Review Approval

- Prepare Request
- Search Requests
- Generate Reports
- Approvals
- Help
- Wizard

Search Requests

- New Search
- Refine Search
- Search Results
- Clone Request
- Edit Request
- Cancel Request

# Search Detail

## Submittal Details

**Document Info**

Title : Quantum Programming for Classical Programmers

Document Number : 5236851 SAND Number : 2005-7966 P

Review Type : Electronic Status : Approved

Sandia Contact : DEBENEDICTIS,ERIK P. Submittal Type : Viewgraph/Presentation

Requestor : DEBENEDICTIS,ERIK P. Submit Date : 12/19/2005

Comments : R&A to permit sharing of this document with specific non-Sandian project collaborators.

Peer Reviewed? : N

**Author(s)**

DEBENEDICTIS,ERIK P.

**Event (Conference/Journal/Book) Info**

Name : No event

City : No City State : NM Country : USA

Start Date : 01/01/2006 End Date : 01/01/2006

**Partnership Info**

Partnership Involved : No

Partner Approval : Agreement Number :

**Patent Info**

Scientific or Technical in Content : Yes

Technical Advance : No TA Form Filed : No

SD Number :

**Classification and Sensitivity Info**

Title : Unclassified-Unlimited Abstract : Document : Unclassified-Unlimited

Additional Limited Release Info : None.

DUSA : None.

## Routing Details

| Role | Routed To | Approved By | Approval Date |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **Derivative Classifier Approver** | HUDGENS,JAMES J. | HUDGENS,JAMES J. | **12/20/2005** |
| **Conditions:** | | | |
| | | | |
| **Classification Approver** | WILLIAMS,RONALD L. | WILLIAMS,RONALD L. | **12/21/2005** |
| **Conditions:** | | | |
| | | | |
| **Manager Approver** | PUNDIT,NEIL D. | PUNDIT,NEIL D. | **12/21/2005** |
| **Conditions:** | | | |
| | | | |
| **Administrator Approver** | LUCERO,ARLENE M. | | |
| | | | |

SAND 2005-7966P

# Quantum Programming for Classical Programmers

Erik P. DeBenedictis

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the
United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

# Overview

- Target Audience
  - Classical programmers who want to know what quantum computer programming is all about
- Limitations of this Approach
  - Only small quantum computers can be simulated
- The following limitations change the form of expression but do not limit expressive power
  - Uses only the computational basis
  - Only simulates Von Neumann measurements

# Outline

- **Representation of Qubits**
- **Non-Entangling Operations**
- **Entangling Operations**
- **Measurements**
- **Addition**

# Quantum Register

Complex amplitudes

n bits
|quantum state>

$1 \leq k \leq 2^n$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1/2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $1/2\ e^{i\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $1/2\ e^{i\pi/2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $1/2\ e^{i3\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Qubit n-1 → Qubit 2 → Qubit 1 → Qubit 0 →

The data are just examples

# Quantum Register

```
typedef double R;               // R for real number
typedef long Bits;              // Bits for bit vector

struct C {                      // complex number
       R re, im;
};

struct Superposition {
       C Amplitude;             // amplitude of superposition
       Bits State;              // state
};

struct QubitRegister {
       int Qubits;              // number of qubits
       int Num;                 // number of non-zero superpositions
       Superposition *Vec;      // pointer to superpositions
       void Rotate(int, R);     // universal set of operations
       void CNot(int, int);
       int Measure(int);
};
```

Sandia National Laboratories

# Notes on State Representation

- Normalization
  - In a quantum register, the sum of amplitudes squared needs to be 1
  - Quantum operations will preserve normalization up to numerical stability
  - This means code needs to periodically check normalization and take appropriate action
- Zero Amplitude States
  - All $2^n$ states can be imagined to exist, with those not explicitly allocated having zero amplitude
- Global Phase
  - Multiplying all amplitudes by the same complex phase factor does not change anything

