Thesis Proposal

Hybrid Quantum–Classical Software Stack for HPC Algorithm Acceleration

Anna Payne

Summary

The fields of quantum computing and high-performance computing together present an opportunity to design hybrid systems that will leverage the strengths of both paradigms. Classical HPC infrastructures, using MPI, CUDA, and multi-node CPU/GPU clusters, excel with heavy parallel numerical workloads and large scale simulations. Quantum computing, through the use of Quantum Processing Units (QPUs) will offer potential advantages in certain algorithms that involve optimization, search, sampling, and simulation. Currently, QPUs are limited in size, coherence, and error correction, requiring the use of hybrid quantum-classical algorithms such as the Variational Quantum Eigensolver (VQE) (Peruzzo et al., 2014) or the Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al., 2014), which offloads the computationally intensive classical optimization loop to conventional computing resources.

This project involves a **hybrid quantum-classical software framework** that is capable of dispatching computational kernels to either QPUs or to classical HPC nodes depending on performance characteristics, algorithm types, and resource availability. The goal is to determine if quantum accelerators embedded into HPC workflows can provide advantages over classical-only implementations. The primary bottleneck to deploying these algorithms is the lack of a highly efficient and stable software stack that seamlessly integrates the quantum and classical components. The existing quantum programming frameworks, such as Qiskit or Cirq, are excellent at QPU control but lack the deep integration with HPC environments. Classical optimization steps (evaluating the objective function, running error mitigation routines, performing extensive classical simulations) demand extensive computational power and typically rely on parallel execution using MPI for multi-node communication and CUDA for GPU acceleration. This hybrid quantum-classical software stack would be designed to bridge the gap by focusing on three key areas:

1. **API Design**: Defining clear, intuitive interfaces that mask the underlying complexity, therefore allowing users to define a unified workflow with automatically assigned subroutines that are executed on either QPU or HPC resources.
2. **Implementation**: Developing the dispatching logic, building the HPC connectors for Qiskit frameworks, and implementing parallel classical kernels with the coordination using MPI and CUDA for accelerating numerical tasks.
3. **Benchmarking**: Implementing end-to-end workflows and conducting extensive performance analysis on multi-node HPU clusters.

The outcome would be a robust, open source framework and a thorough analysis demonstrating the measurable acceleration and scalability achieved by this integration. This will pave the way for larger, more extensive quantum-classical simulations in materials science or in optimization.

Research Objective

To what extent can the developed Hybrid Quantum-Classical Software Stack improve the speed and scalability of end-to- end VQE workflows when deployed on a multi-node cluster, and compared to its performance on a single classical node or a standard, non-optimized quantum framework?

Objective: To design, implement, and benchmark a HPC software framework that achieves a measurable workflow acceleration by dynamically integrating QPUs with multi-node classical HPC resources (MPI, CUDA).

Desired Measurable Outcome

The desired outcome would be a successfully implemented, functionally complete Hybrid Quantum-Classical Software Stack, quantified by:

1. Workflow acceleration: Achieving a 2-3x minimum speedup in the end-to-end wall clock time for the classical optimization loop in the VQE workflow when scaling from N=1 to N=4+ HPC nodes.
2. Scalability: Robust, near-linear scaling of the classical components performance as the number of the HPC nodes or complexity of the classical task increases.
3. API usability: Well documented, modular API that will simplify the creation of hybrid workflows and is measurable by the reduced line of code complexity compared to building the same workflow manually using Qiskit, MPI, and CUDA libraries separately.

Literature Review

This project references other relevant areas of research:

1. **Quantum-classical hybrid algorithms**: The foundation of this project lies in algorithms such as the VQE algorithm which combines a reconfigurable photonic quantum processor with a conventional computer in order to efficiently find the eigenvalue of a given eigenvector while reducing coherence time requirements for the qubits (Peruzzo et al., 2014). Another algorithm referenced in this project is the Quantum Approximate Optimization Algorithm, which produces approximate solutions for optimization problems while balancing quantum-classical components (Farhi et al., 2014).

2. **Quantum Software Stacks**: Current frameworks such as IBM's Qiskit or Google's Cirq provide the fundamental tools for defining and executing quantum circuits (along with advancements for the speed and efficiency of QC workloads) (*Qiskit Runtime Makes Algorithm Development Easier Than Ever | IBM Quantum Computing Blog*, n.d.), but this proposed project will require a focus on the underlying middleware that is necessary for multinode, heterogeneous workload distributions.

Execution Plan

Phase 1: Design & Architecture

Mid Dec - Jan (2-3 weeks)

Define the QC/Classical API specification, dispatcher logic, data structures. Set up an HPC/QPU connection environment.

Phase 2: Classical Implementation

Jan - Mid Jan (2 weeks)

Implement the classical kernels using MPI for parallel distribution and CUDA for GPU acceleration. Develop data serialization/deserialization layer for QPU/HPC communication.

Phase 3: Quantum Implementation

Mid Jan - Feb 26th (3-4 weeks)

Integrate Qiskit as the QPU backend connector. Implement the dynamic workload dispatchers, ensure fault tolerance and efficient resource management across hybrid stack. Submit first draft Jan 26th (with quantum integration introduced).

Phase 4: Benchmarking & Analysis

Feb 26th - March 26th (1 month)

Implement a suite of benchmarks on the application, measuring wall-clock time, speedup, scalability on a varying number of classical nodes. Submit second draft Feb 26th and the full draft March 26th.

Phase 5: Thesis Writing

March 26th - April 26th (1 month)

Final analysis and documentation of the framework written. Submission of the final thesis on April 26th.

Bibliography

Farhi, E., Goldstone, J., & Gutmann, S. (2014, November 14). *A Quantum Approximate Optimization algorithm*. arXiv.org. https://arxiv.org/abs/1411.4028

Peruzzo, A., McClean, J., Shadbolt, P., Yung, M., Zhou, X., Love, P. J., Aspuru-Guzik, A., & O'Brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, *5*(1), 4213. https://doi.org/10.1038/ncomms5213

Qiskit Runtime makes algorithm development easier than ever | IBM Quantum Computing Blog. (n.d.). https://www.ibm.com/quantum/blog/qiskit-runtime-for-useful-quantum-computing