Parallelization of the QC-lib Quantum Computer Simulator Library*

Ian Glendinning

November 26, 2002

*work carried out jointly with Bernhard Ömer

Outline

- Introduction
- Quantum Computing
- QC-lib
- Parallelization
- Conclusions and Future Work

Introduction

- What is Quantum Computing?
- Why all the Excitement?
- What Makes Quantum Computers Different?
- Experimental Realization
- Quantum Computer Simulators

What is Quantum Computing?

- A fundamentally new mode of information processing that can only be performed by harnessing physical phenomena unique to quantum mechanics
- The basic unit of information in a classical computer, the bit, is a 2-state device that can represent the values 0 or 1
- The quantum analogue of a bit is a 2-state quantum system, e.g. electron spin or photon polarization with respect to some axis
- Such a 2-state quantum system is called a *qubit*

Why all the Excitement?

- Quantum computers can solve hard problems
- E.g. prime factorization - finding the prime factors of an integer
- On conventional computers the execution time of the best known factoring algorithm scales exponentially with the number of digits
- Shor's quantum algorithm for prime factorization scales roughly quadratically with the number of digits, *exponentially* faster!
- New complexity class **BQP**, "Bounded-error, Quantum, Polynomial time" – not just of theoretical interest...

Why all the Excitement?

- The security of public-key cryptosystems such as RSA relies on the difficulty of factorizing very large numbers
- RSA is widely used by banks, using numbers of around 250 digits
- In 1994 a 129 digit number known as RSA129 was factored in eight months using approx. 1600 workstations on the internet
- It would take roughly 800,000 years to factor a 250 digit number, and a 1000 digit number would take 10^{25} years ($\gg$ universe's age)
- Shor's algorithm could factor a 1000 digit number in only a few million steps (step = time taken to factor a one-digit number)

What Makes Quantum Computers Different?

- Quantum Parallelism
  - A qubit can exist not only in the classical 0 and 1 states, but also in a *superposition* of both
  - A classical register with N bits can store *one* integer value in the range 0 to $2^N - 1$, but a quantum register with N qubits can be in a state that's a superposition of all 2^N values
  - Quantum operations act on *all* 2^N values *simultaneously*!
- Where's the catch?
  - Measurement gives only one of the 2^N values, randomly

Experimental Realization

- Technologies being explored include:
  - Ion traps
  - Cavity Quantum Electrodynamics
  - Nuclear Magnetic Resonance (NMR)
  - Optical Interferometers
  - Macroscopic Superconductivity (SQUIDs)
  - Semiconductors

Experimental Realization

- Current State of the Art:
  - Experimental realization of the simplest instance of Shor's algorithm: factoring $N = 15$ (whose prime factors are 3 and 5)
  - Seven spin-1/2 nuclei in a molecule were used as qubits
  - Manipulated with room temperature liquid-state nuclear magnetic resonance (NMR) techniques
  - Nature 20/27 December 2001: Nature **414**, 883-887 (2001)
- Proposals for scalable architectures based on Ion traps, Silicon and most recently SQUIDs (Nov. 2002 - PRL **89**, 197902)

Quantum Computer Simulators

- Why are simulators needed?
  - Hardware not currently available outside research labs
  - Currently the only way to experiment with more than 7 qubits
- Problem
  - The execution time and memory requirements of simulators increase exponentially with the number of qubits
- Parallelization
  - Alleviates the problem, allowing more qubits to be simulated, by providing more computational power and more memory

Quantum Computing

- Quantum Bits and Registers
- Quantum Gates
- Quantum Functions
- Quantum Parallelism
- Quantum Algorithms

Quantum Bits

The states of a qubit corresponding to the classical values 0 and 1 are called the *computational basis states*, and are written $|0\rangle$ and $|1\rangle$.

