Enabling a Programming Environment for an Experimental Ion Trap Quantum Testbed

Austin Adams, Elton Pinto, Jeffrey Young, Creston Herold, Alex McCaskey, Eugene Dumitrescu, Thomas M. Conte

ICRC 2021
November 30th, 2021

Georgia Tech

Introduction

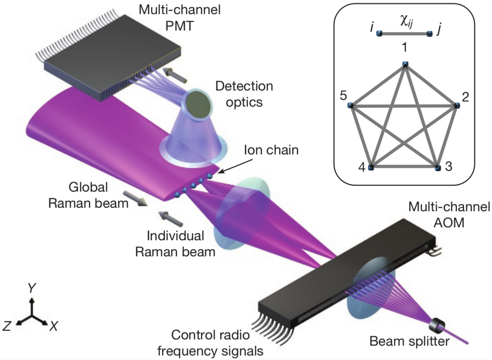
- Idea: connect an existing quantum compiler framework to the Georgia Tech Research Institute (GTRI) quantum testbed
- Motivation: Current approach is hardware expert–oriented and requires programming in assembly. Our backend introduces a more programmer-driven flow for programmers who may not be hardware experts
- Our contributions:
    - New compiler backend that interacts with the low-level testbed control software
    - Show multi-level optimizations: hardware-agnostic level and hardware-specific level
    - Performance evaluation of our backend
    - Investigation of the impact of future hardware upgrades

Background

Ion Trap Quantum Computers

- Ion trap quantum computers realize qubits by manipulating spin of trapped ions using electromagnetic radiation (e.g., lasers, microwaves)
- We consider two popular native gates on ion trap systems:
    - The single-qubit gate $R_\phi(\theta)$: A rotation of θ around the angle ϕ in the X-Y plane of the Bloch sphere
    - The two-qubit entangling gate $XX(\alpha)$: can perform a CNOT when combined with a few single-qubit gates

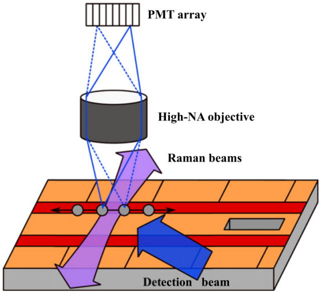
An early IonQ machine [1]:



GTRI Quantum Testbed

- The CIPHER Quantum Systems Division at Georgia Tech Research Institute (GTRI) has a quantum testbed based on an ion trap
  - Original 2016 configuration [3] (right) could only target two ions (qubits) simultaneously, but this has been upgraded to allow single-qubit addressing
  - Native operations: $R_\phi(\pi/2)$ and $XX(\pi/4)$
    - Rudimentary compiler exists for decomposing quantum assembly into a sequence of these operations
    - Control software also contains an ideal simulator originally used in calibration
  - Currently repurposed for domain-specific computations based on global operations [7]

GTRI testbed apparatus as of 2016 [3]:



QCOR

- QCOR: specification for compiler framework intended for heterogeneous quantum–classical algorithms on near-term hardware
- QCOR implementation:
  - Uses Clang syntax handler to allow running quantum circuits inline in C++ code (right)
  - Behind the scenes, uses the lower-level XACC compiler framework
    - We add a GTRI compiler backend to XACC, which surfaces it on the QCOR level

QCOR C++ program which generates and measures the GHZ state $\frac{1}{\sqrt{2}}|00\cdots0\rangle + \frac{1}{\sqrt{2}}|11\cdots1\rangle$:

```
__qpu__ void ghz(qreg q) {
  H(q[0]);
  for (int i = 1; i < q.size(); i++)
    CNOT(q[i-1], q[i]);
  Measure(q);
}

int main(int argc, char **argv) {
  auto q = qalloc(atoi(argv[1]));
  ghz(q);
  q.print();
}
```

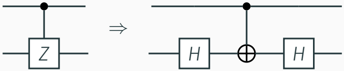
Compiler Backend Design

Backend Implementation

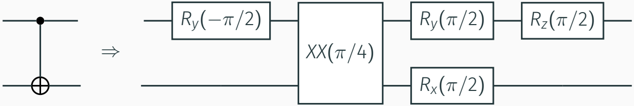
- Our XACC backend for the GTRI testbed takes XACC IR as input and:
    1. Runs an IR transformation for two-qubit gates
    2. Runs another IR transformation for single-qubit gates
    3. Writes a sequence (or table) of primitive operations to a file in a directory polled by the control software
    4. Parses simulation result written by control software and returns measurements

