EE 522 Project - Quantum Message Passing Algorithm Implementation (QMPI): Midterm Report

Tommy Nguyen[1], Yue Shi[1], Ang Li[2], Samuel Stein[2], Tim Stavenger[2], Marvin Warner[2], Martin Roetteler[3], Peter J. Pauzauskie[1], and Torsten Hoefler[4]

[1]University of Washington
[2]Pacific Northwest National Labs
[3]Microsoft
[4]ETH Zürich

May 1, 2023

Contents

1 Abstract

Quantum computers today are noisy and are limited in the number of qubits that can be set with reasonable fidelities and gate times on each system. Since the current limit on the number of qubits is difficult to surpass, it is reasonable to expect that the future of quantum computing will lie in connecting different quantum computers together to form a quantum network, each with its own limits. Here lies the field of distributed quantum computing and the development of distributed systems and algorithms optimized for quantum systems will be extremely important going forward. One such system that has been designed is the Quantum Message Passing Interface (QMPI) [2]. This interface, unlike the classical MPI, takes advantage of quantum teleportation to communicate between different quantum nodes but leverages the framework built for classical MPI. This sort of general interface is important as we try to connect different quantum computers, as it can be used for the development of distributed quantum algorithms across different nodes. This is what

has already been built in this project and we next plan to characterize its performance concerns using Qiskit. This includes the point-to-point and collective operations and optimizations for larger distributed quantum networks (already done). To test our framework, we demonstrate our implementation with applications that can leverage the MPI framework including the transverse field Ising model, quantum arithmetic units, and distributive quantum phase estimation.

2 Project Background

The Quantum Message Passing Interface is a communication framework designed for the distributed quantum computation system, inspired by the classical Message Passing Interface used in high-performance classical computing. In the distributed system, the smaller quantum chips are connected coherently, allowing for inter-node communication of quantum information.[2]

2.1 Project Description

The objective of this project is to implement and develop the operations of QMPI in Qiskit and finally get an interface library for running distributed quantum computing. Although the QMPI concept has been proposed recently, the practical implementation is still missing. Nevertheless, there are a few papers that demonstrated the preliminary strategies to build up the QMPI framework. The literature review based on recent papers will be shown in this section.

2.2 Past work

2.2.1 QMPI

This paper first proposes the concept of the Quantum Message Passing Interface to enable high-performance implementations of distributed quantum computing. The operations introduced in this paper can be divided into two classes. One is point-to-point communication operations including the Send, Recv operation, and so on. Another class is collective communication including the Bcast, Gather, Scatter, and so on. The paper also presented a table of the resource requirements of non-local operations of distributed quantum algorithm primitives such as EPR pair, and classical registers. [2]

2.2.2 CutQC

This paper introduces a scalable hybrid computing approach to cut large quantum circuits into smaller subcircuits, allowing them to be executed on smaller quantum devices. The underlying theory behind the methods originates from the fact that the unitary matrix of an arbitrary quantum operation in a quantum circuit can be decomposed into any set of orthonormal matrix bases. [3]

2.2.3 AutoComm

AutoComm is an automatic compiler framework to extract burst communication patterns from input programs and then optimizes the communication steps of burst communication. Here, burst communication is a specific qubit-node communication pattern introduced in this paper that widely exists in various distributed quantum algorithms and can be leveraged to optimize communication overhead. By taking advantage of the burst communication, AutoComm can reduce the communication resource consumption and the program latency by 72.9% and 69.2% respectively on average in simulation. [5]

2.2.4 CollComm

CollComm is a new strategy discovered to boost the efficiency of quantum communication by decoupling communication resources from remote operations. This is done by buffering entangled particle (EPR) pairs generated by the communication hardware in the qubits of the computational hardware. The experimental results show that this approach can significantly reduce the communication cost of various distributed