Sandia National Laboratories

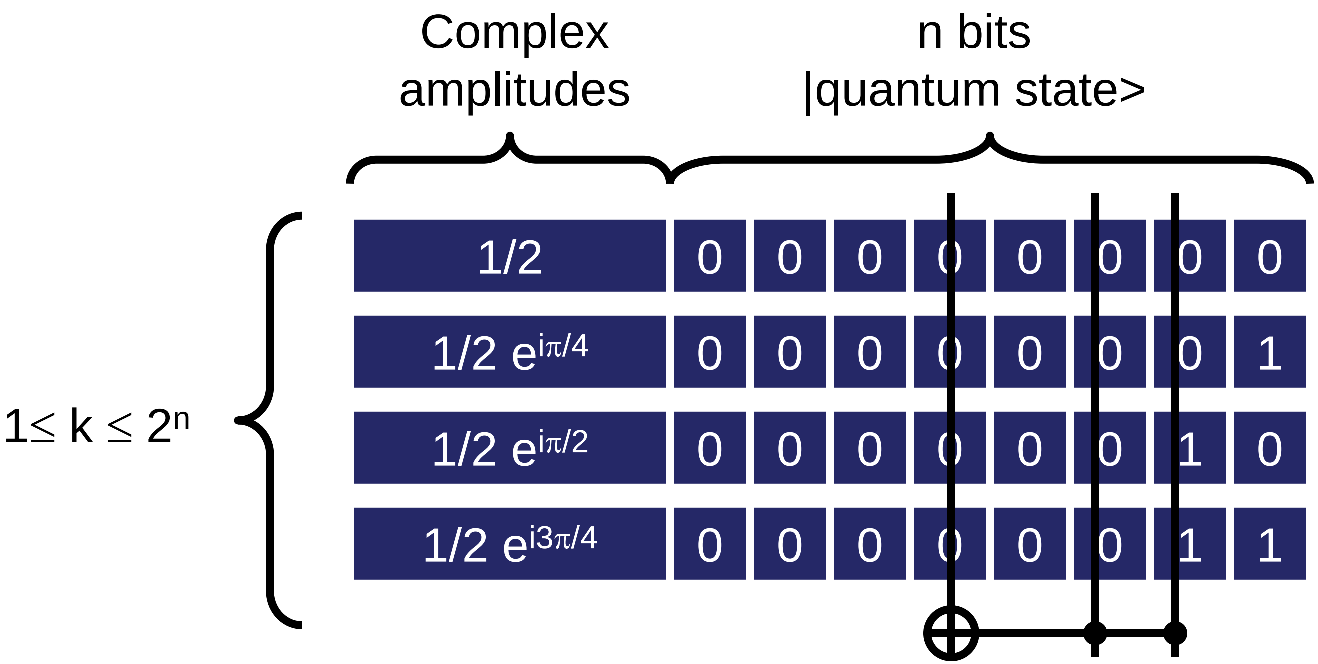
# Notes on State Representation

- Number of Qubits
  - Algorithms that fill the quantum superposition space will bog down a classical computer before exceeding 32 qubits
  - On the other hand, other algorithms can use >32 qubits
  - Therefore, provide the option of >32 qubits
- Memory Allocation
  - Some key algorithms start with a sparsely filled superposition space and end with a QFT largely filling the superposition space
  - Therefore, allocate states dynamically

# Outline

- Representation of Qubits
- Non-Entangling Operations
- Entangling Operations
- Measurements
- Addition

# Non-entangling Operations




Non-entangling operations execute logic operations on the qubit values in the superposition states without changing the number of states or the amplitudes.

