a matrix *A* into a product *A* = *QR* of an orthogonal matrix *Q* and an upper triangular matrix *R*. QR decomposition is often used to solve linear systems and least squares problems and is the basis for a particular eigenvalue algorithm, the QR algorithm. The process of QR factorization is iterative and involves much matrix computation. This feature makes it easy to benefit from parallelism, improving the efficiency of solving linear systems.

4. Challenge

One challenge is the data partition and synchronization. To obtain a good performance in parallel computing, we need to distribute tasks evenly to workers, while considering the cost of communication. Therefore, how to partition the matrix is a major concern. In the shared address space model, all threads are sharing the same input matrix, which may lead to frequent synchronization. In the message passing model, processors need to send and receive data explicitly. Balancing the workload as much as possible while minimizing the communication overhead is the final goal.

Another possible challenge is profiling and optimization. It is likely that the performance cannot reach expectations at first, so profiling and finding the bottleneck is necessary. The factorization algorithm consists of multiple steps. We need to think about strategies to improve performance of steps taking up too much execution time.

5. Resources

We will start from scratch, following the mathematical formula on this website (https://www.baeldung.com/cs/solving-system-linear-equations) to implement the sequential version first, then we will use GHC machines to test and measure the performance of CUDA version and OpenMP version and use PSC machines for MPI version.

6. Goals and Deliverables v

Plan to achieve: We plan to implement the sequential version of QR factorization first, and then parallelize it in three different ways, including using CUDA threads in GPU, OpenMP and MPI. Then we will pick one with the best performance and analyze it.

Hope to achieve: There are some existing parallel QR factorization algorithms. We will further optimize one of our parallel programming models and hope our model can achieve similar or better performance compared to the existing algorithms on the same type of machines.

We will plot some graphs to show the speedup of different parallel programming models over the sequential version as the visualization result of our demo.

Our project is an analysis project, so we want to compare these three programming models and find out which one has the highest speedup on the same type of machine. We also want to analyze why some models are better than others in QR factorization considering the communication, data movement, and synchronization cost.

7. Platform Choice

We will use GHC machines and PSC machines to test and measure the performance of our code(As mentioned in the Resources part). GHC machines have enough cores for us to parallelize our algorithm and it also has the coding environment that is necessary for these programming models.

We will use C++ for this project, because C++ is compatible with all the programming models we will use such as CUDA, OpenMP and MPI.

8. Schedule

Week 1 (11/7 - 11/13)	(Project proposal due) Research on ideas to parallel algorithm
Week 2 (11/14 - 11/20)	Baseline sequential implementation of QR factorization, profile code to figure out bottlenecks
Week 3 (11/21 - 11/27)	Implement parallel versions of baseline code
Week 4 (11/28 - 12/4)	(Milestone Report due) Implement more parallel versions and compare the results
Week 5 (12/5 - 12/11)	Analyze the results and best-performed model
Week 6 (12/12 - 12/18)	Final report & demo