In general a qubit can be in a *superposition* of these two states:

$$\alpha|0\rangle + \beta|1\rangle$$

where α and β are complex numbers called *amplitudes*, which are subject to the requirement that:

$$|\alpha|^2 + |\beta|^2 = 1$$

Qubits - Measurement

A *measurement* of the state of a qubit always gives either the result 0, with probability $|\alpha|^2$, or the result 1, with probability $|\beta|^2$, so the requirement:

$$|\alpha|^2 + |\beta|^2 = 1$$

expresses the fact that the sum of the probabilities of all possible outcomes must be one. For example, a qubit can be in the state:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

which when measured, gives either the result 0, with probability $|1/\sqrt{2}|^2 = 1/2$, or the result 1, also with probability $1/2$.

Qubits - Vector Representation

The constraint $|\alpha|^2 + |\beta|^2 = 1$ allows us to interpret α and β as components of a two-dimensional complex vector of length 1.

Therefore the general state of a qubit can be represented as a unit vector in a two-dimensional complex vector space.

It is conventional to define the computational basis vectors as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Qubits - Vector Representation

A general qubit state is thus:

$$\alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

subject to the *normalization* condition:

$$|\alpha|^2 + |\beta|^2 = 1$$

Quantum Registers

Two classical bits have four possible states, 00, 01, 10 and 11, and a two qubit system has four corresponding computational basis states, denoted:

$$|0\rangle|0\rangle,\ |0\rangle|1\rangle,\ |1\rangle|0\rangle,\ |1\rangle|1\rangle$$

or more concisely

$$|00\rangle,\ |01\rangle,\ |10\rangle,\ |11\rangle$$

or more even more concisely

$$|0\rangle,\ |1\rangle,\ |2\rangle,\ |3\rangle$$

Quantum Registers

A pair of qubits can exist in superpositions of these states, and a general two-qubit state can be written:

$$\alpha_0|0\rangle + \alpha_1|1\rangle + \alpha_2|2\rangle + \alpha_3|3\rangle = \sum_{i=0}^{3} \alpha_i|i\rangle$$

where the α_n are complex amplitudes, subject to the normalization condition that

$$\sum_{i=0}^{3} |\alpha_i|^2 = 1$$

A measurement always gives one of the values 0, 1, 2 or 3, and the probability of measuring the result i is $|\alpha_i|^2$.

Vector Representation

Again the α_i can be interpreted as components of a complex vector, this time in *four dimensions*. The computational basis vectors are:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Vector Representation

A general two-qubit state is of the form:

$$\sum_{i=0}^{3} \alpha_i |i\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$$

subject to the normalization condition:

$$\sum_{i=0}^{3} |\alpha_i|^2 = 1$$

N-Qubit Registers

An n-qubit register has 2^n computational basis states $|i\rangle$, where $0 \leq i \leq 2^n - 1$, and it can be in a superposition of these states:

$$\sum_{i=0}^{2^n-1} \alpha_i |i\rangle$$

where $|\alpha_i|^2$ is the probability that measurement will give the value i.

This state can be represented as a complex vector in 2^n dimensions, with the normalization condition:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$$

Entanglement

Except in special cases, you can't just list the state of each qubit. E.g. in the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ neither qubit has a definite value that is independent of the other. Such states are said to be *entangled*.

In general 2^n complex numbers are needed to specify the state of an n-qubit register. For $n = 500$ this number is larger than the estimated number of atoms in the universe!

Nature manipulates huge quantities of data, even for systems of a few hundred atoms. The aim of quantum computing is to exploit this!

Quantum Gates

- Classical computer circuits are built from logic gates which operate on bits and registers
- By analogy we can define quantum gates which operate on qubits and quantum registers

The Quantum NOT Gate

The only non-trivial classical single-bit gate is the NOT gate, defined by its 'truth table' in which $0 \to 1$ and $1 \to 0$.

We can define an analagous quantum NOT gate, conventionally written as an operator X, such that $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$

This does not tell us what happens to superpositions of the states $|0\rangle$ and $|1\rangle$. In fact, the quantum NOT gate acts *linearly*, i.e.:

$$\begin{aligned} X(\alpha|0\rangle + \beta|1\rangle) &= X\alpha|0\rangle + X\beta|1\rangle \\ &= \alpha X|0\rangle + \beta X|1\rangle \\ &= \alpha|1\rangle + \beta|0\rangle \end{aligned}$$

Matrix Representation

Linearity is a general property of quantum mechanics, and means that quantum gates can be conveniently represented in matrix form:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In vector notation, for the state $\alpha|0\rangle + \beta|1\rangle$ we have

$$X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$

Notice that X preserves the normalization. This is a property of *all* quantum gates, and turns out to be the *only* constraint on them.