Two-Qubit Gate Compiler Pass

1. Decompose two-qubit gates in XACC IR into combinations of CNOT and single-qubit gates. For example,



2. Decompose CNOTs into $XX(\pi/4)$ native gates and single-qubit gates:



Single-Qubit Gate Compiler Pass

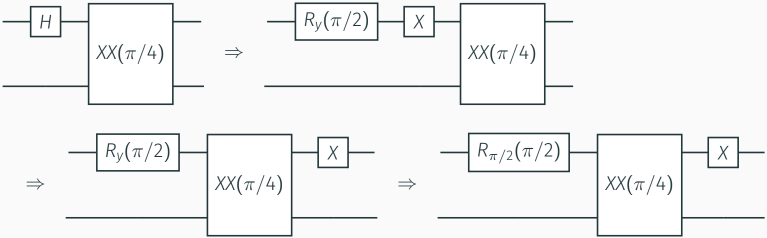
- Find adjacent single-qubit gates and multiply them together to get a goal unitary G
- Need to decompose G into the product of $R_\phi(\pi/2)$ gates
  - For example, up to a global phase, $H = XR_y(\pi/2) = R_0(\pi)R_{\pi/2}(\pi/2) = R_0(\pi/2)R_0(\pi/2)R_{\pi/2}(\pi/2)$, so

$$-\boxed{H}- \quad \Rightarrow \quad -\boxed{R_{\pi/2}(\pi/2)}-\boxed{R_0(\pi/2)}-\boxed{R_0(\pi/2)}-$$

- Use numerical optimizer to find the ϕ angles
- Start with 1 rotation and keep adding rotations until we get a sufficiently close decomposition
  - In our experiments, the maximum needed is 4 rotations

Single-Qubit Decomposition up to an X-Rotation

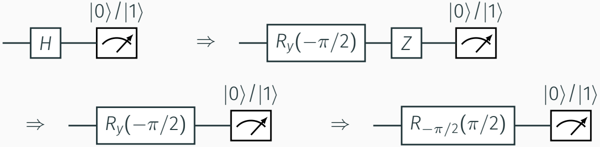
- We can ask the optimizer for a different decomposition in different situations
- Fun fact: XX commutes with X-rotations
- So when G ends at an XX gate, we can ask for a decomposition up to an X-rotation, and commute that final $R_x(\theta)$ to the other side of the XX. Example:



- Deal with the $R_x(\theta)$ in a later iteration

Single-Qubit Decomposition up to a Z-Rotation

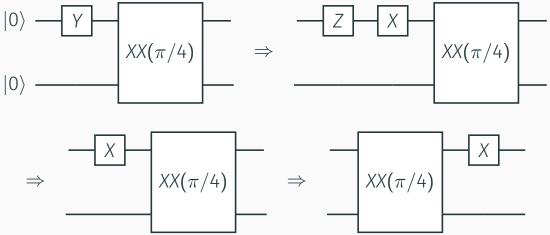
- Z-rotations do not change measurement outcomes when measuring in the $|0\rangle$ / $|1\rangle$ basis
- When the gates to decompose end at a measurement, we can ask optimizer for a decomposition up to a Z-rotation ($R_z(\theta)$ gate)
- Example using $H = ZR_y(-\pi/2)$:



- Can discard the ending $R_z(\theta)$

Single-Qubit Decomposition *from* a Z-Rotation

- Up to a global phase, $R_z(\theta)\,|0\rangle = |0\rangle$
- So when the gates start at the beginning of the circuit, we can ask the optimizer for a decomposition starting with a Z-rotation
- Example using $Y = XZ$ (up to a global phase):



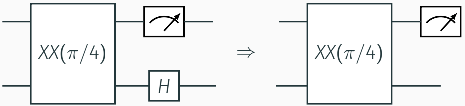
- Discard the leading $R_z(\theta)$
- Can combine with the previous two optimizations!