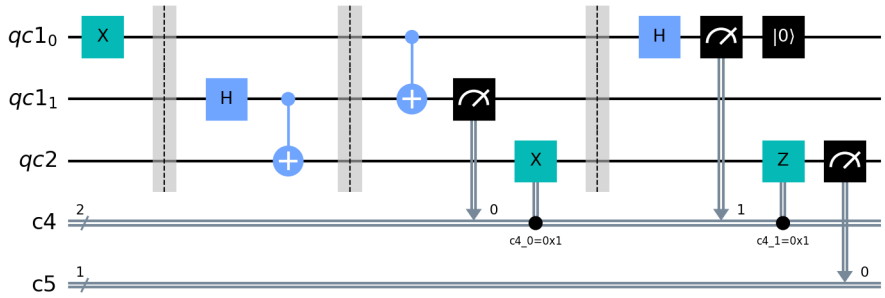


Figure 1: Quantum Teleportation Circuit

quantum programs, with nearly a 50% reduction compared to the state-of-the-art compiler for distributed quantum computing [4]

3 Methods

Our QMPI implementation uses Qiskit and takes advantage of their data structures to build the interface. We designate different quantum registers as different quantum nodes/computers and have the constraint that each node can only make EPR pairs with one other node i.e. three nodes can't make a GHZ state. We don't specify how these EPR pairs are created as this would be dependent on the hardware, and so any two-qubit gates must be created and optimized with our implementation. We imposed the constraint that each node can only make 2 EPR pairs in total at a time but this constraint can be relaxed and generalized in future implementations. Future work might be to give this parameter to the user so our implementation can be generalized to any abstraction for arbitrary EPR constraints between different quantum nodes. We also assume that each node has unlimited access to classical bits which is a safe assumption given the abundance of classical hardware. All of our work is done on the aer simulator so this assumes lossless communication, but we are working towards incorporating noise models and tests on actual quantum hardware including ionq and IBM quantum computers. We use 1000 shots and assume a tolerance of 5 percent to compare results with and without our implementation.

To communicate between different quantum computers only having access to EPR pairs, we use the quantum teleportation circuit as seen in 1. This circuit teleports the state from one quantum node to the other which we can then use to do two qubit gates. We are working on towards implementing the catComm and tpCOMM methods and optimizing these for general quantum circuits.[5]. We are working towards building a model to characterize the distributed QMPI framework we are building. We currently have the ability to track how many EPR pairs are created but haven't characterized our circuits out completely for optimizations. We are thinking of using the SENDQ model in [2] or the model in [1] which counts the number of EPR pairs used to optimize over. We are currently using the open source QASMBench benchmark suite from PNNL to test our implementation.

4 Results and Analysis

4.1 Point Communication

We have implemented the point-to-point operations for the QMPI framework which includes the send and receive functions. Our code can be found here: (INCLUDE LINK). We didn't implement the buffering point communications as we are unsure of how this would look. This could either be in the form of heralded vs. unheradled entanglement for buffering, or using classical MPI to buffer whenever we perform a measurement for sending and receiveing qubits. As a general test of our implementation, we use the circuit in 2. This tests multiple EPR pairs transferring between different nodes, mid-circuit measurements between the nodes,

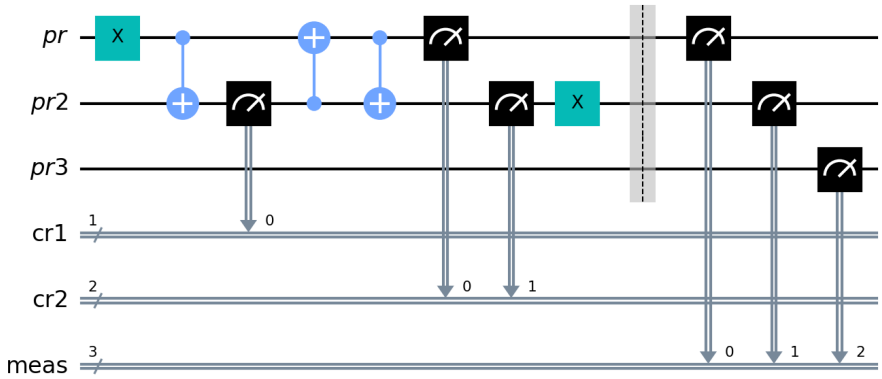


Figure 2: Quantum circuit to test our point-to-point operations.