The Hadamard Gate

In contrast to the classical case, there are many non-trivial single-qubit gates. One of the most useful is the *Hadamard* gate

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Acting on the Basis states $|0\rangle$ and $|1\rangle$:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

and

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The Controlled-NOT Gate

The prototypical multi-qubit gate is the *controlled*-NOT or CNOT gate.

It has two inputs, known as the *control* and *target* qubits, and two outputs.

If the control qubit is set to 0, the target qubit is unchanged, and if the control qubit is set to 1, the target qubit is flipped ($|c\rangle|t\rangle$):

$$|00\rangle \to |00\rangle; \ |01\rangle \to |01\rangle; \ |10\rangle \to |11\rangle; \ |11\rangle \to |10\rangle$$

CNOT is a generalization of the classical XOR gate, since its action may be summarized as $|x\rangle|y\rangle \to |x\rangle|y \oplus x\rangle$, where $\oplus$ is addition modulo two, which is the same as XOR.

Matrix Representation

The matrix representation of CNOT is:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Every n-qubit gate has a $2^n \times 2^n$ complex matrix representation, and as for single qubit gates it must preserve the normalization of states.

It can be shown that this implies that all quantum gates have an inverse, i.e that quantum computations are *reversible*.

Quantum Functions

Suppose $f(x) : \{0,1\} \to \{0,1\}$ is a function with a one-bit domain and range.

We can compute this value using a two-bit quantum computer which starts in the state $|x, y\rangle$, using a function U_f, defined as:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

Note that even if f has no inverse, keeping a copy of x in the result register means that U_f is invertible, as required of any quantum gate.

U_f can easily be generalised to domains and ranges of more than one qubit, and is an important building block for quantum algorithms.

Quantum Parallelism

Consider the action of U_f to the superposition state $(|0\rangle + |1\rangle)/\sqrt{2}$, which can be created with a Hadamard gate acting on $|0\rangle$:

$$U_f \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) |0\rangle = U_f \left(\frac{|0\rangle|0\rangle + |1\rangle|0\rangle}{\sqrt{2}} \right) = \frac{|0\rangle|f(0)\rangle + |1\rangle|f(1)\rangle}{\sqrt{2}}$$

It is almost as if we have evaluated $f(x)$ for two values of x simultaneously! This is *quantum parallelism*.

This procedure can easily be generalized to functions of an arbitrary number of qubits. We first create a superposition state using a *Hadamard transform*, which is just n Hadamard gates acting in parallel on n qubits.

Quantum Parallelism

For example, for $n = 2$, acting on $|00\rangle$ it gives:

$$H|0\rangle H|0\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}$$

More generally, the result of the Hadmard transform on n qubits initially all in the $|0\rangle$ state is:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

which is an equal superposition of all 2^n basis states. By applying a linear number of gates (n), we have generated a state with an exponential number of terms!

Quantum Parallelism

Applying a generalized version of U_f with an n-qubit input register (x) to this state gives:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle$$

So in some sense we have computed all 2^n possible values of $f(x)$ simultaneously!

However, measurement of the above state would only give $f(x)$ for a single random value of x, which a classical computer can do easily.

Quantum Algorithms

- Quantum computation requires more than just quantum parallelism to be useful
- The solution is to use destructive interference to cancel out all components of the superposition except the one corresponding to the answer to a computation involving more than one value of $f(x)$. An example of destructive interference is:

$$H\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)=\left(\frac{|0\rangle+|1\rangle}{2}\right)+\left(\frac{|0\rangle-|1\rangle}{2}\right)=|0\rangle$$

- The combination of *exponential parallelism* and *interference* is what makes quantum computation powerful