Single-Qubit Gate Compiler Pass: Two More Optimizations

Can skip the optimizer entirely in two situations:

1. If G is closer than the configured threshold to identity, we discard the sequence of gates. For example:

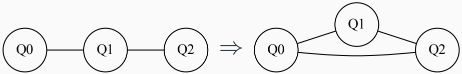
2. If the sequence of gates ends with the end of the circuit, without a measurement, then we can safely discard the gates without affecting measurement outcomes the programmer cares about:



Future Hardware Upgrades

What if the GTRI testbed had a tightly-focused beam for each ion as demonstrated by IonQ [2]? We consider the following two benefits:

1. **All-to-all connectivity:** Can reduce number of SWAP gates needed to execute logical circuit on linear chain of ions (qubits)
    - Easy: QCOR handles qubit placement, so have our `Accelerator` tell QCOR we have full connectivity instead of linear



2. **Parallel single-qubit operations:** Execute multiple $R_\phi(\pi/2)$ gates across different qubits in the same "cycle"
    - We use a greedy algorithm that takes resulting IR from two compiler passes and builds a table of native operations
    - Example for the Bell state circuit `H 0; CNOT 0,1`:

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

$\Rightarrow$

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

Experiments and Discussion

Evaluation

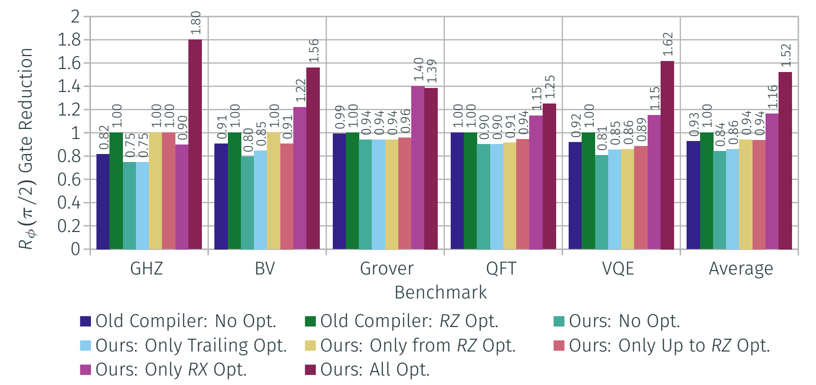
- Physical testbed hardware has been repurposed for domain-specific computations based on global operations, so we cannot test on hardware
- Instead, we:
  1. Validate results using the simulator already included in the control software
  2. Roughly estimate fidelity by counting native operations
- Benchmark QCOR programs on three-qubit programs:
  - GHZ, which generates the state $\frac{1}{\sqrt{2}} |000\rangle + \frac{1}{\sqrt{2}} |111\rangle$
  - Bernstein-Vazirani with secret string $s = 11$
  - Grover with one iteration and marked states $|101\rangle$ and $|110\rangle$
  - Quantum Fourier Transform using the `qft()` QCOR routine
  - VQE (Variational Quantum Eigensolver) on a three-qubit Hamiltonian using the QCOR tooling for VQE

Validation Results

- We compare probability distribution of measurements based on the final state vectors produced by our backend and the existing Quantum++ simulator backend.
- Why not compare final state vectors?
  - By design, the single-qubit pass will produce different final states, even considering global phase. Example: for Bernstein–Vazirani, our compiler discards a trailing Hadamard on the ancilla qubit, so the final state becomes $\frac{1}{\sqrt{2}}\left|110\right\rangle - \frac{1}{\sqrt{2}}\left|111\right\rangle$ rather than $\left|111\right\rangle$
    - The programmer measures only the other two qubits, so no observable difference

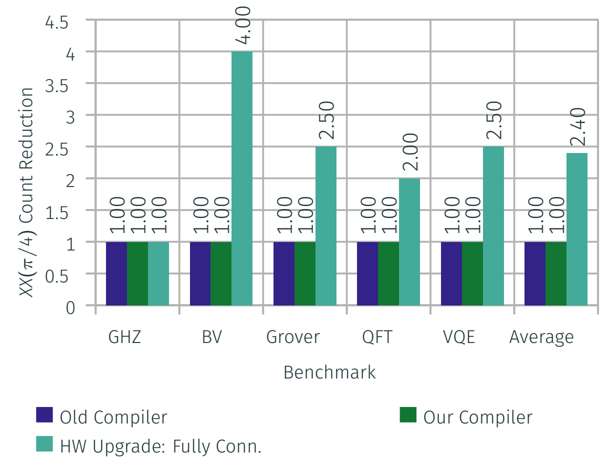
Gate Count Reduction

As expected, we saw no reduction in $XX(\pi/4)$. For $R_\phi(\pi/2)$ gates, we saw an average of 1.52× reduction:



