ENABLING A PROGRAMMING ENVIRONMENT FOR AN EXPERIMENTAL ION TRAP QUANTUM TESTBED

A Thesis
Presented to
The Academic Faculty

By

Austin Adams

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science
School of Computer Science

Georgia Institute of Technology

May 2022

ENABLING A PROGRAMMING ENVIRONMENT FOR AN EXPERIMENTAL ION TRAP QUANTUM TESTBED

Thesis committee:

Dr. Thomas Conte
School of Computer Science
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Jeffrey Young
School of Computer Science
Georgia Institute of Technology

Dr. Santosh Pande
School of Computer Science
Georgia Institute of Technology

Date approved: April 29, 2022

Dedicated to Tillman Bell

ACKNOWLEDGMENTS

First, I would like to thank Dr. Tom Conte and Dr. Jeff Young for hiring me and guiding me along the way. I also must thank Dr. Creston Herold for our many hours of discussions and his thorough review. Thanks to Elton Pinto for his help in drafting the Related Work section.

Thank you to my parents for creating me and raising me, neither of which I made easy. I also want to thank Brian Campbell and Joel Adams, who taught me more than I can write here. Finally, I thank my brother Travis Adams for never leaving my side.

I gratefully acknowledge partial support from the Institute for Electronics and Nanotechnology's Georgia Tech Quantum Alliance. I also acknowledge support for this work from NSF planning grant #2016666, "Enabling Quantum Computer Science and Engineering" and through the ORNL STAQCS project. Finally, I was supported in part through research infrastructure and services provided by the Rogues Gallery testbed [1] hosted by the Center for Research into Novel Computing Hierarchies (CRNCH) at Georgia Tech and funded by NSF Award Number #2016701.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Ion trap quantum hardware promises to provide a computational advantage over classical computing for specific problem spaces while also providing an alternative hardware implementation path to cryogenic quantum systems as typified by IBM's quantum hardware. However, programming ion trap systems currently requires both strategies to mitigate high levels of noise and also tools to ease the challenge of programming these systems with pulse- or gate-level operations.

This thesis focuses on improving the state-of-the-art for quantum programming of ion trap testbeds through the use of a quantum language specification, QCOR, and by demonstrating multi-level optimizations at the language, intermediate representation, and hardware backend levels. A new QCOR/XACC backend is implemented to target a general ion trap testbed and then demonstrate the usage of multi-level optimizations to improve circuit fidelity and to reduce gate count. These techniques include the usage of a backend-specific numerical optimizer and physical gate optimizations to minimize the number of native instructions sent to the hardware. The new compiler backend is evaluated using several QCOR benchmark programs, finding that on present testbed hardware, the new compiler backend maintains the number of two-qubit native operations but decreases the number of single-qubit native operations by 1.54 times compared to the previous compiler regime. For projected testbed hardware upgrades, the new compiler sees a reduction in two-qubit native operations by 2.40 times and one-qubit native operations by 6.13 times.

CHAPTER 1
INTRODUCTION

Quantum computing promises new computational capabilities over classical computing with quantum algorithms having a theoretical exponential speedup over some classical algorithms [2]. However, given the high error rates of present qubits (quantum bits), computational capability today reaches only as far as is achievable on NISQ (Noisy Intermediate Scale Quantum) devices, near-term quantum computers with 50-100 qubits which provide highly noisy output [3]. Limited coherence time and high gate error rates require compilers for NISQ systems to minimize the number of quantum gates (instructions) and to embrace error mitigation techniques to increase the likelihood of useful results [4].

This thesis details the implementation of an XACC [5] compiler backend targeting an experimental quantum testbed hosted by the GTRI (Georgia Tech Research Institute) Quantum Systems Division [6]. This new compiler backend provides a hardware-agnostic programmer-driven flow for programming the testbed using quantum circuits written inline in C++ via QCOR [7]. This new QCOR-based implementation provides a contrast with the existing testbed tooling, which is based on proprietary software more oriented towards hardware experts. Moreover, by integrating with QCOR, programmers gain access to existing QCOR tooling, such as quantum assembly parsers, circuit optimizers, a variational workflow, and a standard library of quantum subroutines.

This thesis provides the following contributions:

- Demonstration of a new ion trap backend for XACC that interacts with the low-level capture system used to program and interact with a target ion trap testbed.
- Implementation of optimizations for a quantum program at the hardware-agnostic level by QCOR and at the hardware-specific level in particular backends.

- Exploration of the usage of these optimizations to improve circuit fidelity and reduce native gate count.
- Investigation of native gate count reduction achievable with future hardware upgrades.

For evaluation, a set of QCOR example programs are run against the backend, with the existing control code configured to respond with simulation results. Gate count is compared with the existing compiler for the testbed.

1.1 Motivation for a New Ion Trap Backend

The current approach to interacting with the GTRI testbed requires using IGOR Pro (a programming environment focused on data visualization) and detailed knowledge of the testbed mechanics — generally, it is hardware expert–driven. An email-based submission scheme shown in Figure 1.1 improved this situation by allowing for submission and parsing of quantum assembly input files, but quantum assembly–level programming is neither particularly strong as a programming environment nor suitable for near-term heterogeneous quantum–classical computing. Connecting QCOR to the testbed introduces a more programmer-driven flow, making the GTRI testbed more accessible to a wider audience of quantum and classical programmers who may be unaware of hardware details. To further ensure its usefulness, the new compiler optimizes programmer circuits into as few noisy native operations as possible.

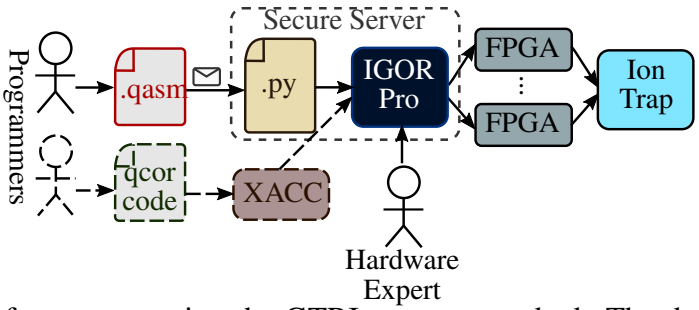

Figure 1.1: Flow for programming the GTRI quantum testbed. The dashed path starting from the bottom left represents the contribution made in this thesis

CHAPTER 2

BACKGROUND

2.1 Ion Trap Quantum Computers

Quantum computers based on an ion trap realize qubits by manipulating the internal spin-like degrees of "trapped" atomic ions with electromagnetic radiation [2]. Native operations can be broadly categorized as either single- or multi-qubit operations which may comprise different universal gate sets. For example, single qubit operations include $R_\phi(\theta)$, a rotation of θ around an axis ϕ in the equatorial plane of the Bloch sphere, shown in Equation 2.2 through Equation 2.3 [8]. The multi-qubit entangling operation considered in this thesis is the Mølmer-Sørenson (MS) interaction [9, 10], which defines the MS gate in Equation 2.5 through Equation 2.6.

$$\sigma_\phi = (\cos\phi)\sigma_x + (\sin\phi)\sigma_y \tag{2.1}$$

$$R_\phi(\theta) = \exp(-i\sigma_\phi\theta/2) \tag{2.2}$$

$$= \begin{bmatrix} \cos\theta/2 & -ie^{-i\phi}\sin\theta/2 \\ -ie^{i\phi}\sin\theta/2 & \cos\theta/2 \end{bmatrix} \tag{2.3}$$

$$\beta_{\ell r} = -ie^{i((-1)^\ell\phi_L + (-1)^r\phi_R)}\sin\alpha \tag{2.4}$$

$$MS(\alpha) = \exp(-i\alpha(\sigma_{\phi_L}\otimes\sigma_{\phi_R})) \tag{2.5}$$

$$= \begin{bmatrix} \cos\alpha & 0 & 0 & \beta_{11} \\ 0 & \cos\alpha & \beta_{10} & 0 \\ 0 & \beta_{01} & \cos\alpha & 0 \\ \beta_{00} & 0 & 0 & \cos\alpha \end{bmatrix} \tag{2.6}$$