Non-entangling operations include Not, CNot, Toffoli

# Quantum Not and CNot

```
void QubitRegister::Not(int Qubitnum) {
        Bits flip = 1<<Qubitnum;
        for (int i = 0; i < Num; i++)
                Vec[i].State ^= flip;
};

// Note: Cnot can be simulated as Toffoli with inputs tied together,
// which will make Toffoli the most frequently used operation
// Author’s actual implementation of Toffoli is highly optimized
void QubitRegister::Toffoli(int C1, int C2, int Bit) {
        Bits c1 = 1<<C1;
        Bits c2 = 1<<C2;
        Bits flip = 1<<Bit;
        for (int i = 0; i < Num; i++)
                if ((Vec[i].State&c1) != 0 && (Vec[i].State&c2) != 0)
                        Vec[i].State ^= flip;
}
```

# Outline

- Representation of Qubits
- Non-Entangling Operations
- Entangling Operations
- Measurements
- Addition

# Entangling Operations

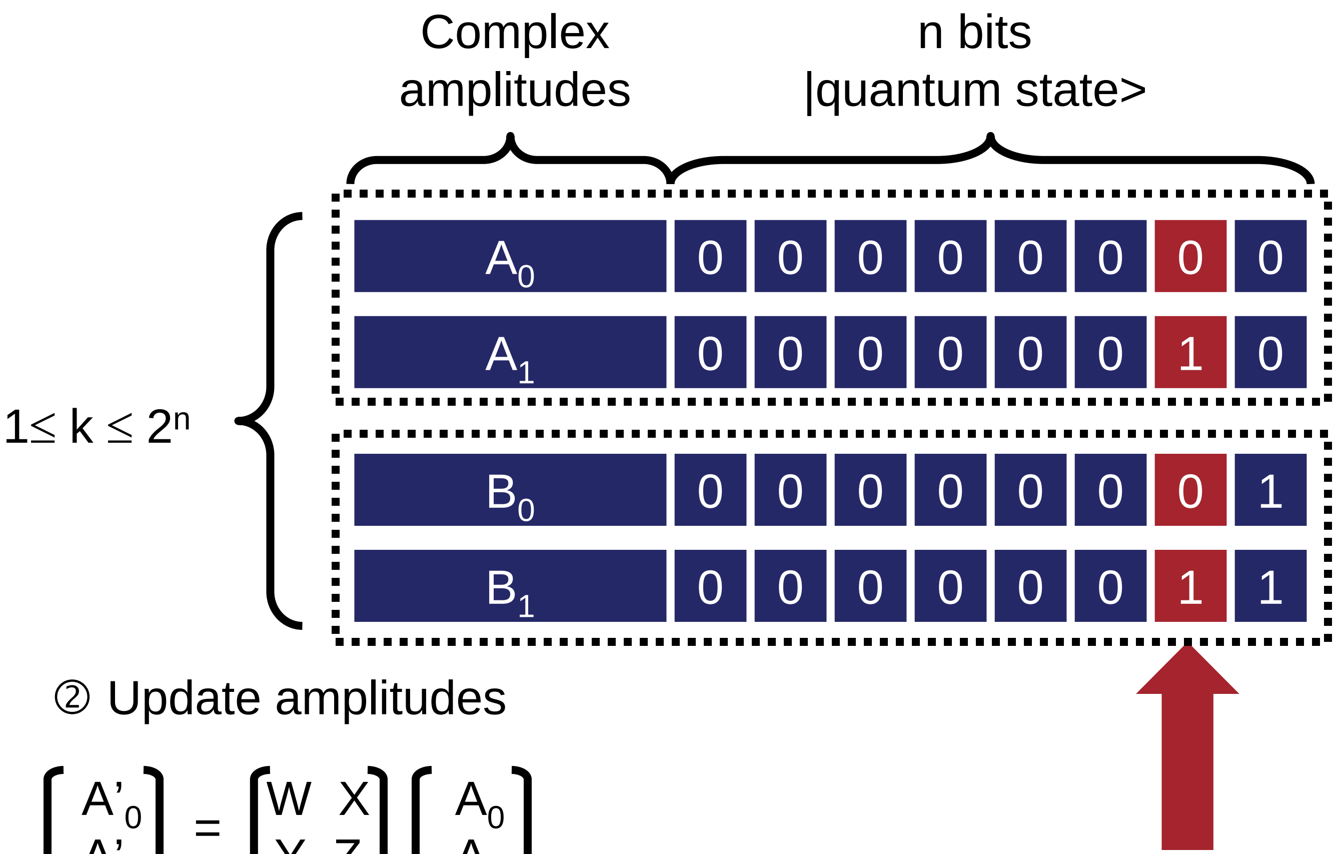
Complex amplitudes — n bits |quantum state>

