Quantum Computing and the Forest SDK

Robert Smith
2 February 2019
a quick poll
Rigetti Computing, in a nutshell

● Build universal, gate-based hybrid classical/quantum computers
  ○ Quantum computers are not more powerful than classical ones, yet
  ○ ... but they can do real, interesting computations

● Full-stack company
  ○ all in-house: design → manufacturing → ... → applications development

● Wide range of papers published

● Flagship product: Quantum Cloud Services
Quantum Cloud Services

- **Fastest** quantum programming environment available to the public
- SW+HW+Infra innovations give **30x** speed-up over HTTP services
  - 2 hours of computation becomes 4 minutes
- Personal Quantum Machine Image (QMI) with SSH access, preloaded with a full suite of advanced tools:
  - Compiler
  - Simulator
  - Python API
  - Optional libraries

**Forest SDK**
3 years ago, released an open standard for Quil
  ○ A portable quantum instruction language for hybrid computation
  ○ Language-independent: Python, OCaml, Lisp, JavaScript, ...

Since then, Rigetti has released a handful of OSS

<table>
<thead>
<tr>
<th>pyQuil</th>
<th>MAGICL</th>
<th>rpcq</th>
</tr>
</thead>
<tbody>
<tr>
<td>forest-benchmarking</td>
<td>oqaml</td>
<td>grove</td>
</tr>
<tr>
<td>ALEXA</td>
<td>cmu-infix</td>
<td>&amp; more</td>
</tr>
</tbody>
</table>

Many contributions back to OSS projects: CAD tools, testing libs, etc.
The Forest SDK

Applications

grove  forest-benchmarking  your app?

Program Construction & API

pyQuil

Quantum RPC Framework

rpcq

Compiler

quilc

Simulator

qvm  PyQVM

Quantum Computer Execution Stack

...
The Forest SDK: today’s talk

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>grove</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Construction &amp; API</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyQuil</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantum RPC Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpcq</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>quilc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulator Quantum Computer Execution Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvm</td>
</tr>
</tbody>
</table>
The Rigetti Quantum Virtual Machine: `qvm`

- Extremely **high-performance**: Eats all available CPU cores and RAM if you let it
- Can execute the **entire** Quil language
- Supports lots of execution modes
  - Standard & stochastic pure-state evolution (latter with Kraus operators)
  - Full density matrix evolution
  - Path integral formulation: calculate 1 amplitude with linear memory
- Simulates **perfect** and **imperfect** quantum computers
- Includes a compiler to **translate Quil into machine code**
  - Screaming fast execution, outperforms many simulators by 2x
The Rigetti optimizing Quil compiler: quilc

- The **only** general purpose, fully automatic, optimizing quantum compiler
- Built with **portability** in mind
  - Can compile to **user-specified** quantum architectures
- Can compile any unitary gate (2q, 3q, 4q, ... doesn’t matter)
- Has lots of special knowledge to do quantum equivalents of:
  - register allocation
  - peephole optimization
  - flow analysis and optimization
  - optimal compilation

One of the most amazing pieces of software I’ve worked on in my career.
demo

./quilc
cat bernstein-vazirani.quil | ./quilc -Pd
Fully automatic compilation is good!

- As if it were the 1950s, some software firms suggest we should be:
  - hand compiling quantum programs
  - have our programs always be aware of the target architecture
    - which changes every 6 months
  - writing un-portable code
  - ... because otherwise it “won’t be appropriate” for NISQ machines
- Computers are fast; what problems they can solve may surprise you
- If people can write C for microcontrollers, then they can write portable Quil for quantum computers

*quilc is a good & improving demonstration of that*
demo

cat bernstein-vazirani.quil | ./quilc -Pd --verbose
What does a compiler target look like?

- Generally a graph of qubits
- Each qubit supports a collection of single-qubit gates
  - Could be static or parametric
  - e.g., $RX(\pi/2)$, $RZ(\theta)$
- Each qubit-pair supports a collection of two-qubit gates
  - e.g., $CZ$, $CNOT$, $CPHASE(\theta)$
- Each qubit-{triplet, quadruplet, …} supports {3, 4, …}-qubit gates
  - The ion trap folks go nuts with these, e.g., Mølmer-Sørensen gate

Different qubits may be tuned for different operations!
quilc can compile for this architecture

Try hand-compiling a GHZ state on a quantum computer with this architecture!
For FOSDEM, we ported quilc...

- ... to Google’s Bristlecone architecture (72 qubits)
- ... to IBM’s ibmqx5 architecture (16 qubits)
- Any program written in Quil in whatever gate set will compile to Rigetti’s, Google’s, and IBM’s architectures portably
  - And quilc optimizes for them
- Can work on the full chip or any subgraph of it
- The only compiler that can do so?
demo

cat molmer.quil | ./quilc -Pd --isa 8Q

cat molmer.quil | ./quilc -Pd --isa bristlecone

cat molmer.quil | ./quilc -Pd --isa ibmqx5

cat molmer.quil | ./quilc -Pd --isa bristlecone --enable-state-prep-reductions
qvm & quilc are free to download

- Free downloadable installers for Linux, macOS, and Windows
  - Comes with a EULA
- Open-source alternative to qvm: PyQVM
  - Just released; part of pyQuil
  - FOSS license: Apache 2.0
  - Much slower for lots of qubits, doesn’t come with all the bells and whistles
- No real alternative to quilc
  - Follow folk advice and hand-compile?
Split open/closed source = Good for startups

Pros of Open Source

● Open source allows us to reap the rewards of sharing the parts that users mostly use so that the customer experience can be improved
● Using RPC and creating good APIs allows anybody to slot in their own open source variants
● Languages (like Quil) and APIs are best fostered as a part of an open source community

Pros of Closed Source

● Closed source programmer tools allow us to innovate, sell, make money, license, and write EULAs
● Can’t afford to “give everything for free” unlike the multi-billion dollar giants with tens or hundreds of thousands of employees
● Relying on the community for the most important tools is a haphazard bet. Otherwise Linux would be the #1 desktop OS
just kidding
qvm & quilc are written in Common Lisp

- Many innovations couldn’t have happened without it
  - Time & money budget aren’t infinite at a startup
  - Developing in Lisp is snappy
- Nobody has figured out expressive syntax for quantum computing
  - Lisp is great—even optimized—for metasyntactic experiments
- Debugging a compiler in Lisp with Emacs+SLIME is much nicer than in Python or C++
  - Optimizing compilers are very difficult to debug
- Our team primarily consists of first-time Lisp programmers
  - New employees are always productive in just a few days
A book about Lisp for programmers

**Practical Common Lisp**

free ebook online
$|\text{Beer}\rangle + |\text{You}\rangle / \sqrt{2}$ Challenge

The first 3 people to...

solve an issue · fix a bug · make a contribution

...will get a beer on me.

Email: robert@rigetti.com
Code: github.com/rigetti/qvm
Code: github.com/rigetti/quilc
Slack: rigetti.com/community