When the MS phase angles for the left and right qubits, ϕ_L and ϕ_R respectively, are

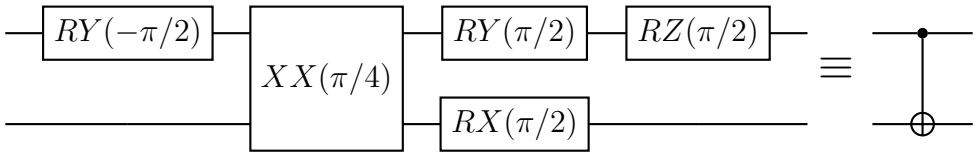

Figure 2.1: Implementation of CNOT on the testbed using single-qubit gates and the native $XX(\pi/4)$ entangling gate. The circuit shown is equal to a CNOT up to an unimportant global phase $e^{-i\pi/4}$

zero, then $\sigma_{\phi_L} = \sigma_{\phi_R} = \sigma_x$, yielding the XX-Ising gate shown in Equation 2.7. For simplicity, this thesis uses this convention. The XX gate can be used to realize a CNOT operation when combined with single qubit gates such as those shown in Figure 2.1 [11, 8].

$$XX(\alpha) = \begin{bmatrix} \cos\alpha & 0 & 0 & -i\sin\alpha \\ 0 & \cos\alpha & -i\sin\alpha & 0 \\ 0 & -i\sin\alpha & \cos\alpha & 0 \\ -i\sin\alpha & 0 & 0 & \cos\alpha \end{bmatrix} \tag{2.7}$$

Together, arbitrary single-qubit gates provided by $R_\phi(\theta)$ [8] and the XX entangling gate support universal quantum computation [12, 13]. Additionally, ion trap systems can support all-to-all qubit connectivity and parallel gate execution using tightly-focused individual gate beams [14] or ion transport with stationary beams [15, 16].

2.2 GTRI Quantum Testbed

Researchers at GTRI have built a quantum testbed based on an ion trap [6, 17]. The physical apparatus consists of a stationary set of lasers which operate on ions (qubits) in the chain. The chain itself is transported to allow the lasers to target different qubits [6]. Originally, the gate laser beams always targeted two ions simultaneously, but due to recent equipment upgrades allowing gate beams to target individual qubits, this thesis assumed single-qubit

Table 2.1: Sequence of Native Operations Produced by the Existing Compiler for the Bell State Circuit `H 0; CNOT 0,1`

Operation	Target Ion	ϕ
$R_\phi(\pi/2)$	0	$-\pi/2$
$R_\phi(\pi/2)$	0	$-\pi/2$
$R_\phi(\pi/2)$	0	$-\pi$
$R_\phi(\pi/2)$	0	$-\pi$
$XX(\pi/4)$	0,1	N/A
$R\phi(\pi/2)$	0	$\pi/2$
$R\phi(\pi/2)$	1	0

addressing. The testbed does not have a tightly-focused laser beam for each ion, so both nearest-neighbor connectivity for two-qubit gates and serialized (i.e., no parallel) single-qubit gates are assumed in this thesis.

When configured as a general-purpose quantum computer, the testbed has the $XX(\pi/4)$ gate and, for ease of calibration, the subset of $R_\phi(\theta)$ gates with $\theta = \pi/2$ as its native operations. The control software for the testbed includes a rudimentary compiler that converts a dialect of quantum assembly to a sequence of native operations ($XX(\pi/4)$ and $R_\phi(\pi/2)$) according to which the control software programs FPGAs as needed to run the circuit on hardware. An example of the sequence for a Bell state circuit is shown in Table 2.1. The existing compiler decomposes CNOTs as shown in Figure 2.1, and it applies arbitrary single-qubit unitaries using an average of 3.25 primitive $R_\phi(\pi/2)$ rotations [6].

2.3 QCOR and XACC

Oak Ridge National Laboratory has developed the QCOR compiler specification [7] and a reference implementation [18]. Both aim to accelerate the development of new applications on NISQ hardware by providing a unified, automated software stack for writing quantum algorithms and mapping them to hybrid classical–quantum systems, such as a CPU-based server paired with a quantum accelerator [19]. In particular, the QCOR implementation

integrates with Clang to allow writing quantum kernels inline in C++ similarly to CUDA kernels, as shown in Figure 2.2.

Behind the scenes, the QCOR implementation uses the XACC framework to parse quantum assembly into quantum IR (Intermediate Representation, implemented as an n-ary tree of quantum gates), transform the IR (e.g., for optimizations), and communicate with accelerators [5]. With its plugin-based architecture, adding an XACC plugin for a new accelerator exposes this new backend on the QCOR level; backends already exist for web APIs for vendors such as IBM, IonQ, and Rigetti, as well as for calls to local simulator libraries.

```
__qpu__ void bernstein_vazirani(qreg q,
        std::string &secret_bits) {
    int n = secret_bits.size();
    // prepare ancilla in |1>
    X(q[n]);
    // input superpositions
    H(q);
    // oracle
    for (int i = 0; i < n; i++) {
        if (secret_bits[i] == '1')
            CX(q[i], q[n]);
    }
    H(q);
    Measure(q.head(n));
}
int main(int argc, char **argv) {
    std::string secret_bits(argv[1]);
    auto q = qalloc(secret_bits.size()+1);
    bernstein_vazirani(q, secret_bits);
    q.print();
}
```

Figure 2.2: Example C++ program using QCOR to generate and execute a Bernstein–Vazirani circuit for a user-provided secret bitstring

CHAPTER 3
BACKEND IMPLEMENTATION

3.1 Overview

This thesis proposes a new backend which consists of an XACC plugin with a new Accelerator for the GTRI testbed. The backend can emit either quantum assembly or a sequence of primitive gates. Small modifications were added to the control software to read inputs from a file in a directory and write simulation outputs to the directory. Future work should evaluate a more robust queueing system than polling a directory on disk. The new backend code has been released as open source online[1].

The remainder of this chapter describes how the new backend prepares a quantum circuit for execution, focusing particularly on decomposing a circuit into primitive testbed operations. The decomposition consists of two passes: the first for two-qubit gates and the second for single-qubit gates, both detailed below. Each pass is an XACC IRTransformation which operates on XACC IR, replacing logical gates with either more logical gates or native operations. After the two passes, the new backend converts the resulting XACC IR to a sequence (or table) of primitive operations and passes it to the control software. The initial implementation used for each compiler pass has time complexity $O(n^2)$, where n is the number of program gates; linear-time implementations would be straightforward but they are reserved for future work, as they would make a negligible difference in runtime for the small benchmarks used for evaluation.

