Software and Architectures for Large-Scale Quantum Computing

Fred Chong
Seymour Goodman Professor of Computer Architecture
Department of Computer Science
University of Chicago

with Ken Brown, Margaret Martonosi, Diana Franklin, Ravi Chugh, John Reppy, Ali Javadi Abhari, Jeff Heckey, Daniel Kudrow, Shruti Patil, Adam Holmes, Alexey Lvov, Sergey Bravyi
(GATech, Princeton, UChicago, IBM)
Why Quantum Computing?

- Factorization (Shor’s Algorithm)
  - $n^3$ instead of exponential
- Search (Grover’s Algorithm)
  - function evaluation
  - $\sqrt{n}$ instead of $n$

(CACM 2010)
Progress in QC Algorithms

http://math.nist.gov/quantum/zoo/
Recent excitement for Quantum Computing

- 5 and 7-bit machines were built 15 years ago [Vandersypen00, Laflamme99]
- Dwave 2048-bit quantum annealing
- Substantial investments by Google, Microsoft, IBM, USA, UK
- Large-scale machine in 10-20 years

(Dwave)
This Talk

- A systems perspective to accelerate QC development
  - Scalable architectures to guide device development
  - Software systems to enable pre-machine, large-scale applications work
- Eg. $10^6$ increase in efficiency in quantum chemistry (Microsoft)
  
  [arXiv:1403.1539v2]

(Chris Monroe, UMD)
Outline

- Introduction to Quantum Computing
- Lessons Learned
  - Specialization for reliability, parallelism, and performance
  - Managing compiler resources for deep optimization
  - Dynamic code generation for arbitrary rotations
- Future research
  - Validation of large quantum programs
  - Network routing to schedule surface code operations
INTRODUCTION TO QUANTUM COMPUTING
Quantum Bits (qubit)

- 1 qubit probabilistically represents 2 states

\[ |a\rangle = C_0|0\rangle + C_1|1\rangle \]

- Every additional qubit doubles the number of states

\[ |ab\rangle = C_{00}|00\rangle + C_{01}|01\rangle + C_{10}|10\rangle + C_{11}|11\rangle \]

- *Quantum parallelism* on an exponential number of states

- But measurement collapses qubits to single classical values
7-qubit Quantum Computer

(Vandersypen, Steffen, Breyta, Yannoni, Sherwood, and Chuang, 2001)

- Bulk spin NMR: nuclear spin qubits
- Decoherence in 1 sec; operations at 1 KHz
- Failure probability = $10^{-3}$ per operation
- Potentially 100 sec @ 10 KHz = $10^{-6}$ per op

- **pentafluorobutadienyl cyclopentadienyldicarbonyliron complex**
Not Gate

- Flips probabilities for $|0\rangle$ and $|1\rangle$
- Conservation of energy
  \[
  \sum_i C_i^2 = \alpha^2 + \beta^2 = 1
  \]
- Reversibility $\Rightarrow$ unitary matrix
  \[
  (X^*)^T X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = I
  \]

(* means complex conjugate)
Controlled Not

- Control bit determines whether X operates
- Control bit is affected by operation
Universal Quantum Operations

H Gate
Hadamard

T Gate

Z Gate
Phase-flip

Controlled Not
CNot

Controlled X

\[
\begin{pmatrix}
1 & 1 \\
1 & -1
\end{pmatrix}
\begin{pmatrix}
\alpha \\
\beta
\end{pmatrix} = \frac{(\alpha + \beta)}{\sqrt{2}} |0> + \frac{(\alpha - \beta)}{\sqrt{2}} |1>
\]

\[
\begin{pmatrix}
e^{-\frac{i\pi}{8}} & 0 \\
0 & e^{\frac{i\pi}{8}}
\end{pmatrix}
\begin{pmatrix}
\alpha \\
\beta
\end{pmatrix} = \alpha e^{-\frac{i\pi}{8}} |0> + \beta e^{\frac{i\pi}{8}} |1>
\]

\[
\begin{pmatrix}
1 & 0 \\
0 & -1
\end{pmatrix}
\begin{pmatrix}
\alpha \\
\beta
\end{pmatrix} = \alpha |0> - \beta |1>
\]

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
c \\
d
\end{pmatrix} = a |00> + b |01> + d |10> + e |11>
\]
Reliability is hard

- Quantum computing *should* be hard
  - Short lived
  - Small systems
- Can’t copy data (no-cloning theorem)
  - Need to protect it
- Can’t measure data
  - How do we detect errors?