$1 \leq k \leq 2^n$

| Complex amplitudes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1/2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $1/2\ e^{i\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $1/2\ e^{i\pi/2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $1/2\ e^{i3\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Say you want to rotate this qubit

# Entangling Operations




← ① Create pairs of superposition states differing only in the designated qubit

Say you want to rotate this qubit

② Update amplitudes

$$\begin{bmatrix} A'_0 \\ A'_1 \end{bmatrix} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$$

$$\begin{bmatrix} B'_0 \\ B'_1 \end{bmatrix} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} \quad \ldots$$

Sandia National Laboratories

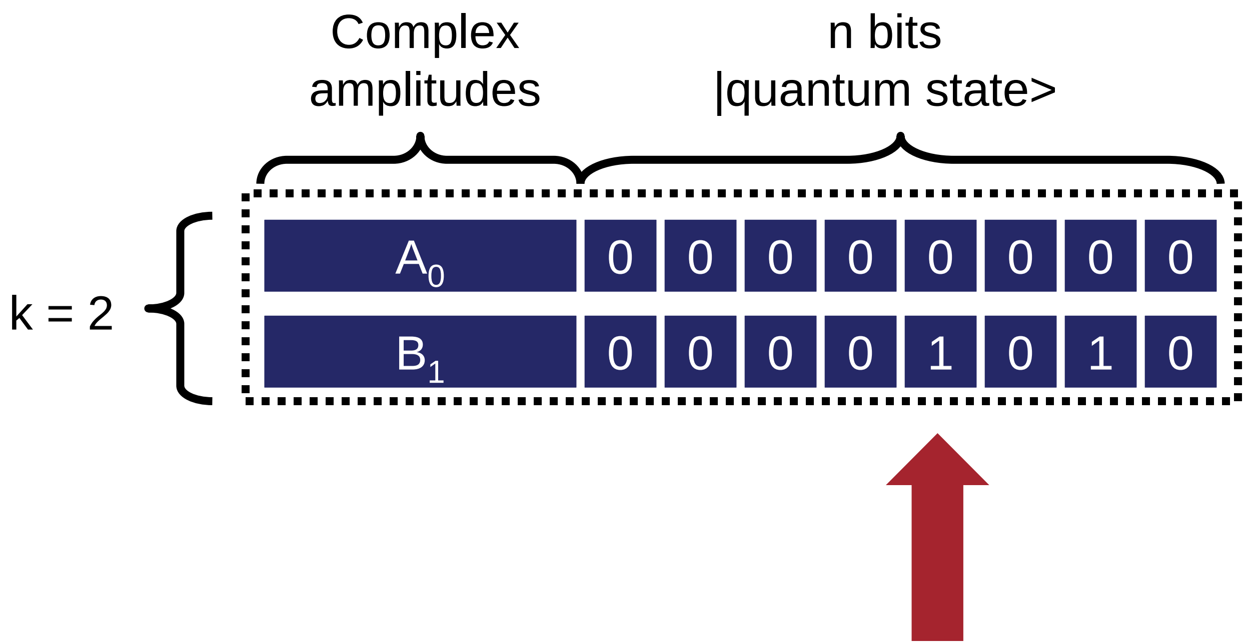
# Entangling Operations

- **Memory Allocation**
  - **Every superposition state must be paired with another state that differs only in the designated bit EVEN IF THAT STATE DOES NOT EXIST**
  - **If the state does not exist, it must be allocated, increasing memory usage**

- **Entangling operations include Hadamard, which is defined by**

$$\begin{bmatrix} A'_0 \\ A'_1 \end{bmatrix} = 1/\sqrt{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$$