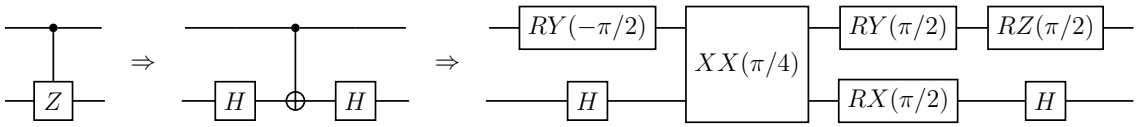

Figure 3.1: An example of how the two-qubit pass would decompose a controlled-Z gate

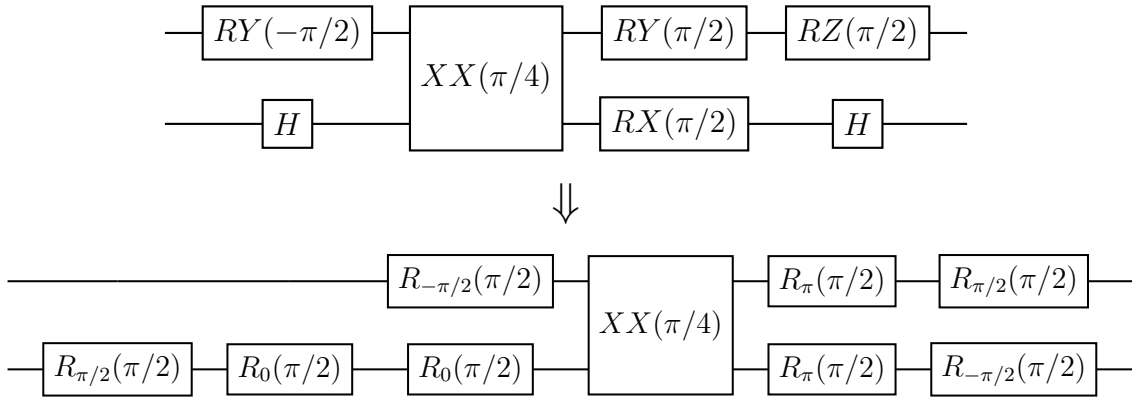

Figure 3.2: An example of how the single-qubit pass would decompose the single-qubit gates in the circuit shown in Figure 3.1

3.2 Two-Qubit Gate Compiler Pass

The first decomposition pass decomposes two-qubit gates into a combination of the native two-qubit operation $XX(\pi/4)$ and logical single-qubit gates. When the first pass encounters a two-qubit gate in the XACC IR gateset other than a CNOT, such as a CZ or SWAP, the first pass decomposes it into CNOTs and single-qubit gates. Then, when the pass encounters a CNOT, including one it introduced, the pass decomposes the CNOT as shown in Figure 2.1, leaving $XX(\pi/4)$ native operations as the only two-qubit gates in the IR. Figure 3.1 shows an example of how the first decomposition pass transforms the IR.

3.3 Single-Qubit Gate Compiler Pass

The second decomposition pass finds sequences of adjacent single-qubit gates operating on the same qubit, calculates the product G of them, and uses a numerical optimizer to find the rotation angles for $R_\phi(\pi/2)$ native operations to approximate G up to a user-configurable

[1]https://github.com/ausbin/xacc/tree/ion-trap-backend

tolerance (default 10^{-4}). This is similar to the existing compiler [6], but the new compiler has some additional optimizations. Figure 3.2 shows an example of how the single-qubit pass transforms the IR.

The new compiler uses an L-BFGS optimizer to minimize an objective function that measures the "distance" between a sequence of rotations and the 2×2 goal unitary, starting with one rotation and adding rotations until the final objective function value is satisfactorily small. This approach, particularly the objective function definition, draws from how [20] decomposes 4×4 unitaries into native two-qubit gates.

With a 2×2 goal unitary G and rotations $\vec{\phi} = (\phi_1, \phi_2, \ldots, \phi_n)$, the optimization function to minimize is shown in Equation 3.3. Like the existing compiler [6], empirical results gathered have always shown $n \leq 4$; future work should prove that some gates require four $R_\phi(\pi/2)$ rotations, since a Z-Y decomposition for example requires a maximum of only three rotations [2]. The absolute value in Equation 3.2 is squared to simplify finding the gradient $\nabla_{\vec{\phi}} f$ from Equation 3.3 using the product rule. The gradient for $n \in \{1, 2, 3, 4\}$ was pre-computed using Mathematica and converted into parameterized C++ code.

$$A_{\text{ex}}(\vec{\phi}) = R_{\phi_n}(\pi/2) \cdots R_{\phi_2}(\pi/2) R_{\phi_1}(\pi/2) \tag{3.1}$$

$$f(\vec{\phi}) = 4 - |\operatorname{Tr}(G^\dagger A(\vec{\phi}))|^2 \tag{3.2}$$

$$= 4 - \operatorname{Tr}(G^\dagger A(\vec{\phi}))\operatorname{Tr}(G^\dagger A(\vec{\phi}))^* \tag{3.3}$$

Choosing $A_{\text{ex}}(\vec{\phi})$ as the actual decomposition $A(\vec{\phi})$ allows the aforementioned strategy to produce an approximately "exact" decomposition. Further optimizations were found by changing $A(\vec{\phi})$ or skipping decomposition entirely depending on the situation. The following sections describe situation-specific optimizations:

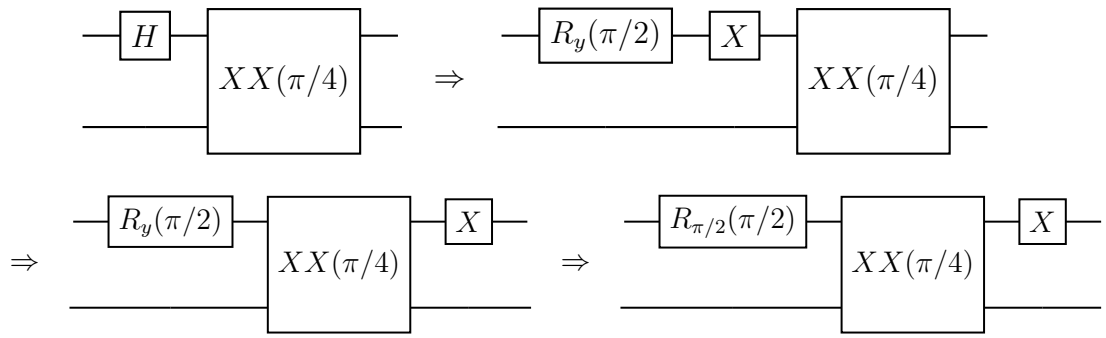

Figure 3.3: An example of how X-rotations can be commuted around XX gates

3.3.1 Decomposition up to an X Rotation

It is easy to show that $RX(\theta)$ commutes with XX, but $RY(\theta)$ and $RZ(\theta)$ only commute with XX if $\theta \equiv 0 \bmod 2\pi$ [8]. (In this thesis, it is assumed that RX, RY, and RZ are defined according to Nielsen and Chuang [2].) Thus, if an XX gate follows the sequence of single-qubit gates, G is decomposed up to $RX(\theta)$ for some θ, and then the $RX(\theta)$ is commuted to the other side of the XX gate, to be decomposed later. Figure 3.3 shows an example of this, with the X gate shown in the last step left for later decomposition. This is achieved with the optimizer by choosing $A_x(\vec{\phi})$ as $A(\vec{\phi})$ per the definition shown in Equation 3.4. Note this requires a separate pre-computed gradient to pass to the optimizer.

$$A_x(\vec{\phi}) = RX(\phi_n)R_{\phi_{n-1}}(\pi/2)\cdots R_{\phi_2}(\pi/2)R_{\phi_1}(\pi/2) \tag{3.4}$$

In some rare cases (0.05% of cases tested), the optimizer fails to find a decomposition up to an $RX(\theta)$ such that $f(\vec{\phi})$ is less than the configured threshold, possibly due to floating point rounding errors. In such cases, the new compiler falls back to the exact decomposition $A_{\text{ex}}(\vec{\phi})$ shown in Equation 3.1 to avoid harming fidelity.