(Credit: Harald Risch)
# Quantum Error Correction

<table>
<thead>
<tr>
<th>$X_{12}$</th>
<th>$X_{23}$</th>
<th>Error Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>no error</td>
<td>no action</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>bit 3 flipped</td>
<td>flip bit 3</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>bit 1 flipped</td>
<td>flip bit 1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>bit 2 flipped</td>
<td>flip bit 2</td>
</tr>
</tbody>
</table>

(3-qubit code)
Syndrome Measurement
3-bit Error Correction
Error Correction is Crucial

- Need continuous error correction
  - can operate on encoded data
    [Shor96, Steane96, Gottesman99]
- Threshold Theorem [Ahanorov 97]
  - failure rate of $10^{-4}$ per op can be tolerated
- Practical error rates are $10^{-6}$ to $10^{-9}$
Concatenated Codes

1 logical qubit

Level 1:
7 physical qubits

Reliability increases doubly exponentially.
Exponentially slower.
Exponentially greater resources.

Level 2:
49 physical qubits

Concatenated Steane Code
Error Correction Overhead

- 7-qubit code [Steane96], applied recursively

<table>
<thead>
<tr>
<th>Recursion time (k)</th>
<th>Storage (7^k)</th>
<th>Operations (153^k)</th>
<th>Min. (5^k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>153</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
<td>23,409</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>343</td>
<td>3,581,577</td>
<td>125</td>
</tr>
<tr>
<td>4</td>
<td>2,401</td>
<td>547,981,281</td>
<td>625</td>
</tr>
<tr>
<td>5</td>
<td>16,807</td>
<td>83,841,135,993</td>
<td>3125</td>
</tr>
</tbody>
</table>
Trapped Ions for Quantum Computation

- Trapping electrodes are attached on aluminum substrate
- Qubits are stored in the internal electronic states of each ion
- Lasers (or microwaves) implement logic gates and measurement
- Sympathetic recooling ions reduce vibrational heating

Quantum Teleportation?

- Two ions are “entangled” in an inseparable state.
- One is sent to Alice and one to Bob
- With some error, Alice can force Bob’s ion into resembling her data ion.
- By sending Bob two-bits of classical information of what the error is, Bob can accurately recreate the Data ion.

Bennett, et. al. PRL, v70, 1993; Barrett, et. al. Nature 429, 2004
LESSON 1: SPECIALIZATION
“Quantum FPGA”

[Metodi et al, Micro05]
Factoring an Integer

- **128-bit**: 63,730 Toffoli Gates with 21 ECC steps per Toffoli for modular exponentiation. Thus we have $21(63,730) + \text{QFT} = 1.34 \times 10^6$ time steps = ~16 hours. $\Rightarrow 16 \times 1.75 \Rightarrow \approx 21$ hours
- **512-bit**: 397.910 Toffoli Gates + QFT $\Rightarrow \approx 5.5$ days
- **1024-bit**: 964,919 Toffoli Gates + QFT $\Rightarrow \approx 13.4$ days
- **2048-bit**: 2,301,767 Toffoli Gates + QFT $\Rightarrow \approx 32$ days
Area Problem

Solution: Specialized Architecture Elements?

90cm x 90cm
Limited Parallelism

- **Modular Exponentiation Component**: The Draper Carry-Lookahead Adder (64-qubit Adder)
Specialization

Ancilla : Data
2 : 1

Compute Block

Ancilla : Data
1 : 8

Memory Block

Logical Data Qubits

Logical Ancilla Qubits
Area Reduced

![Area Reduced Factor of Shor's Alg. Adder Input Size](chart.png)
Area Reduced

90cm x 90cm

28cm x 28cm
Design Pyramid - Specialization

- Speed
- Reliability
- Area

Orig → Specialized

Speed
Area
Reliability
Concatenated Codes

1 logical qubit

Level 1: 7 physical qubits

Level 2: 49 physical qubits

Reliability increases doubly exponentially.

Exponentially slower.

Exponentially greater resources.
Faster Hierarchy

[Thaker et al, ISCA 2006]
Performance Benefits

Shor’s Alg. Adder Input Size

<table>
<thead>
<tr>
<th>Size</th>
<th>256-bit</th>
<th>512-bit</th>
<th>1024-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area Reduced</td>
<td>Factor of</td>
<td>Performance Change</td>
<td>Hierarchy: Area Reduced</td>
</tr>
<tr>
<td>256-bit</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>512-bit</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1024-bit</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Design Pyramid – Memory Hierarchy

Orig

Area

Speed

Reliability

Hierachy
LESSION 2: MANAGING COMPILER RESOURCES
The Scaffold Language and Compiler

- Extended C
  - No pointers
  - Quantum datatypes
  - Extensible gates
  - Parallel loops
  - Reversible logic
    synthesis for classical functions
    (includes fixed point arithmetic)

1 #include "gates.h"
2 module main ( ) {
3   int i=0;
4   qreg extarget[4];
5   qreg excontrol[4];
6   forall(i=0; i<4; i++) {
7     CNOT(extarget[i],excontrol[i]);
8   }
9 }

[Heckey et al, ASPLOS 2015]
Scalable Tailored QC Compilation

Quantum circuits often specialized to one problem input or size:

Benefits of Customization:
Efficient circuits. Deeply and statically analyzable.