# Hadamard Example




Say you want to do a
Hadamard on this qubit

# Entangling Operations

k = 4

| Complex amplitudes | n bits \|quantum state> | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $A_0$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $A_1$=0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $B_0$=0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $B_1$ | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

← ① Create pairs of superposition states differing only in the designated qubit, allocating new ones from dynamic memory with zero amplitude

Say you want to do a Hadamard on this qubit

② Update amplitudes

$$\begin{bmatrix} A'_0 \\ A'_1 \end{bmatrix} = 1/\sqrt{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$$

$$\begin{bmatrix} B'_0 \\ B'_1 \end{bmatrix} = 1/\sqrt{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} \ldots$$

Sandia National Laboratories

# Exemplary Method

- **Sort the superposition states such that states differing only by the designated bit become adjacent**
- **Sweep through superposition states**
  - **If necessary, allocate a state to create a pair**
  - **Rotate amplitudes per** $\begin{bmatrix} A'_0 \\ A'_1 \end{bmatrix} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$
- **Sweep through superposition states deleting states with amplitudes < 1E-9**
- **List is left in no particular sorted order**

# Exemplary Code (1)

```
#define LIMIT (3.125e-8/16)
void QubitRegister::Gate(int QubitNum, C A00, C A01, C A10, C A11) {
        QR->Sort2(QubitNum, 1);   // sort – sets FullRangeBits and GroupedBits
        int BothPresent = 0;                // identify existing pairs
        for (int i = 0; i < Num-1; i++)
                if ((Vec[i].State&FullRangeBits) == (Vec[i+1].State&FullRangeBi
                        BothPresent++;
        forcespace(Num*2 - BothPresent);  // allocate memory

        // walk through sorted list rotating pairs
        // to complete pairs, add a state at the end of the list
        int OldNum = Num;
        for (int i = 0; i < OldNum; i++) {
                Superposition *p0 = &Vec[i], *p1;
                if (i+1 < OldNum && (Vec[i].State&FullRangeBits) ==
                                (Vec[i+1].State&FullRangeBits)) {
                        p1 = &Vec[i+1];
                        i++;
                }
                else {
                        p1 = &Vec[Num++];
                        p1->State = p0->State^GroupedBits;
                }
```

Sandia National Laboratories

# Exemplary Code (2)

```
                if ((p0->State&GroupedBits) != 0) {
                        Superposition *p = p0;
                        p0 = p1;
                        p1 = p;
                }

                C t = p0->Amplitude;
                p0->Amplitude = p0->Amplitude*A00 + p1->Amplitude*A01;
                p1->Amplitude =             t*A10 + p1->Amplitude*A11;
        }

        // delete superposition states with amplitude below threshold
        for (int i = 0; i < Num; i++)
                while (i < Num && Vec[i].Amplitude.re*Vec[i].Amplitude.re +
                                Vec[i].Amplitude.im*Vec[i].Amplitude.im < LIMI
                        Vec[i] = Vec[--Num];

}
```

# Outline

- Representation of Qubits
- Non-Entangling Operations
- Entangling Operations
- Measurements
- Addition

# Measurement

- According to Quantum Information Theory, the only measurement necessary is the measurement of a single bit
- More complex measurements (POVMs) can be emulated by ancillae, gate operations, and then single bit measurements
- Multi-bit measurements are equivalent to measuring the bits one at a time and combining the classical results into an integer

Sandia National Laboratories

# Measurement Process

- Measurement Outcome
  - Note: Amplitude squared is probability of a state being detected by measurement
  - Pick a state at random but weighted by probability; outcome is value of designated bit in this state
  - This method needs adjustment for round off errors (later slide)
- Resulting State
  - Delete all states where the designated bit differs from the measurement outcome
  - Renormalize

# Measurement Example

Complex amplitudes | n bits |quantum state>

k = 4

| Complex amplitudes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1/2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $1/2\ e^{i\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $1/2\ e^{i\pi/2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $1/2\ e^{i3\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Measure this bit (second-to-last bit)