3.3.2 Decomposition up to a Z Rotation

Relative phase shifts, i.e. Z rotations, do not affect measurements when measuring in the standard computational basis of Z-eigenstates. So if a measurement follows the sequence

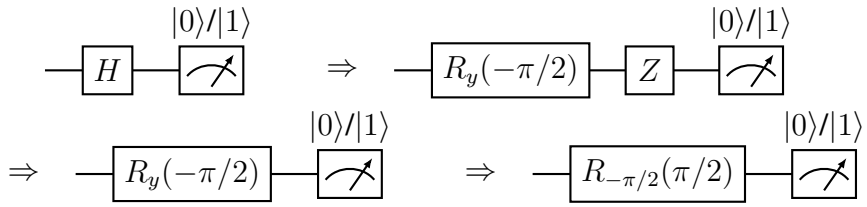

Figure 3.4: An example of how Z-rotations can be elided before measurement

of single-qubit gates, G is decomposed up to $RZ(\theta)$ for some θ and then discard the $RZ(\theta)$ gate entirely. Figure 3.4 shows an example of how this can reduce native operation count. Similar to the previous optimization, the numerical optimizer finds θ for us after $A(\vec{\phi})$ is set to $A_z(\vec{\phi})$ per the definition below in Equation 3.5. Note this again requires a separate pre-computed gradient to pass to the optimizer.

$$A_z(\vec{\phi}) = RZ(\phi_n)R_{\phi_{n-1}}(\pi/2)\cdots R_{\phi_2}(\pi/2)R_{\phi_1}(\pi/2) \tag{3.5}$$

The existing compiler has the ability to perform this optimization, which is enabled in evaluation comparing it with the new compiler.

3.3.3 Decomposition starting with a Z Rotation

Immediately after state preparation, a qubit is in state $|0\rangle$, and $RZ(\theta)\,|0\rangle = |0\rangle$ up to a global phase regardless of angle θ. Consequently, an initial Z rotation is ignored when decomposing the first single-qubit gate sequence for any qubit not preceded by a two-qubit

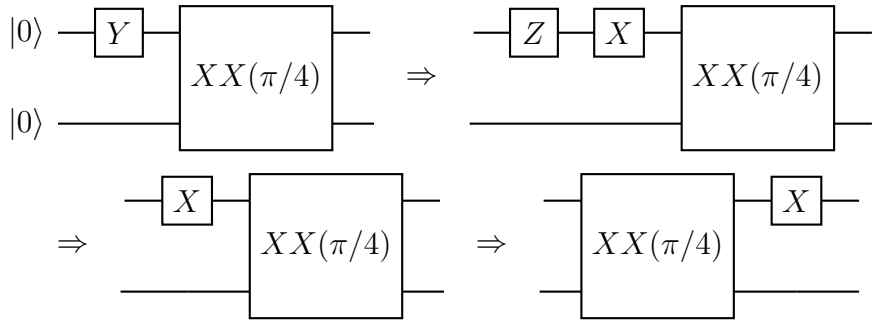

Figure 3.5: An example of how Z-rotations can be elided after state preparation

gate acting on that qubit. This optimization can be combined with the previous two optimizations as well as generation of an exact decomposition. Figure 3.5 shows an example of removing a initial Z-rotation and still commuting X-rotations around XX gates. To find these decompositions, the new compiler asks the numerical optimizer for from-Z-to-exact, from-Z-up-to-X, and from-Z-up-to-Z decompositions using the following respective definitions for $A(\vec{\phi})$:

$$A_{z,\text{ex}}(\vec{\phi}) = R_{\phi_n}(\pi/2)\cdots R_{\phi_2}(\pi/2)RZ(\phi_1) \tag{3.6}$$

$$A_{z,x}(\vec{\phi}) = RX(\phi_n)R_{\phi_{n-1}}(\pi/2)\cdots R_{\phi_2}(\pi/2)RZ(\phi_1) \tag{3.7}$$

$$A_{z,z}(\vec{\phi}) = RZ(\phi_n)R_{\phi_{n-1}}(\pi/2)\cdots R_{\phi_2}(\pi/2)RZ(\phi_1) \tag{3.8}$$

Similar to the previous optimization, the $RZ(\phi_1)$ gate of the decomposition is discarded. Note that Equation 3.6 through Equation 3.8 each require a new, separate pre-computed gradient for $f(\vec{\phi})$.

The existing compiler supports the from-Z-to-exact case [6], producing a decomposition equivalent to Equation 3.6. This optimization together with the previous exact-up-to-Z optimization is enabled in the evaluation comparing the new compiler with the existing compiler; henceforth, this combination is called "RZ optimizations."

3.3.4 Ignoring Identity

If setting $A(\vec{\phi})$ equal to the identity matrix I and invoking $f(\vec{\phi})$ reveals that G is closer than the configured tolerance to I, the new compiler skips generating any rotations at all. Figure 3.6 shows an example of how the IR would change in this situation.

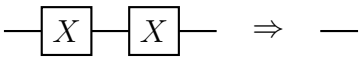

Figure 3.6: An example of removing gates whose product is equivalent to the identity gate

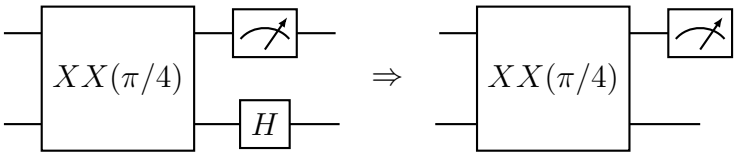

Figure 3.7: An example of removing gates whose qubits are not measured

3.3.5 Discarding Trailing Gates

If the sequence of gates immediately precedes the end of the circuit, without an explicit measurement by the programmer, the new compiler discards all the gates entirely, as they will not affect measurement outcomes. Figure 3.7 shows an example of how the IR would be transformed in this case.

3.4 Future Hardware Upgrades

In anticipation of future hardware upgrades, namely a tightly-focused beam for each individual ion, rudimentary support for all-to-all qubit connectivity and parallel single-qubit operations are implemented in the new compiler, either of which can be enabled by configuration flags.

QCOR itself handles qubit placement, so full connectivity is supported by simply having the new backend pass a fully connected coupling map to QCOR instead of a coupling map indicating a linear chain. Full connectivity reduces the number of SWAP gates needed to execute a logical circuit on a linear chain of qubits, in turn reducing the number of CNOTs inserted by QCOR to perform SWAPs, and ultimately reducing the number of

Table 3.1: Native Operations Produced for the Bell State Circuit `H 0; CNOT 0,1` for Serial Single-Qubit Gates

Operation	Target Ion	ϕ
$XX(\pi/4)$	0,1	
$R\phi(\pi/2)$	0	$\pi/2$
$R\phi(\pi/2)$	1	0

Table 3.2: Native Operations Produced for the Bell State Circuit `H 0; CNOT 0,1` for Parallel Single-Qubit Gates

Operation	Target Ion 1	ϕ_1	Target Ion 2	ϕ_2
$XX(\pi/4)$	0,1			
$R\phi(\pi/2)$	0	$\pi/2$	1	0

$XX(\pi/4)$ and $R_\phi(\pi/2)$ gates which together effect the swapping CNOTs.

Rudimentary support for parallel single-qubit is implemented using a naïve algorithm that greedily constructs a table from the XACC IR produced by the decomposition passes, with multiple $R_\phi(\pi/2)$ per row, and with each $XX(\pi/4)$ having its own row. Parallel single-qubit operations would not reduce the number of $XX(\pi/4)$ gates, but such a hardware upgrade could allow multiple $R_\phi(\pi/2)$ gates to occur across different qubits at once. Table 3.1 and Table 3.2 show examples of serial and parallel native operations, respectively, for the same input program. Clearly, one cannot compare runtime between serial and parallel configurations by counting the total number of gates; for example, Table 3.1 and Table 3.2 have the same number of $R_\phi(\pi/2)$ operations, but Table 3.2 offers a shorter runtime. Thus, this thesis estimates the length of the critical path by counting the number of rows in the table of native operations sent to the control software, henceforth calling each row a "cycle" whenever it is necessary to distinguish from native gate counts.