Vs. Lack of Scalability:
Code explosion: $>10^{12}$ ops for some applications!

- Need better balance of optimization and scalability
  - QASM format changes: QASM-H, QASM-HL: 200,000X or more code size savings
  - Modular analysis
Effect of Remodularization

- Based on resource analysis, flatten modules with size less than a threshold
- Limited by memory on compilation machine
LESSON 3: DYNAMIC CODE GENERATION
Quantum Code Generation for Arbitrary Rotations

- Arbitrary rotations are important, difficult to compile for, and expensive to execute
- Unique sequence for every distinct rotation
  - Can be 4 TB of code!
- Sometimes need dynamic code generation
  - Rotation angles determined at runtime
  - Large code size

[Kudrow et al, ISCA 2013]
Rotation Decomposition

H gate
T gate
X gate
H gate
$T^\dagger$ gate
...

H gate
T gate
X gate
H gate
$T^\dagger$ gate
...
Rotation Decomposition

Scaffold QPL

module RotatePhi(qbit q) {
    Rz(q, Phi);
}

QASM

module RotatePhi(qbit q) {
    T q
    H q
    Z q
    H q
    T q
    Z q
    ...
}

Rotation gate

Decomposition
Precomputed Library

- Example: binary construction

Generate library:

Concatenate appropriate sequences to approximate desired angle:

T, H, T, Z, T, Z, H, ...
Results – Compilation Time

The graph illustrates the compilation time in seconds for different accuracy levels of approximation. The x-axis represents the accuracy of approximation, ranging from $10^{-11}$ to $0.1$. The y-axis shows the compilation time, ranging from $10^{-8}$ to 10 seconds.

Three data sets are plotted:
- **Solovay-Kitaev (SQCT)**: Represented by green dots and a green line.
- **Library Construction**: Represented by red triangles with error bars.

The data suggests that as the accuracy of approximation increases, the compilation time decreases. The Solovay-Kitaev data shows a more gradual decrease compared to the library construction data, which consistently shows lower compilation times at higher accuracies.
Results – Compilation Time

- Ion Trap
- Neutral Atom
- Superconductor
- Photons

Accuracy of approximation vs. Compilation time (seconds)
Dynamic Compilation Summary

- Up to 100,000X speedup for dynamic compilation with 5X increase in sequence length
FUTURE WORK 1: PROGRAM CORRECTNESS
How do I know if my QC program is correct?

- Need: Specification language for QC algorithms
- Check implementation against the specification
  - Simulation for small problem sizes (~30 qubits)
  - Symbolic execution for larger
  - Type systems
  - Model checking
  - Certified compilation passes
- Compiler checks general quantum properties
  - No-cloning, entanglement, uncomputation
- Checks or compiles based on programmer assertions too, where possible
**Programmer Assertion Example**

\[
b_{\text{eig} \ U} = \text{Eigenvalue}(b, U) \\
\text{CascadeU}(a, b, U) \\
\text{if not}(b_{\text{eig} \ U}) \\
\text{assert}(\text{Entangled}(a, b))
\]
FUTURE WORK 2: SCALABLE SURFACE CODE SCHEDULING
Surface Codes vs Network Routing
Summary

- QC is at an exciting time
- Software and architecture can generate key insights and accelerate progress
- With the right models and abstractions, classical techniques can have significant impact

https://github.com/ajavadia/ScaffCC.git
http://people.cs.uchicago.edu/~ftchong
BACKUP SLIDES
Longest Path First Scheduling

Strategy: Minimize qubit motion by assigning long dependence chains to a single compute region, where they can compute locally with little communication.
Algorithms in Scaffold

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean Formula</td>
<td>479</td>
</tr>
<tr>
<td>Linear Systems</td>
<td>1741</td>
</tr>
<tr>
<td>Binary Welded Tree</td>
<td>608</td>
</tr>
<tr>
<td>Class Number</td>
<td>226</td>
</tr>
<tr>
<td>Triangle Finding</td>
<td>1231</td>
</tr>
<tr>
<td>Shortest Vector Problem</td>
<td>539</td>
</tr>
<tr>
<td>Ground State Estimation</td>
<td>554</td>
</tr>
</tbody>
</table>
Scaffold Design Goals

