Programming Languages and Compilers for Quantum Computers

Jason Briceno
M.S. Computer Science
Columbia '12
Introduction

- Quantum Computation is a fundamentally different model of computation
  - Advantages and disadvantages

- Relatively new field with few (if any) mature technologies
  - Deutsch-Josza Algorithm, 1992
  - IBM NMR quantum computer, 1996

- QuID is a language for describing quantum computations

- Ask questions
Presentation Outline

• Example Quantum Algorithms

• Quantum Mechanics & Computation
  • Physics
  • Postulates
  • Quantum Circuit

• QuID (Quantum Imperative Description)
  • Design Goals
  • Language
  • Compiler
Deutsch-Josza

Query: A function $f : \{0,1\}^n \rightarrow \{0,1\}$ that is either constant or balanced.

Constant: All inputs result in the same output.
Balanced: An equal number of inputs result in 0 and 1.
Deutsch-Josza

Query: A function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that is either constant or balanced.

Constant: All inputs result in the same output.
Balanced: An equal number of inputs result in 0 and 1.

How many queries to find constant or balanced?

Classical Algorithm
Number of Queries:
Deutsch-Josza

Query: A function \( f : \{0,1\}^n \rightarrow \{0,1\} \) that is either constant or balanced.

**Constant:** All inputs result in the same output.

**Balanced:** An equal number of inputs result in 0 and 1.

How many queries to find constant or balanced?

Classical Algorithm

Number of Queries:

\[
\frac{2^n}{2} + 1
\]
**Deutsch-Josza**

**Query:** A function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that is either *constant* or *balanced*.

**Constant:** All inputs result in the same output.

**Balanced:** An equal number of inputs result in 0 and 1.

How many queries to find *constant* or *balanced*?

**Classical Algorithm**

Number of Queries:

$$\frac{2^n}{2} + 1$$

**Quantum Algorithm**

Number of Queries:
Deutsch-Josza

Query: A function $f : \{0,1\}^n \rightarrow \{0,1\}$ that is either constant or balanced.

Constant: All inputs result in the same output.
Balanced: An equal number of inputs result in 0 and 1.

How many queries to find constant or balanced?

Classical Algorithm
Number of Queries:

$$\frac{2^n}{2} + 1$$

Quantum Algorithm
Number of Queries:

1
Other Algorithms

• Search
  • Given a list of $N$ elements, find a single “correct” element.
  • Classical: $O(N)$
  • Quantum: Grover's Algorithm $- O(\sqrt{N})$

• Factoring
  • Factoring a large integer $N$.
  • Classical: Number Field Sieve $- O\left(2^{c(\log N)^{1/3}(\log \log N)^{2/3}}\right)$
  • Quantum: Shor's Algorithm $- O\left((\log N)^3\right)$
Quantum Mechanics

- Normally occurs on a sub-microscopic scale
- Particles exist in probabilistic states – superpositions between possible measurement outcomes
- Measurements cause states to collapse – a state gets “chosen”
- Quantum mechanical elements that have two possible measurement outcomes can be used as qubits
Physical Examples

- Photons – Polarization
- Electrons – Spin
- Trapped Ions – Energy State
- Josephson Junction – Flux Qubit
Computational Thinking for Quantum Computing

The Four Postulates of Quantum Mechanics

M. A. Nielsen and I. L. Chuang
Quantum Computation and Quantum Information
Cambridge University Press, 2000
State Space Postulate

Postulate 1

The state of an isolated quantum system can be described by a unit vector in a complex Hilbert space.
Qubit: Quantum Bit

- The state of a quantum bit can be described by a unit vector in a 2-dimensional complex Hilbert space (in Dirac notation)

\[ |\psi\rangle = \alpha |0\rangle + \beta |1\rangle \]

where \( \alpha \) and \( \beta \) are complex coefficients called the amplitudes of the basis states, and

\[ |\alpha|^2 + |\beta|^2 = 1 \]
Time-Evolution Postulate

Postulate 2

The evolution of a closed quantum system can be described by a unitary operator $U$. (An operator $U$ is unitary if $U^*U = I$.)