Parallel two-qubit operations stand to reduce the number of cycles spent on $XX(\pi/4)$ gates [14]. However, given parallel two-qubit gates cannot be implemented on the three-qubit benchmarks used for evaluation or the testbed in its original three-qubit configuration [6, 17], investigating parallel two-qubit gates on the testbed is left as future work.

CHAPTER 4
EVALUATION

4.1 System Configuration

The targeted physical ion trap testbed and its control scheme have been modified for domain-specific computations based on global operations [21], so the benchmark circuits cannot be executed on the physical testbed itself. However, correctness of generated code can be verified by executing the native operations generated in the simulator already included in the control software, originally used to aid in calibration. Rather than performing noisy simulations, the evaluation in this chapter instead estimates the performance of compilation by simply counting the number of primitive operations; Section 4.5 argues why this is reasonable.

4.2 Optimizations Tested

The new compiler has the capability to perform the following three types of optimizations which were used when compiling benchmarks:

1. High level optimizations already included in QCOR [18], which includes approaches for simplifying large circuits [22]

2. Single-qubit optimizations explained in Section 3.3

3. Optimizations for future hardware mentioned in Section 3.4

Given #1 is designed for larger, more complex circuits on a higher number of qubits, it was observed that #1 made no difference in final native gate counts for the benchmarks used. Consequently, the remainder of this chapter focuses on the impact of #2 and #3. However, one should note that less trivial quantum algorithms run on the testbed in the future could

benefit significantly from the high-level QCOR optimizations, with Nguyen et al. seeing an average 23.2% reduction in gates on benchmarks ranging from 5 to 96 qubits [18, 23].

4.3 Benchmarks Chosen

To evaluate the new backend, the following set of benchmark QCOR programs were compiled for three qubits:

- GHZ (Greenberger-Horne-Zeilinger), which generates the state $\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$
- Bernstein-Vazirani with secret string $s = 11$, similar to Figure 2.2
- Grover with one iteration and marked states $|101\rangle$ and $|110\rangle$ [24]
- Quantum Fourier Transform using the `qft()` routine included in QCOR
- VQE (Variational Quantum Eigensolver) on the three-qubit Hamiltonian from [25] using the QCOR tooling for VQE

Each benchmark was run with the following three configurations of compiler and simulator:

1. Against an existing XACC backend which runs the instructions from XACC IR directly on the local Quantum++ simulator [26]
2. Against the new XACC backend configured to generate assembly, which the existing compiler compiles to a sequence of primitive operations for the control software to simulate
3. Against the new XACC backend configured to generate a sequence of primitive operations on its own as described in Section 3.2 through Section 3.4, also simulated by the control software

To validate the results of the new backend, the probability distribution of measurements calculated from the final state vectors is compared between compiler configurations #1, #2, and #3 above. Next, to evaluate the optimizations for current hardware, the number of native operations produced by the existing compiler in #2 is compared with the compiler in #3. Finally, the optimizations for possible future hardware are evaluated by comparing the native gate count produced by #2 with the native gate count produced by #3 with future hardware optimizations enabled.

4.4 Validation Results

To validate the results of simulating the benchmarks, the probability distribution of measurements from the final state vector produced by the Quantum++ simulator was compared with the distribution of measurements for the control system simulation of the sequence of primitive operations generated by both the existing compiler and new compiler. The VQE benchmark represents a special case in that instead of simply running a quantum kernel, the ansatz quantum kernel acts as part of the objective function for an optimizer on the variational parameters. As a result, in addition to comparing the probability of different measurements of the first VQE iteration (subsequent iterations diverged in parameters), it was verified that the VQE benchmark converged to a result approximately equal to the value found by Dumitrescu et al. [25].

The resulting states from the new compiler often did not match those produced by either Quantum++ or the control software simulation of the existing compiler results, even up to a global phase, but this is intended behavior of the single-qubit pass described in Section 3.3. First, the optimization in subsection 3.3.2 discards trailing Z-rotations, introducing relative phase differences in the final state. Second, the discarding of trailing gates mentioned in subsection 3.3.5 introduces some differences in final state compared to the others; for example, with Bernstein-Vazirani, the new compiler produces final state $\frac{1}{\sqrt{2}}|110\rangle - \frac{1}{\sqrt{2}}|111\rangle$ rather than $|111\rangle$ owing to an elided final Hadamard gate on the ancilla qubit, which the

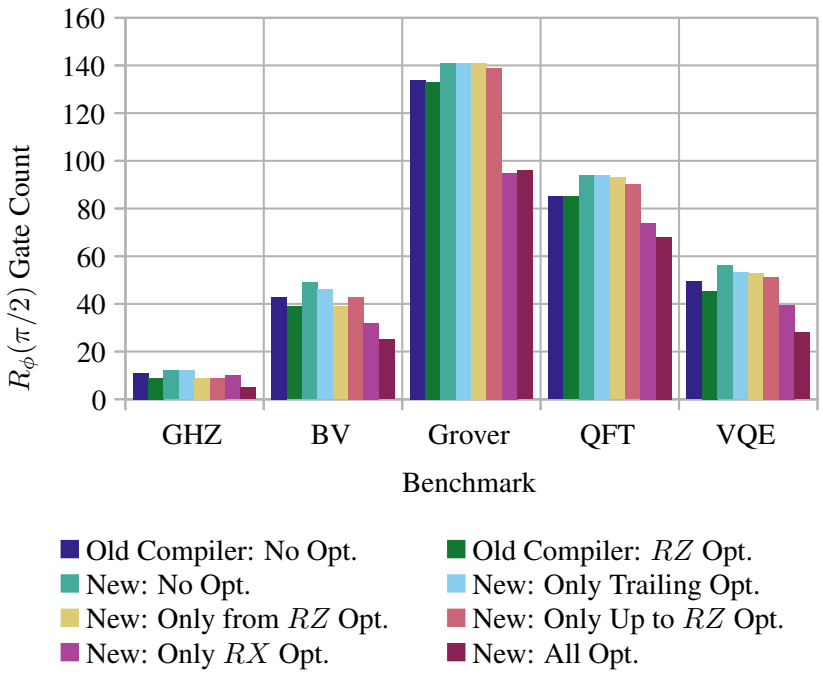

Figure 4.1: Comparison of $R_\phi(\pi/2)$ operation counts generated across the benchmarks by the existing compiler and by different configurations of the new compiler (lower is better)

benchmark program does not explicitly `Measure`[1]. In all cases, the differences in final quantum state do not affect the probability distribution of measurements.

4.5 Gate Count Comparison

As expected, the two-qubit pass described in Section 3.2 eliminates no $XX(\pi/4)$ gates, leaving the new count of $XX(\pi/4)$ gates identical to the previous compiler. However, the one-qubit pass in Section 3.3 achieves an average $1.54\times$ reduction in $R_\phi(\pi/2)$ operations, even with the RZ optimizations enabled in the existing compiler. Figure 4.1 shows a comparison of primitive operation counts between the new compiler and the existing compiler for all benchmarks used.

To examine which optimizations contributed most to the reduction in single-qubit operations, the benchmarks were also compiled with only individual optimizations from Sec-

[1]On real hardware, one might choose to measure the ancilla to verify that it is $|1\rangle$ as an error detection measure, but the BV benchmark is left as-is to ensure the evaluation tests how the new compiler handles gates on unmeasured qubits.