# Measurement Example

Complex amplitudes | n bits |quantum state>

| Probability | Complex amplitude | n bits | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| p = .25 | 1/2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| p = .25 | $1/2\ e^{i\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| p = .25 | $1/2\ e^{i\pi/2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| p = .25 | $1/2\ e^{i3\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

① compute ↑ probabilities as norm-squared of amplitudes ($re^2 + im^2$)

← ② pick a state based on probabilities

Measure this bit

③ This is the measurement outcome

# Measurement Example

| | Complex amplitudes | n bits \|quantum state> | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ~~p = .25~~ | ~~1/2~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ |
| ~~p = .25~~ | ~~$1/2\ e^{i\pi/4}$~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~1~~ |
| p = .25 → p = .5 | → $1/\sqrt{2}\ e^{i\pi/2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| p = .25 → p = .5 | → $1/\sqrt{2}\ e^{i3\pi/4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

← ④ delete states incompatible with measurement outcome

⑤ renormalize ↑ probabilities

Measure this bit

# Measurement Notes

- Round off errors and imperfect normalization can cause measurement problems
- Recommended method:
  - Sweep through all states calculating $p_0$ and $p_1$ (probability of measuring a 0 and 1)
  - Note $p_0 + p_1 \approx 1$
  - Use a pseudo random number generator to pick the measurement outcome based on relative probabilities of $p_0$ and $p_1$
  - Delete all states incompatible with the measurement outcome
  - Renormalize

Sandia National Laboratories

# Exemplary Measurement Code

```
int QubitRegister::MeasureBit(int bit) {
        Bits mask = 1 << bit;
        R prob0 = 0.0, prob1 = 0.0;
        for (int i = 0; i < Num; i++) {    // probability of 0 vs. 1
                C *x = &Vec[i].Amplitude;
                R p = x->re*x->re + x->im*x->im;
                if ((Vec[i].State&mask) == 0) prob0 += p;
                else prob1 += p;
        }

        // decide result of measurement
        int rval = R(genrand_real2()*(prob0+prob1)) > prob0 ? 1 : 0;

        // delete states inconsistent with the measurement, normalize others
        R renormal = R(sqrt((prob0+prob1)/(rval == 0 ? prob0 : prob1)));
        for (int i = 0; i < Num; i++) {
                while (i < Num && ((Vec[i].State&mask) == 0) != (rval == 0))
                        Vec[i] = Vec[--Num];      // delete incompatible state
                if (i < Num)
                        Vec[i].Amplitude *= renormal;      // renormalize
        }

        return rval;
}
```

# Outline

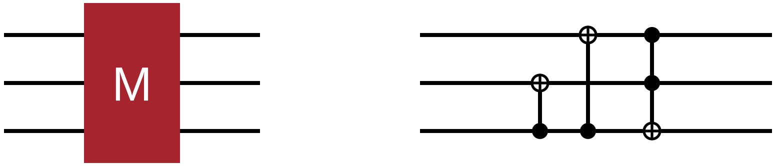
- Representation of Qubits
- Non-Entangling Operations
- Entangling Operations
- Measurements
- Addition

# Addition

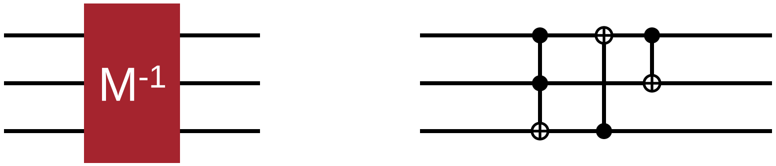
- There are various quantum addition circuits
  - Some options are quantum+quantum and quantum+classical
  - ArXiv:quant-ph/0410184 is a ripple-carry adder
  - ArXiv:quant-ph/0008033 is a qft based adder with no ancilla but other issues
- Let's try the ripple carry adder in ArXiv:quant-ph/0410184