\[ |\psi\rangle \rightarrow U |\psi\rangle \]

state of the system at time $t_1$  
state of the system at time $t_2$
NOT Operator

\[ |\psi\rangle \rightarrow X \rightarrow X|\psi\rangle \]
NOT Operator

\[ |\psi\rangle \rightarrow X \rightarrow X|\psi\rangle \]

\[ |0\rangle \rightarrow X \rightarrow |1\rangle \]
NOT Operator

\[ |\psi\rangle \rightarrow X \rightarrow X|\psi\rangle \]

\[ |0\rangle \rightarrow X \rightarrow |1\rangle \]

\[ |1\rangle \rightarrow X \rightarrow |0\rangle \]
NOT Operator

\[ |\psi\rangle \rightarrow X \rightarrow X|\psi\rangle \]

\[ |0\rangle \rightarrow X \rightarrow |1\rangle \]

\[ |1\rangle \rightarrow X \rightarrow |0\rangle \]

\[ \alpha|0\rangle + \beta|1\rangle \rightarrow X \]
NOT Operator

\[ |\psi\rangle \rightarrow X \rightarrow X |\psi\rangle \]

\[ |0\rangle \rightarrow X \rightarrow |1\rangle \]

\[ |1\rangle \rightarrow X \rightarrow |0\rangle \]

\[ \alpha |0\rangle + \beta |1\rangle \rightarrow X \rightarrow \alpha |1\rangle + \beta |0\rangle \]
Composition-of-Systems Postulate

Postulate 3

The state space of a combined physical system is the **tensor product** space of the state spaces of the component subsystems.

If one system is in the state $|\psi_1\rangle$ and another is in the state $|\psi_2\rangle$, then the combined system is in the state $|\psi_1\rangle \otimes |\psi_2\rangle$.

$|\psi_1\rangle \otimes |\psi_2\rangle$ is often written as $|\psi_1\rangle|\psi_2\rangle$ or as $|\psi_1\psi_2\rangle$. 
Two Qubit Example

- The state of a two quantum bits can be described by a unit vector in a 4-dimensional complex Hilbert space (in Dirac notation)

\[ |\psi\rangle = a_{00} |00\rangle + a_{01} |01\rangle + a_{10} |10\rangle + a_{11} |11\rangle \]

And,

\[ |a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1 \]
The CNOT gate maps
\[ |00\rangle \mapsto |00\rangle, \quad |01\rangle \mapsto |01\rangle, \quad |10\rangle \mapsto |11\rangle, \quad |11\rangle \mapsto |10\rangle \]
Quantum measurements can be described by a collection \( \{ M_m \} \) of operators acting on the state space of the system being measured. If the state of the system is \( |\psi\rangle \) before the measurement, then the probability that the result \( m \) occurs is

\[
p(m) = \langle \psi | M_m^\dagger M_m | \psi \rangle
\]

and the state of the system after measurement is

\[
\frac{M_m |\psi\rangle}{\sqrt{\langle \psi | M_m^\dagger M_m | \psi \rangle}}
\]
Measurement

The measurement operators satisfy the completeness equation:

$$\sum_{m} M_{m}^{\dagger} M_{m} = I$$

The completeness equation says the probabilities sum to one:

$$\sum_{m} p(m) = \sum_{m} \langle \psi | M_{m}^{\dagger} M_{m} | \psi \rangle = 1$$
Measurements Explained

• Quantum mechanical magic

• When a system is measured, it “chooses” a basis state

• The probability of the choice is just the square of the amplitude

• The system is left in the basis state after the measurement

• **Limitation:** All states can exist superposed during evolution, but only a single state exists after measurement
Hadamard Operator

\[ |\psi\rangle \rightarrow H \rightarrow H |\psi\rangle \]

\[ |0\rangle \rightarrow H \rightarrow \frac{1}{\sqrt{2}} |1\rangle + \frac{1}{\sqrt{2}} |0\rangle \]

\[ |1\rangle \rightarrow H \rightarrow \frac{1}{\sqrt{2}} |1\rangle - \frac{1}{\sqrt{2}} |0\rangle \]
Deutsch-Josza Diagram

The measurement will be $|0000\rangle$ iff $f$ is constant.