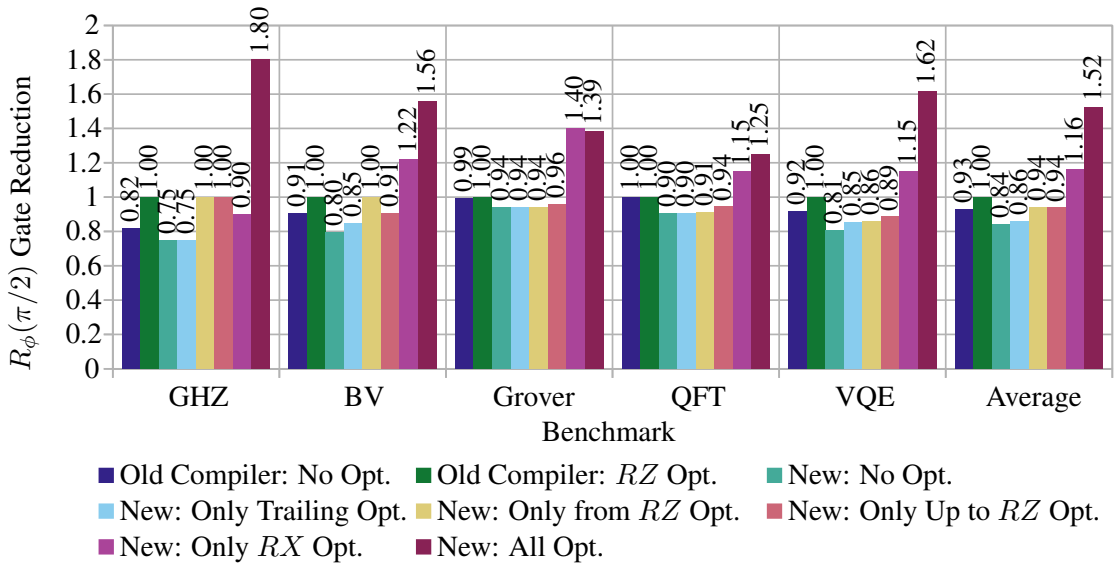

Figure 4.2: Comparison of reduction in $R_\phi(\pi/2)$ operations generated across the benchmarks by the existing compiler and by other compiler configurations (higher is better)

tion 3.3 enabled, and also none of them enabled. The $R_\phi(\pi/2)$ count reduction numbers are plotted in Figure 4.2, with reduction calculated by dividing the $R_\phi(\pi/2)$ gates the existing compiler generates with RZ optimizations enabled by the number of $R_\phi(\pi/2)$ gates generated in some other configuration. These results show that individual optimizations struggle to compete with the RZ optimizations in the existing compiler, with the exception of the new RX optimizations usually offering some individual reduction. On the GHZ benchmark, however, the RX freedom alone is not as effective, with all optimizations combined performing better. Indeed, on average, each optimization provides a modest reduction in gates, but the combination readily beats any individual optimization.

The error and duration estimates proposed by Maslov for $R_\phi(\theta)$ gates on ion trap systems are defined in terms of θ [8], not ϕ, so with θ fixed to $\pi/2$, the number of $R_\phi(\pi/2)$ gates reasonably correlates with error and duration. Consequently, it is expected that the benchmarks used and other quantum circuits with similar complexity will experience higher overall fidelity on the testbed using the new compiler. Furthermore, an $n\times$ reduction in physical $R_\phi(\pi/2)$ gates could allow $n\times$ more sequences of adjacent logical single-qubit gates with the same approximate total duration and total error as before. Thus, if combined

with future optimizations for two-qubit gates, the new compiler could allow running longer, more complicated programs on the testbed.

4.6 Impact of Hardware Upgrades

The new compiler does not reduce the number of $XX(\pi/4)$ gates on present hardware, but fully-connected qubits could reduce the $XX(\pi/4)$ gate count by $2.40\times$ on average, as seen in Figure 4.3. The Bernstein-Vazirani benchmark showed the greatest reduction at $4.00\times$, going from 8 operations to 2 operations, owing to the removal of six CNOTs used for two SWAPs. GHZ, on the other hand, showed no benefit, as it executes CNOTs only on adjacent ions. In general, then, the benefit of full connectivity is proportional to the share of of two-qubit gates between non-adjacent ions.

As shown in Figure 4.4, full connectivity can also decrease the number of $R_\phi(\pi/2)$ operations since they are used to effect a CNOT using $XX(\pi/4)$ (see Figure 2.1). With a $5.13\times$ reduction on average, a fully-connected upgrade could readily beat the average $1.52\times$ reduction seen with the new compiler on current hardware. Still, some benchmarks see no benefit, such as GHZ for the reasons previously mentioned.

Although it cannot reduce $XX(\pi/4)$ cycles, Figure 4.4 shows that hardware support for parallel $R_\phi(\pi/2)$ operations on different qubits could reduce $R_\phi(\pi/2)$ cycles by an average of $2.40\times$ over the existing compiler and hardware regime, compared to $1.52\times$ with the new compiler on present hardware. While all benchmarks saw at least some parallelism, future work could likely increase this reduction by investigating more complex strategies for single-qubit unitary decomposition (Section 3.3) that take parallelism into account.

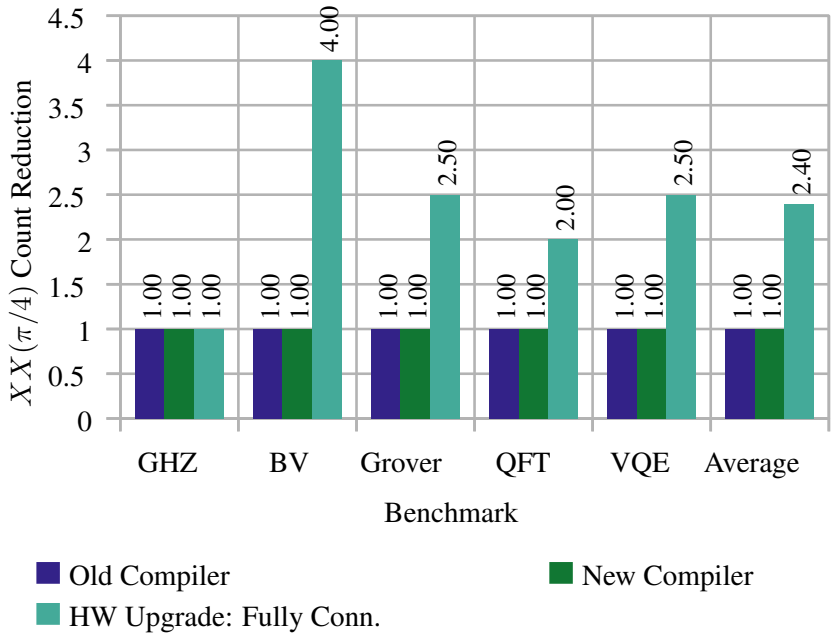

Figure 4.3: Comparison of reduction in $XX(\pi/4)$ operation counts generated across the benchmarks by the new compiler between present hardware and upgraded hardware with all-to-all qubit connectivity (higher is better)