and other operations. We have also built the framework that for any arbitrary quantum circuit, our QMPI program can take it in and rewrite the circuit for point-to-point commnication given that each quantum register is a different quantum node. To test this we have tested the adder, grover, deutsch, and error correction circuits and they give the expected results with our point-to-point implementation. We haven't compared fidelities and latency but this will be implemented soon!

4.2 Collective Operations

We have implemented the broadcast, scatter, and gather collective operations for the QMPI framework. These operations are called upon by the user whenever a collective operation would like to be used. We are working towards optimizing these and coming up with a way to implement allgather, reduce, all to all, scan and allreduce. We are also working towards testing these with some quantum circuits that we can creatively come up with that take advantage of this framework. Additionally, applications including the ising model, phase estimation, and other applications with a large number of quantum nodes that can take advantage of this collective QMPI framework are currently being investigated. On this note, we are considering building a way similar to that already built with the point QMPI that given any quantum circuit, it would automatically convert it to use the collective operations efficiently.

5 Future Work

There are many possible steps that we can take to improve our implementation! So much so that it would be impossible to implement everything within this course. We list a couple of future directions this work can take that we might have time for before the end of this course:

1. Look into alternative teleportation multi-qubit schemes. For instance, are there better ways to teleport two qubits at once than doing each of them sequentially?

2. Optimization within our implementation for parallelization using the autocomm framework and implementing an EPR buffer for increased optimization as specified in the collcomm framework. Both of these ideas would decrease the latency and increase the resource usage for arbitrary quantum circuits. In a sense, this is a compiling problem. We also would like to consider entanglement distillation for optimizations.

3. Optimizing the collective operations of QMPI. For instance, use a minimum spanning tree (MST) for broadcast instead of the current greedy approach. This is an entirely new field so we will have to be very creative here and think of ways of utilizing the power of quantum entanglement to implement the collective operations more efficiently.

4. Implement the buffering and Rsend operations. This could either be in the form of being hardware-specific or implementing classical MPI on top of our QMPI.