Any other measurement indicates that $f$ is balanced.
The Compiler
Quantum Computer Compiler

QIR: quantum intermediate representation
QASM: quantum assembly language
QPOL: quantum physical operations language

Computational abstractions

K. Svore, A. Aho, A. Cross, I. Chuang, I. Markov
A Layered Software Architecture for Quantum Computing Design Tools
Quantum Languages

• Quantum computation languages already exist

• Examples: QCL, Q Language, QML

• However, the focus is on *simulation* rather than *description*

• Difficult to hand to another technology for simulation or application.
QASM – Quantum Assembly Language

- Developed by:
  - Krysta Svore (PhD, Columbia)
  - Andrew Cross (PhD, MIT)
  - Isaac Chuang (MIT)

- Designed with circuit description in mind

- Can create complete circuit description

- Tedious to create large circuits

```qasm
# Qubit Declarations
#
qubit x000,0
qubit x001,0
qubit x002,0
qubit x003,0

# Main Body
#
H x000
nop x001
nop x002
nop x003
RK(2).1.1 x001,x000
nop x002
nop x003
nop x001
RK(3).1.1 x002,x000
nop x003
nop x001
nop x002
```
QuID Design Goals

- Quantum circuit description
- High-Level
- Quantum notation
- Flexible intermediate representations
- QASM output
QuID Example Algorithm – QFT

• Quantum Fourier Transform
  • Discrete Fourier Transform on state

• Important for many algorithms

• Used (along with order-finding) in Shor's Algorithm

```cpp
operator QFT();
operator CRK(qbv ctl, int j);

int q_main()
{
    qbv x[4];
    QFT |x>;
    return 0;
}

operator QFT()
{
    int i, j, k;
    for (i = 0; i << q_size(psi); i++)
    {
        H |psi[i]>
        k = 2;
        for (j = i + 1; j << q_size(psi); j++)
        {
            CRK(psi[j], k) |psi[i]>
            k++;
        }
    }
    SWAP|psi>;
}

operator CRK(qbv ctl, int j) : control(ctl)
{
    RK(j) |psi>;
}
```
QuID Language

- C-like syntax
- Variables: int, float, qbv
- qbv – special qubit handler
- Operators (special functions)
qbv: Qubit Vector

- A vector of qubits
- Similar to an array – easy way to handle qubit groups
- Can allocate on initialization
- Can reference other qbv
- Index and subset return qbv
- Concatenation operator (&)

```python
def example_code(
    qbv x[4];  # allocated qubits
    qbv y = x;  # register

    qbv sub_x, sub_y, whole;

    sub_x = x[0];  # qbv index
    sub_y = y[1:3];  # qbv subset

    whole = sub_x & sub_y;  # qbv concatenation
)```

Operator Use

- Special kind of function
- Declared as an operator
- Takes two sets of parameters
  - “normal” (optional)
  - Operating qbv
- Returns operating qbv
- Conjugate Transpose
  - Reverses operator

```cpp
operator QFT();
operator MAGIC(int rot);

qbv x[4];

QFT |x>;       # operator without params
MAGIC(2) |x>;  # operator with params
QFT^ |x>;      # conjugate transpose
```
Operator Definition

- Implicit $\psi$ qbv
  - Operating qbv

- Special function: `p_size()`
  - Size of a qbv

- `control()`, `ncontrol()`
  - Define control qubits

```cpp
operator CRK(qbv ctl, int j);

// controlled operation
operator CRK(qbv ctl, int j) : control(ctl)
{
    if (p_size(psi) > 2)
        RK(j) |psi>;
}
```
Primitive Operators

• Atomic units of operations

• Operators: CNOT, G, H, RK, RX, RY, RZ, SWAP, TOFF, X, Y, Z

• Undefined operators
  • Declared, but not defined

```cpp
CNOT |psi>
G(1.7) |psi>
H |psi>
RK(2) |psi>
RX(1.7) |psi>
RY(1.7) |psi>
RZ(1.7) |psi>
SWAP |psi>
TOFF |psi>
X |psi>
Y |psi>
Z |psi>

operator UNDEF();
```
Compiler

- Originally written as a PLT I project.

- Re-written over the summer.

- g++ compiler does much of the heavy lifting

- Used a number of third-party tools and libraries
  - Lex & Yacc
  - tree.hh (AST)
  - Boost C++ Libraries (QCG and misc. tools)

- Intermediate structures can be output as graphviz
Parser

• Used Lex & YACC

• Explicitly creates an Abstract Syntax Tree (AST)
Abstract Syntax Tree

• A new node is created for each production.

• Terminal nodes represent tokens.

• Created using tree.hh

• AST Walker
  • Recursively crawls AST
  • Produces intermediate.cc
External Compilation & Execution

- Intermediate C++ file

- Compiled with QuID Library to create intermediate executable

- Executable is run