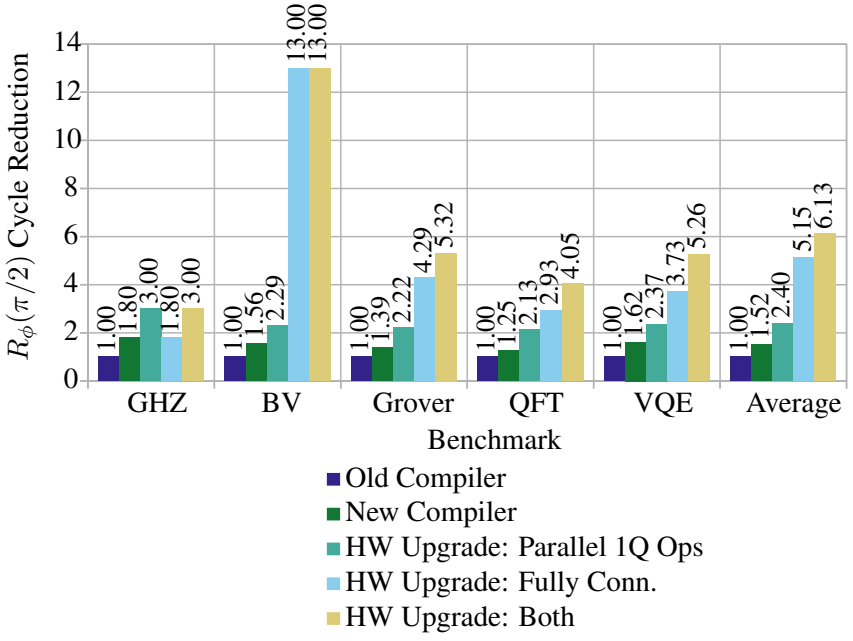

Figure 4.4: Comparison of reduction in $R_\phi(\pi/2)$ operations generated across the benchmarks by the new compiler between present hardware and different hardware upgrades (higher is better)

CHAPTER 5
ADAPTATION TO OTHER HARDWARE

While the new compiler outlined in this thesis was designed for a particular ion trap machine, the backend developed can be adapted for any hardware with a similar gateset. For example, some IonQ hardware and the QSCOUT ion trap quantum testbed operated by Sandia National Laboratories natively support $R_\phi(\theta)$ and $XX(\alpha)$ gates (Equation 2.2 and Equation 2.7 in Section 2.1) [27, 11, 28]. On these machines, the angle θ in $R_\phi(\theta)$ is not limited to $\pi/2$ as in the GTRI testbed, so the numerical optimizer approach explained in Section 3.3 is not necessary, as there is a closed form solution for decomposing arbitrary single-qubit unitaries into two $R_\phi(\theta)$ gates [8]. However, the approach of removing unnecessary RZ gates and commuting RX gates described in the remainder of Section 3.3 would likely still reduce gate count.

Still, it is a common choice to restrict θ in $R_\phi(\theta)$ to limit the set of calibrations. For example, a recent ion trap quantum computer developed by Honeywell supports $R_\phi(\theta)$ with θ constrained to π or $\pi/2$, much like the $\theta = \pi/2$ restriction on the GTRI testbed [16]. The numerical optimization strategy described in Section 3.3 could be adapted by running the up-to-Z optimizer on all seven combinations of zero through two $R_\phi(\theta)$ rotations with each individual angle $\theta \in \{\pi, \pi/2\}$, all followed by a Z-rotation. The machine has the entangling gate $ZZ(\alpha) = \exp(-i\alpha(Z \otimes Z))$, analogous to the $XX(\alpha)$ gate. $ZZ(\alpha)$ commutes with $RZ(\theta)$, so the approach for commuting $RX(\theta)$ around $XX(\alpha)$ gates could be adapted to commute $RZ(\theta)$ gates around $ZZ(\alpha)$ gates instead. Despite these similarities, a compiler for the Honeywell machine would need to understand the multiple transport operations supported; the new backend does not consider transport operations, since the testbed control software infers them automatically.

The current domain-specific configuration of the GTRI testbed could also be supported,

as well as other hardware with global operations in general, but this would require adding new instructions to XACC IR targeting all qubits. Moreover, the IR transformations described in Section 3.2 and Section 3.3 will not be useful on hardware with only global operations, since one would not typically program them using a typical quantum circuit. However, backends for any hardware whose native operations support typical quantum circuits benefit from the existing high-level circuit optimizations already present in QCOR [18].

CHAPTER 6

RELATED WORK

Martinez et al. discuss techniques for compiling arbitrary multi-qubit gates to ion-trap architectures supporting collective rotations of the whole qubit register about any axis ($C(\theta, \phi)$ gate), single qubit rotations around the Z axis ($Z_n(\theta)$ gate), and MS gates [29]. They show how any multi-qubit gate can be decomposed into a sequence of single-qubit local unitaries and MS gates. The new compiler employs this decomposition concretely in the XACC backend restricted to the single-qubit and two-qubit case. Maslov proposes a generic architecture for optimizing compilers targeting ion-trap quantum computers [8]. QCOR does not match the architecture exactly, but instead provides a framework for describing and applying relevant optimizations. The implementation of the new compiler does not support the peep-hole optimizations proposed in the paper.

QFAST innovates in circuit synthesis by presenting a novel representation of circuits allowing them to use numerical optimization to replace expensive searches over circuit structures [30]. They use a bottom-up approach which stems from a special encoding that enables them to find better building blocks for circuit synthesis. QGo is a quantum circuit optimization framework that aims to be scalable (optimizing circuits containing 60+ qubits in a reasonable amount of time) [31]. It is able to achieve this performance by employing a partitioning scheme in which the circuit is broken down into smaller components, independently optimized, and then composed together.

Lu et al. discuss a scalable scheme for implementing a global multi-qubit entangling gate which can potentially lead to exponential speedups as compared to a circuit decomposition involving single- and two-qubit entangling gates [32]. The GTRI testbed is currently configured for specific multi-qubit global entangling operations [21], but they are not considered them in this thesis.

Gokhale et al. have implemented a compiler that exploits pulse-level optimizations without resorting to Quantum Optimal Control (QOC) approaches, thereby bypassing the experimental barrier of measuring and maintaining the Hamiltonian of a quantum system [26]. They are able to achieve about $1.6\times$ error reductions and $2\times$ speedups on near-time quantum algorithms run using the OpenPulse interface on IBM's quantum computers. The new compiler does not currently employ these optimizations, which are left as future work.

Lobser et al. propose JaqalPaw, a pulse-level interface for programming an ion trap quantum computer [33]. JaqalPaw is an extension for Jaqal, a quantum assembly language created for the QSCOUT ion trap quantum testbed at Sandia National Laboratories [27]. The pulse-level interface provided by JaqalPaw is lower-level than the gate-level interface assumed by this thesis, but to enable deeper optimizations, the backend described in this thesis could be extended to lower the gates produced to a pulse schedule in JaqalPaw or a similar format.

Pino et al. demonstrate a quantum computer also based on an ion trap, but contrary to the assumptions in this thesis about future GTRI testbed upgrades (i.e., per-ion beams), their design transports ions through shared beams to perform gates [16]. Ion swapping operations help provide full qubit connectivity, and multiple beams provide support for parallel operations. The authors briefly mention a compiler that performs qubit mapping such that it minimizes the number of native transport operations, which may include linear ion movement, swapping ion order, and splitting or combining ion crystals, but they do not go into detail on the compiler design. Since the assumptions in this thesis about future hardware upgrades to the GTRI testbed are only guesses, future work should consider minimizing the number of these transport operations in addition to native gate count, the primary consideration in this thesis.

The TILT (Trapped-Ion Linear Tape) architecture for ion trap hardware proposed by Wu et al. consists of a stationary set of lasers targeting a subset of the ions in a single ion chain [34]. Compared to the previously proposed QCCD (Quantum Charge-Coupled

Device) architecture [35], TILT offers simpler hardware and avoids expensive shuttling operations. The authors detail LinQ, their compiler framework designed for TILT hardware, which employs two heuristic-based algorithms: one for inserting SWAP gates, and another for transport operations. The first algorithm facilitates two-qubit gates between qubits not within the "execution zone" of the lasers, and the second attempts to avoid unnecessary tape movement, which may introduce qubit noise. Their discussion of LinQ further emphasizes the importance of future work on the new backend minimizing the number of transport operations. Additionally, it is not mentioned if their compiler employs the optimizations employed in Section 3.3. Future work should investigate how to apply these optimizations to TILT systems.