- Completeness and Expressiveness
- Integration with toolchain
- Leveraging existing compiler infrastructures
- Familiarity and Ease of use
Scaffold Design Decisions

- Imperative Programming Model
- Variant of C and Verilog
- C2QG modules
- Control primitives
- Leveraging Open-Source Compilation
- Modular Design
- Aggressive Syntax Checking
Scaffold Safety

- No pointer arithmetic or arbitrary dereferencing
- Will use aggressive syntax and out-of-bounds checking at compile time
- Type casting only for classical variables
1 #include "gates.h"
2 module main ( ) {
3    int i=0;
4    qreg extarget[4];
5    qreg excontrol[4];
6    forall(i=0; i<4; i++) {
7        CNOT(extarget[i],excontrol[i]);
8    }
9 }
Data Types: Quantum Registers

- Quantum Registers
  - Basic use:
    - qreg example_reg[n];
    - example_reg[m]
    - example_reg[a..b]
    - length (example_reg)
  - Concatenation:
    - qreg reg1[p];
    - qreg reg2[q];
    - qreg reg = {reg1, reg2};
  - Individual qubits are also treated as qregs of size 1
    - qreg example_qubit[1];
Expressing Parallelism

- Parallelism will be important to runtime (and resource estimates)
- Initial coding of algorithms involved many `for` loops that need not be sequential
- Borrowing from data-parallel programming languages, we introduce a `forall` construct:
  - `forall` is a `for` loop in which all iterations are independent and can be executed in parallel
Forall Examples

- **Linear Systems**
  
  ```c
  // Apply H ^ tensor N to g
  for(i = 0; i < n_0; i++) {
      H(g[i]);
  }
  
  ```

- **Boolean Formula**

  ```c
  forall (i =0;i <(length(topregister)/2);i++){
      Swap (topregister [i], topregister [topregister.length -1-i]);
  }
  
  ...
  
  forall (index0=0; index0<w+m; index0++){
      TOFFOLI(register[index0], register[index0-m], ctrl[0]);
      TOFFOLI(register[index0-m], register[index0], ctrl[0]);
  }
  ```
Classical-code-to-Quantum-Gate-sequence (C2QG)

- Similar to C-to-HDL tools such as System-C
- Behavioral versus structural description
- Called with quantum variables as parameters
- Classical code within C2QG modules synthesized using reversible logic synthesis (BBRL or RMDDS)
- C2QG modules are *statically* synthesized
  - C2QG modules can only call C2QG modules internally
// Normal module representing a Toffoli gate
module Toffoli( qreg target,
    qreg control1,
    qreg control2){
    H(target);
    CNOT(target,control2);
    T(control2)
    Tdg(target);
    CNOT(target,control1);
    CNOT(control2,control1);
    Tdg(control2);
    T(target);
    CNOT(control2,control1);
    CNOT(target,control2);
    Tdg(target);
    CNOT(target,control1);
    T(target);
    T(control1);
    H(target);
}

// C2QG module representing a Toffoli gate
c2qg Toffoli( qint<1> target,
    qint<1> control1,
    qint<1> control2){
    if(control1 == 1 && control2==1) {
        target= ~target;
    }else {
        target=target;
    }
}
C2QG for Oracle Code

- C2QG body sent to reversible gate synthesis tool
Variable definitions

Operators: Normal C operators may be used between variables, but mixing qubits and classical will cause a compiler error.

Conditional and loop constructs: C conditional and iteration constructs are allowed, but they must be analyzable at compile time to allow the generation of reversible logic.

Calls to other c2qg modules: Within a C2QG module, only calls to other C2QG modules are allowed. Classical variables may be passed by reference.
Scaffold Control Flow

- **Runtime conditions**: compiler passes the conditions in classical assembly
- **Static classical conditions**: evaluated by the compiler and circuit generated
- **Qubit state conditions**: compiler generates code when using quantum bits as control lines
Quantum Control Primitive

- Compiler generates controlled versions of all gates within control conditionals

- Restrictions:
  - Control lines can not be input parameters
  - Only unitary operations in the conditional body

```plaintext
// Module prototypes. They are defined elsewhere
module U (qreg input[4], int n);
module V (qreg input[4]);
module W (qreg input[4], float p);

// Quantum Control Primitive
module control_example (qreg input[4]) {
    if (control_1[0]==1 && control_2[0]==1){
        U(input);
    }
    else if (control_1[0]==1 && control_2[0]==0){
        V(input);
    }
    else{
        W(input);
    }
}
```