```cpp
#include "ql_qbv.h"
#include "ql_hermitian.h"
#include "ql_operators.h"
#include "ql_qcg.h"
#include "ql_utilities.h"
using namespace ql;

qbv _op_QFT(qbv psi, qbv _arg_control, const QuidHermitian& _arg_qh, const QuidReverse& _arg_qr);
qbv _op_CRK(qbv ctl, int j, qbv psi, qbv _arg_control, const QuidHermitian& _arg_qh, const QuidReverse& _arg_qr);

int q_main()
{
    qbv _quid_control;
    QuidHermitian _quid_qh = QH_NONE;
    qbv x(4, "x");
    _op_QFT(x, _quid_control, _combine_qh(_quid_qh, QH_NONE), QR_NONE);
    return 0;
}
```
QuID Library: Quantum Circuit Graph

- A Directed Acyclic Graph
- A node for every operator (including no-op)
- Synchronization nodes between operators
- All nodes have balanced in and out edges
  - Except start and stop nodes
- Edges weighted by input/output order
- Uses boost::graph
QCG Walker – QASM Output

- Enumerates “new” operator nodes
- Enumerates qubits
- Breadth-First Search
  - Starting at start node
  - Outputs operators when encountered
- Outputs QASM

```
# # Produced by the QuID compiler.
#

# # Operator Declarations
#
   def RK(2).1.1,1,'RK(2)'
   def RK(3).1.1,1,'RK(3)'
   def RK(4).1.1,1,'RK(4)'

# # Qubit Declarations
#
   qubit x000,0
   qubit x001,0
   qubit x002,0
   qubit x003,0

# # Main Body
#
   H x000
   nop x001
   nop x002
   nop x003
```
qasm2circ

- QASM is hard to read directly

- qasm2circ produces nice PDFs given a QASM file
Complete Example

• Input QuID source – Deutsch-Josza Algorithm

```cpp
operator UF();

int q_main()
{
    qbv x[4];

    qbv y[1];
    X |y>;

    H |x & y>;
    UF |x & y>;
    H |x>;

    return 0;
}
```
// BEGIN
#include "ql_qbv.h"
#include "ql_hermitian.h"
#include "ql_operators.h"
#include "ql_qcg.h"
#include "ql_utilities.h"

using namespace ql;

qbv _op_UF(qbv psi, qbv _arg_control, const QuidHermitian& _arg_qh, const QuidReverse& _arg_qr);

int q_main()
{
    qbv _quid_control;
    QuidHermitian _quid_qh = QH_NONE;
    qbv x(4, "x");
    qbv y(1, "y");
    _op_X(y, _quid_control, _combine_qh(_quid_qh, QH_NONE), QR_NONE);
    _op_H(x & y, _quid_control, _combine_qh(_quid_qh, QH_NONE), QR_NONE);
    _op_UF(x & y, _quid_control, _combine_qh(_quid_qh, QH_NONE), QR_NONE);
    _op_H(x, _quid_control, _combine_qh(_quid_qh, QH_NONE), QR_NONE);
    return 0;
}

qbv _op_UF(qbv psi, qbv _arg_control, const QuidHermitian& _arg_qh, const QuidReverse& _arg_qr)
{
    return _op_undefined("UF", psi, _arg_control, _arg_qh, _arg_qr);
}

// END
Output QASM file, and qasm2circ visualization
Future Work

- Bug-fixing, documentation, efficiency
- Syntax improvements
- Alternate target languages
- Alternate operator primitives
- Circuit optimization
- Circuit error correction
Conclusion

- Quantum algorithms have a potential for exponential speed-up compared to classical counterparts

- Limitations of quantum computing have yet to be proven

- Quantum computing is still a young field

- QuID is just a start
Additional Resources

- Quantum Mechanics
  - Feynman Lecture Series (http://vega.org.uk/video/subseries/8)

- QuID
  - tree.hh Library (http://tree.phi-sci.com/)

- Boost C++ Libraries (http://www.boost.org/)

- Code & Documentation