CHAPTER 7
CONCLUSION

This thesis details efforts to add a new ion trap backend to the XACC quantum toolchain as well as a demonstration of multi-level optimization strategies to provide algorithmic optimizations at the language, IR, and backend levels. As a demonstration of this strategy, the implementation of the new backend allows heterogeneous quantum–C++ programs to be compiled and optimized to use fewer physical operations, along with a basic simulation functionality using the testbed's existing development tools.

Future work in this space should look to extend this programming environment to support further optimizations as well as to test with the quantum hardware instead of the simulated environment. It will be important to consider optimizations such as parallel two-qubit gates, non-$R_\phi(\pi/2)$ operations, and multi-qubit optimizations to improve the fidelity of generated quantum circuits. Future improved versions of the proposed compiler backend could target more complex architectures such as TILT and QCCD [34, 35], as well as produce pulses encoded in a format such as JaqalPaw [33].

REFERENCES

[1] J. S. Young, J. Riedy, T. M. Conte, V. Sarkar, P. Chatarasi, and S. Srikanth, "Experimental insights from the rogues gallery," in *2019 IEEE International Conference on Rebooting Computing (ICRC)*, Nov. 2019, pp. 1–8.

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 1st edition. Cambridge ; New York: Cambridge University Press, Dec. 9, 2010, 702 pp., ISBN: 978-1-107-00217-3.

[3] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 6, 2018.

[4] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware," *Nature*, vol. 549, no. 7671, pp. 180–187, Sep. 2017.

[5] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble, "XACC: A system-level software infrastructure for heterogeneous quantum–classical computing," *Quantum Science and Technology*, vol. 5, no. 2, p. 024 002, Feb. 2020.

[6] C. D. Herold *et al.*, "Universal control of ion qubits in a scalable microfabricated planar trap," *New Journal of Physics*, vol. 18, no. 2, p. 023 048, Feb. 2016.

[7] T. M. Mintz, A. J. McCaskey, E. F. Dumitrescu, S. V. Moore, S. Powers, and P. Lougovski, "QCOR: A language extension specification for the heterogeneous quantum-classical model of computation," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 16, no. 2, 22:1–22:17, Mar. 18, 2020.

[8] D. Maslov, "Basic circuit compilation techniques for an ion-trap quantum machine," *New Journal of Physics*, vol. 19, no. 2, p. 023 035, Feb. 2017.

[9] K. Mølmer and A. Sørensen, "Multiparticle entanglement of hot trapped ions," *Physical Review Letters*, vol. 82, no. 9, pp. 1835–1838, Mar. 1, 1999.

[10] A. Sørensen and K. Mølmer, "Quantum computation with ions in thermal motion," *Physical Review Letters*, vol. 82, no. 9, pp. 1971–1974, Mar. 1, 1999.

[11] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, "Demonstration of a small programmable quantum computer with atomic qubits," *Nature*, vol. 536, no. 7614, pp. 63–66, Aug. 2016.

[12] J.-L. Brylinski and R. Brylinski, "Universal quantum gates," *arXiv:quant-ph/0108062*, Aug. 13, 2001. arXiv: quant-ph/0108062.

[13] M. J. Bremner *et al.*, "Practical scheme for quantum computation with any two-qubit entangling gate," *Physical Review Letters*, vol. 89, no. 24, p. 247 902, Nov. 25, 2002.

[14] C. Figgatt *et al.*, "Parallel entangling operations on a universal ion-trap quantum computer," *Nature*, vol. 572, no. 7769, pp. 368–372, Aug. 2019.

[15] L. E. de Clercq *et al.*, "Parallel transport quantum logic gates with trapped ions," *Physical Review Letters*, vol. 116, no. 8, p. 080 502, Feb. 24, 2016.

[16] J. M. Pino *et al.*, "Demonstration of the trapped-ion quantum CCD computer architecture," *Nature*, vol. 592, no. 7853, pp. 209–213, Apr. 2021.

[17] S. D. Fallek, C. D. Herold, B. J. McMahon, K. M. Maller, K. R. Brown, and J. M. Amini, "Transport implementation of the bernstein–vazirani algorithm with ion qubits," *New Journal of Physics*, vol. 18, no. 8, p. 083 030, Aug. 2016.

[18] T. Nguyen, A. Santana, T. Kharazi, D. Claudino, H. Finkel, and A. McCaskey, "Extending C++ for heterogeneous quantum-classical computing," *arXiv:2010.03935 [quant-ph]*, Oct. 8, 2020. arXiv: 2010.03935.

[19] Oak Ridge National Laboratory, *QCOR Website*, https://qcor.ornl.gov/.

[20] L. Lao, P. Murali, M. Martonosi, and D. Browne, "Designing Calibration and Expressivity-Efficient Instruction Sets for Quantum Computing," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2021, pp. 846–859.

[21] J. Rajakumar, J. Moondra, S. Gupta, and C. D. Herold, "Generating target graph couplings for QAOA from native quantum hardware couplings," *arXiv:2011.08165 [physics, physics:quant-ph]*, Nov. 16, 2020. arXiv: 2011.08165.

[22] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Information*, vol. 4, no. 1, pp. 1–12, May 10, 2018.

[23] M. Amy and V. Gheorghiu, "Staq—a full-stack quantum processing toolkit," *Quantum Science and Technology*, vol. 5, no. 3, p. 034 016, Jun. 2020.

[24] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath, and C. Monroe, "Complete 3-qubit grover search on a programmable quantum computer," *Nature Communications*, vol. 8, no. 1, p. 1918, Dec. 4, 2017.

[25] E. F. Dumitrescu *et al.*, "Cloud quantum computing of an atomic nucleus," *Physical Review Letters*, vol. 120, no. 21, p. 210 501, May 23, 2018.

[26] V. Gheorghiu, "Quantum++: A modern C++ quantum computing library," *PLOS ONE*, vol. 13, no. 12, e0208073, Dec. 10, 2018.

[27] S. M. Clark *et al.*, "Engineering the Quantum Scientific Computing Open User Testbed," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–32, 2021.

[28] B. C. A. Morrison *et al.*, "Just another quantum assembly language (Jaqal)," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Oct. 2020, pp. 402–408.

[29] E. A. Martinez, T. Monz, D. Nigg, P. Schindler, and R. Blatt, "Compiling quantum algorithms for architectures with multi-qubit gates," *New Journal of Physics*, vol. 18, no. 6, p. 063 029, Jun. 24, 2016.

[30] E. Younis, K. Sen, K. Yelick, and C. Iancu, "QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis," *arXiv:2103.07093 [quant-ph]*, Mar. 12, 2021. arXiv: 2103.07093.

[31] X.-C. Wu, M. G. Davis, F. T. Chong, and C. Iancu, "QGo: Scalable quantum circuit optimization using automated synthesis," *arXiv:2012.09835 [quant-ph]*, Jul. 25, 2021. arXiv: 2012.09835.

[32] Y. Lu *et al.*, "Global entangling gates on arbitrary ion qubits," *Nature*, vol. 572, no. 7769, pp. 363–367, Aug. 2019.

[33] D. Lobser *et al.*, *JaqalPaw: A guide to defining pulses and waveforms for Jaqal*, https://www.sandia.gov/quantum/Projects/Uploads/JaqalPaw__A_Guide_to_Defining_Pulses_and_Waveforms_for_Jaqal.pdf, 2021.

[34] X.-C. Wu *et al.*, "TILT: Achieving higher fidelity on a trapped-ion linear-tape quantum computing architecture," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 153–166.

[35] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, no. 6890, pp. 709–711, Jun. 2002.