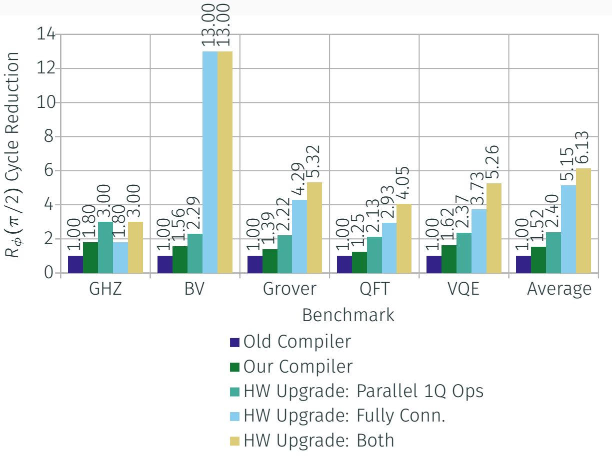
$XX(\pi/4)$ Gate Count Reduction: Hardware Upgrades

Full connectivity showed a 2.40× reduction in $XX(\pi/4)$ native operations:



$R_\phi(\pi/2)$ Gate Count Reduction: Hardware Upgrades

Together, full connectivity and parallel operations showed a 6.13× reduction in $R_\phi(\pi/2)$ native operations:



Adaption to Other Hardware

- The $XX(\alpha)$ and $R_\phi(\theta)$ are common native gates for ion trap hardware
  - See: IonQ hardware [1], Sandia QSCOUT testbed [4]
- But does other hardware restrict the θ angle in $R_\phi(\theta)$?
  - Aforementioned hardware does not, but it's not an uncommon choice. For example, the 2021 Honeywell machine limits θ to π or $\pi/2$ [6]
  - Adjusting our optimizer-based decomposition for these machines would be straightforward
- What about hardware with only global operations? (E.g., the current configuration of the GTRI testbed)
  - Possible with XACC, but our current backend is not totally compatible
  - However, any hardware programmed with a typical quantum gateset will benefit from existing high-level optimizations in QCOR [5]

Future Work

- Run this on actual hardware!
- Consider parallel two-qubit gates [2], non-$R_\phi(\pi/2)$ operations, make decompositions consider parallelism
- What about different hardware, like the 2021 Honeywell QCCD machine? Or TILT hardware? [6, 8]

Thank you!

- QCOR website: `https://qcor.ornl.gov/`
- Backend source code: `https://github.com/ausbin/xacc/tree/ion-trap-backend/`

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, 536(7614):63–66, Aug. 2016.
Number: 7614 Publisher: Nature Publishing Group.

C. Figgatt, A. Ostrander, N. M. Linke, K. A. Landsman, D. Zhu, D. Maslov, and C. Monroe.
Parallel entangling operations on a universal ion-trap quantum computer.
Nature, 572(7769):368–372, Aug. 2019.

C. D. Herold, S. D. Fallek, J. T. Merrill, A. M. Meier, K. R. Brown, C. E. Volin, and J. M. Amini.
Universal control of ion qubits in a scalable microfabricated planar trap.
New Journal of Physics, 18(2):023048, Feb. 2016.
Publisher: IOP Publishing.

B. C. A. Morrison, A. J. Landahl, D. S. Lobser, K. M. Rudinger, A. E. Russo, J. W. Van Der Wall, and P. Maunz.
Just Another Quantum Assembly Language (Jaqal).
In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 402–408, Oct. 2020.

References iii

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. 2020.
arXiv: 2010.03935.

J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, and B. Neyenhuis.
Demonstration of the trapped-ion quantum CCD computer architecture.
Nature, 592(7853):209–213, Apr. 2021.

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. 2020.
arXiv: 2011.08165.

X.-C. Wu, D. M. Debroy, Y. Ding, J. M. Baker, Y. Alexeev, K. R. Brown, and F. T. Chong.
TILT: Achieving Higher Fidelity on a Trapped-Ion Linear-Tape Quantum Computing Architecture.
In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 153–166, Feb. 2021.
ISSN: 2378-203X.