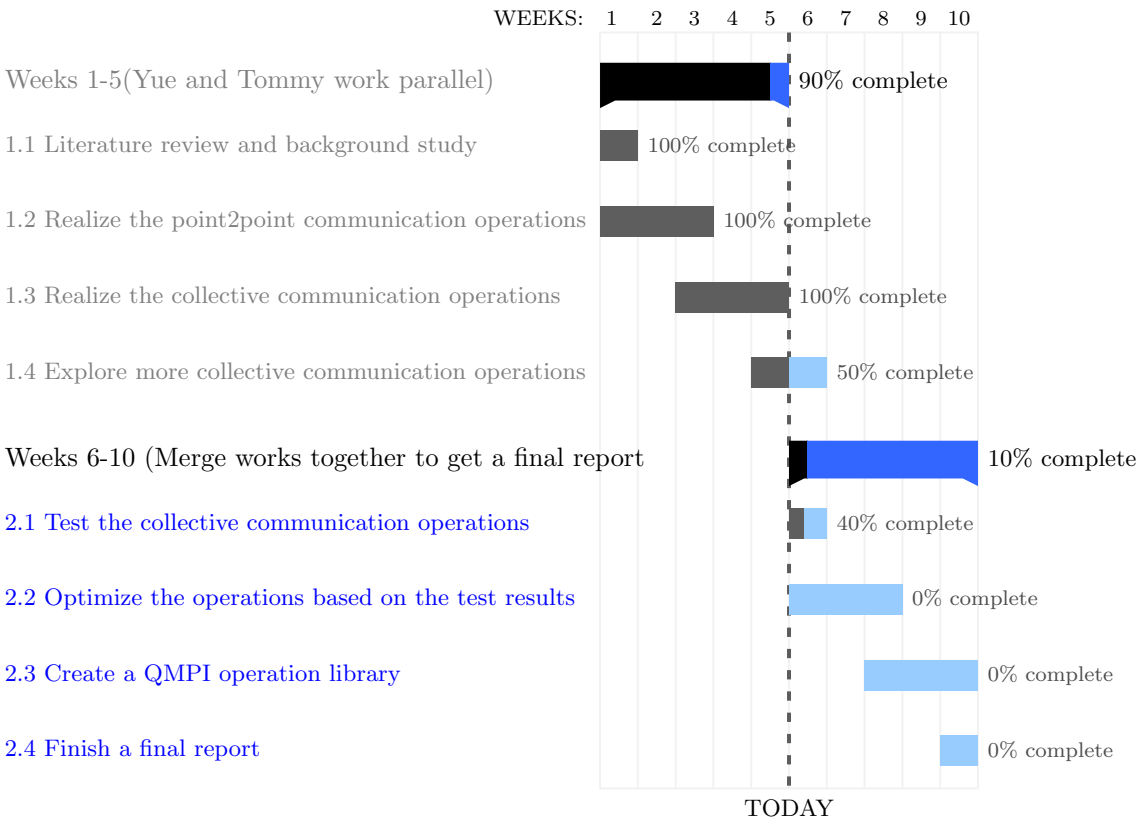
5. For our implementation, we require the user to specify which qubits belong to which quantum node. However, one can imagine a more general case where this can be optimized and left to our implementation. Our work would be able to distribute and allocate which qubits would be best on different quantum nodes for an arbitrary quantum circuit. Of course, this would be dependent on the quantum hardware including local gate times and fidelity and its availability to connect to different quantum nodes, and the efficiency that it can make EPR pairs to other quantum nodes.

6. Our current implementation relies on the quantum circuit being decomposed into cx gates for two-qubit operations between nodes. However, this may not be the most efficient gate based on hardware so having this be more general would have practical applications.

There is no way we will be able to do all of this and so some risk and mitigation strategies would be to implement the ones we think we can do given the timeline of this course and not to be overly ambitious. Another strategy would be to break down problems into logical steps that we can implement slowly but surely. Another thing is to make sure that we always test our implementation to handle all niche cases. For anything that we can't get to, perhaps this project can be continued over the summer or by another student as there is a lot of rich and interesting material to implement within this framework.

6 Project Schedule

Below is the updated gantt chart which holds the initial plan in weeks 1-5 and updates for weeks 6 onwards. The task breakdown is that Tommy and Yue are working in parallel on the tasks. Tommy is focusing on building up the framework for the QMPI and Yue is focusing on optimizations with autocomm and collcomm and incorporating noise models.



References

[1] James Ang, Gabriella Carini, Yanzhu Chen, Isaac Chuang, Michael Austin DeMarco, Sophia E. Economou, Alec Eickbusch, Andrei Faraon, Kai-Mei Fu, Steven M. Girvin, Michael Hatridge, Andrew Houck, Paul Hilaire, Kevin Krsulich, Ang Li, Chenxu Liu, Yuan Liu, Margaret Martonosi, David C. McKay, James Misewich, Mark Ritter, Robert J. Schoelkopf, Samuel A. Stein, Sara Sussman, Hong X. Tang, Wei Tang, Teague Tomesh, Norm M. Tubman, Chen Wang, Nathan Wiebe, Yong-Xin Yao, Dillon C. Yost, and Yiyu Zhou. Architectures for multinode superconducting quantum computers, 2022.

[2] Thomas Häner, Damian S. Steiger, Torsten Hoefler, and Matthias Troyer. Distributed quantum computing with qmpi. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. CutQC: using small quantum computers for large quantum circuit evaluations. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, apr 2021.

[4] Anbang Wu, Yufei Ding, and Ang Li. Collcomm: Enabling efficient collective quantum communication based on epr buffering, 2022.

[5] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1027–1041, 2022.