Software and Architectures for Large-Scale Quantum Computing

Symposium in honor of Paul Benioff's fundamental contributions in quantum information

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

with Ken Brown, Margaret Martonosi, Diana Franklin, Isaac Chuang, John Reppy, Ali Javadi Abhari, Jeff Heckey, Daniel Kudrow, Shruti Patil, Adam Holmes, Alexey Lvov, Sergey Bravyi
(GATech, Princeton, UChicago, MIT, IBM)
This Talk

- A futuristic systems perspective
  - 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]
  - Apply tools and ideas to 100-qubit machine
Progress in QC Algorithms

http://math.nist.gov/quantum/zoo/
Outline

- **Lessons Learned**
  - Specialization for reliability, parallelism, and performance
  - Managing compiler resources for deep optimization
  - Dynamic code generation for arbitrary rotations

- **Future research**
  - Retarget SW tools for surface codes
  - Validation of quantum programs
  - What can we do with a 100-qubit machine?

(CACM 2010)
LESSON 1: SPECIALIZATION
“Quantum FPGA”

Classical Control Processors

Logical Qubit

Logical Qubit

Logical Qubit

Logical Qubit

[Metodi et al, Micro05]
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

![Diagram showing the factor of area reduction for different input sizes of Shor's Algorithm adders. The diagram includes bars representing the area reduced and performance change for 64-bit, 256-bit, 512-bit, and 1024-bit input sizes. The factor of area reduction ranges from 6.4 to 7.4, with a performance change of 9.1 for the 1024-bit input size.]
Faster Computation

1 logical qubit

Level 1:
7 physical qubits

Level 2:
49 physical qubits

Concatenated Steane Code

Reliability increases doubly exponentially.

Exponentially slower.

Exponentially greater resources.
Error-Correction Hierarchy

[Thaker et al, ISCA 2006]
Performance Benefits

The diagram illustrates the performance benefits of Shor's Algorithm for different adder input sizes. The x-axis represents the size of the adder input: 256-bit, 512-bit, and 1024-bit. The y-axis represents the factor of performance change.

- **Area Reduced**: Represented by black bars.
- **Perf. Change**: Represented by red bars.
- **Hierarchy: Area Reduced**: Represented by blue bars.
- **Hierarchy: Perf. Change**: Represented by yellow bars.

The graph shows a significant reduction in area with an increase in performance change as the input size grows from 256-bit to 1024-bit.
LESSON 2: MANAGING COMPILER RESOURCES
Deep Optimization

- QC similar to circuit synthesis for ASICs
- Program inputs known at compile time
  - Enables compiler optimizations
    - Constant propagation
    - Loop unrolling
- Scarce resources
  - Every qubit and gate is important
Execution Model

Scaffold → QASM → Classical Processor

Quantum Co-processor → Classical Processor
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)

```plaintext
#include "gates.h"
module main ( ) {
  int i=0;
  qreg extarget[4];
  qreg excontrol[4];
  forall(i=0; i<4; i++) {
    CNOT(extarget[i],excontrol[i]);
  }
}
```

[Heckey et al, ASPLOS 2015]
Tool Flow

Compilation
- Scaffold Quantum Program
- LLVM Infrastructure
- Fault-Tolerant Redundancies
- Physical Backend

Program Checks
- Modified Clang Parser
- Conversion to Reversible Circuit
- Classical Control Resolution
- Decomposing to Standard Gates
- LLVM Intermediate Representation
- Resource Estimation
- Module Minimizing and Flattening
- No-Cloning Verification
- Logical QASM Generation
- LPFS Scheduler
- Logical Schedule
- Architectural Simulator

Logical Backend
- Qubit Redundancy
- Fault-Tolerant Gate Conversion
- Zero States for ECC
- Magic States for T Gates
- EPR states for Teleportation
- Physical QASM Generation
- Communication Optimizer
- Physical Scheduler
- Physical Schedule

Logical Schedule

Logical Backend

Fault-Tolerant Redundancies

Physical Backend

https://github.com/epiqc/ScaffCC
## 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>
<tr>
<td>Shor’s Algorithm (invert SHA-1)</td>
<td>1055</td>
</tr>
<tr>
<td>Grover’s Algorithm (invert SHA-1)</td>
<td>388</td>
</tr>
<tr>
<td>Ising Model</td>
<td>113</td>
</tr>
</tbody>
</table>
Tool Output: Resource Estimation

- **Binary Welded Tree Call Graph**
  - Shows quantum resources used at each module
  - Maximum qubits used: 911 (for n=300)
Effect of Remodularization

- Based on resource analysis, flatten modules with size less than a threshold
- Limited by memory on compilation machine
Mapping Qubits

- Modified heuristic graph partitioner
  - based on Metis [Karypis and Kumar, 1995]
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.
Tool Output: Speedup Estimates

![Graph showing speedup estimates for different benchmarks (BF, BWT, CN, Grovers, GSE, SHA1, Shors, TFP) with bars representing Multi-SIMD (4,1), Multi-SIMD (4,4), and Multi-SIMD (4,64).]
Small-Scale Simulation Path

- Simulation effort at TU Delft
- Takes Scaffold QASM output
- Optimized for Intel supercomputing resources [http://www.xpu-project.net/qx/download.html](http://www.xpu-project.net/qx/download.html)
- Other closed-source tools: LIQUId [http://github.com/msr-quarc/Liquid](http://github.com/msr-quarc/Liquid)
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]
Dynamic Code Generation

- Scaffold
- QASM
- Classical Processor
- Quantum Co-processor

Static Compilation

Dynamic Compilation
Rotation Decomposition

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

Diagram showing the rotation decomposition of various quantum gates.
Rotation Decomposition

Scaffold QPL

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

QASM

```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: $T, H, T, Z, T, Z, H, ...$

Concatenate appropriate sequences to approximate desired angle:
Results – Compilation Time

![Compilation Time Graph]

- Solovay-Kitaev
- SQCT
- Library Construction
Results – Compilation Time

The diagram illustrates the compilation time for different quantum computing models: Ion Trap, Neutral Atom, Superconductor, and Photons. The x-axis represents the accuracy of approximation, ranging from $10^{-11}$ to $10^{-1}$, while the y-axis shows the compilation time in seconds, ranging from $10^{-8}$ to $10^{1}$. The graph compares the Solovay-Kitaev Quantum Circuit (SQCT) and Library Construction methods for each of these models.
Dynamic Compilation Summary

- Up to 100,000X speedup for dynamic compilation with 5X increase in sequence length (T-gate depth)
FUTURE WORK 1: TARGETING SURFACE CODES
Surface vs Concatenated Codes

- Less sensitive to communication distance
- Sensitive to braid crossings
  - Serializes communication
  - Qubit mapping for locality is important
- Network routing heuristics for scalable scheduling
FUTURE WORK 2:
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 problems
  - Type systems
  - Model checking
  - Certified compilation passes
- Compiler checks general quantum properties
  - No-cloning, entanglement, uncomputation
- Checks based on programmer assertions where possible
FUTURE WORK 3:
ENABLING A PRACTICAL-SCALE QUANTUM COMPUTER (EPIQC)
“Practical” Quantum Computing

- Algorithms and Software for a 100-qubit quantum computer
  - Chong, Reppy, Franklin, Schuster (UChicago), Shor, Farhi, Harrow (MIT), Brown (GATech), Harlow (UCSB)

- Fill the gap between theory and experiment
  - Expose physical effects to software and algorithms
  - Exhaustive optimizations
  - Compiler analysis and partial simulation for correctness
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/epiqc/ScaffCC
http://people.cs.uchicago.edu/~ftchong