# Ref arXiv quant-ph/0410184

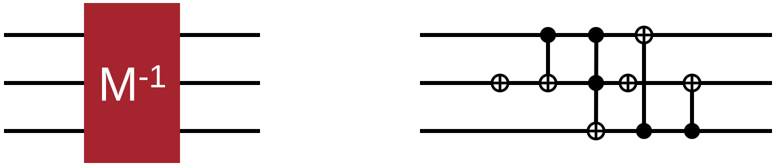
- **Majority Element**




- **Uncompute Majority Element**

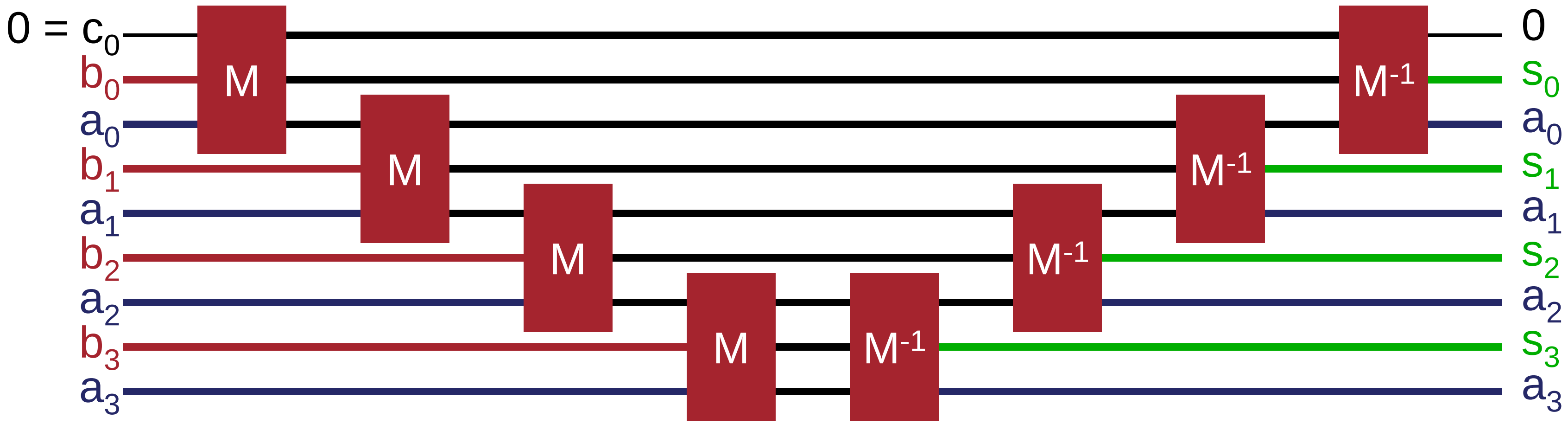



- **Alternate (more parallelism)**



# Ref arXiv:quant-ph/0410184

- **Adder Layout**
  - **Inputs a and b**
  - **Outputs a (unchanged input) and s (sum)**
  - **Also inputs $0 = c_0$ and carry out 0**



# Exemplary Addition Code

```
#define M(X, Y, Z) { Y.CNot(Z); X.CNot(Z); Z.Toffoli(X, Y); }
//#define MI(X, Y, Z) { Z.Toffoli(X, Y); X.CNot(Z); Y.CNot(X); }
#define MI(X, Y, Z) { Y.Not(); Y.CNot(X); Z.Toffoli(X, Y); Y.Not(); \
        X.CNot(Z); Y.CNot(Z); }
int bits = 6, tabsize = 8;
QuantumInt A(bits), C(1);
for (int row = 0; row < tabsize; row++)
        for (int col = 0; col < tabsize; col++) {
                QuantumInt B(bits);
                A = row;
                B = col;
                C = 0;
                M(C, B[0], A[0]);
                for (int i = 1; i < bits; i++) M(A[i-1], B[i], A[i]);
                for (int i = bits-1; i >= 1; i--) MI(A[i-1], B[i], A[i]);
                MI(C, B[0], A[0]);

                int Bx = int(B);
                printf("%d + %d = %d %s\n", row, col, Bx,
                        (row+col)%(1<<bits) != Bx ? " ERR" : "");
        }
```