Categories of Quantum Algorithms

- Fourier transform (QFT)
  - Period-finding, Factoring, Discrete logarithm, etc.
  - All special cases of the *Hidden subgroup problem*, which encompases all known 'exponentially fast' QFT applications
- Search
  - Grover, Minimum, Quantum counting, speeding up solution of NP-complete problems
- Simulation of quantum systems

QC-lib

- A C++ library for the simulation of quantum computers at an abstract functional level, written by Bernhard Ömer
- QC-lib's main features are:
  - Basis vectors of arbitrary length (not limited to word length)
  - Efficient representation of quantum states using hash tables
  - Nesting of substates and arbitrary combinations of qubits
  - Composition and tensor product of operators (gates)
  - Easy extendibility by class inheritance

QC-lib Classes

The top-level classes in QC-lib are:

- **bitvec** - arbitrary length bit vectors which represent basis states
- **term** - a basis vector with a complex amplitude
- **termlist** - internal representation of a quantum state
- **quState** - user class for quantum states
- **opOperator** - user class for quantum operators

Class bitvec

- Data structure

```
class bitvec {
  int len;
  union {
    word *pvec;
    word vec;
  };
```

- Member functions
  - (contructor, destructor), assignment, length, getbits, setbits, concatenation, bitwise logical operations etc...

Class term

- Data structure

```
class term {
  bitvec _vect;
  complx _ampl;
```

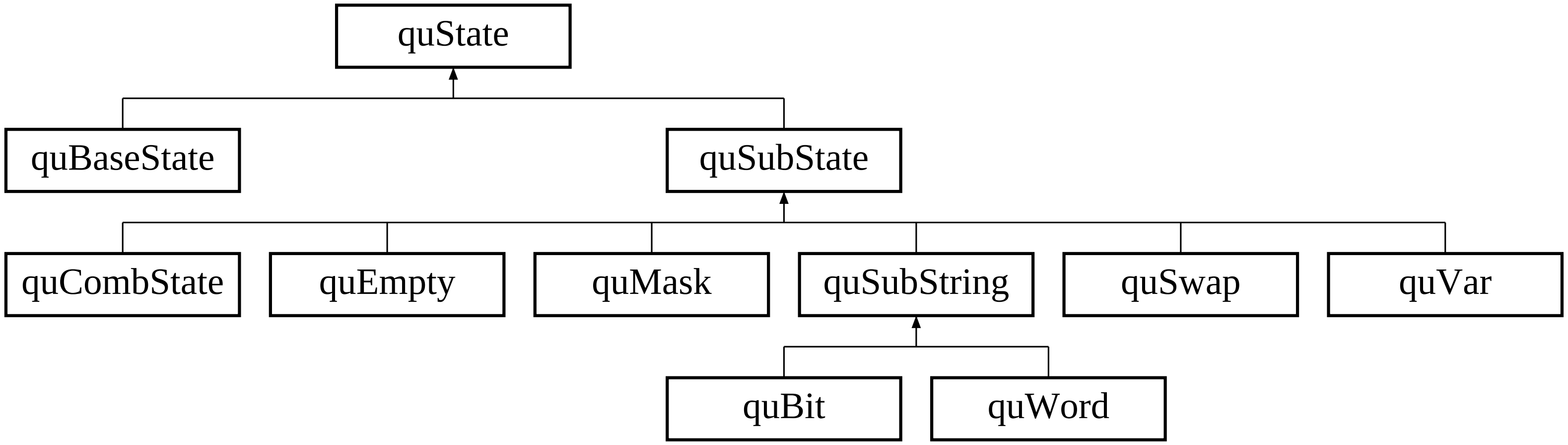
- Member functions
  - assignment, no. of bits, get and set vector and its amplitude

Class termlist

- Data structure
  - Linear array in combination with a hash table
  - The array and hash table are dynamically doubled in size if the array fills up
  - Only terms with non-zero amplitudes are stored
- Member functions
  - assignment, no. of terms, add terms, i^{th} term from list, clear list, and a couple of others

Class quState

- Inheritance Diagram



- `quBaseState` represents the overall state of the quantum memory from which subregisters `quSubState` can be allocated

Class quBaseState

- Data structure

```
class quBaseState : public quState {
  termlist *pterms;
  termlist *ptermbuf;
```

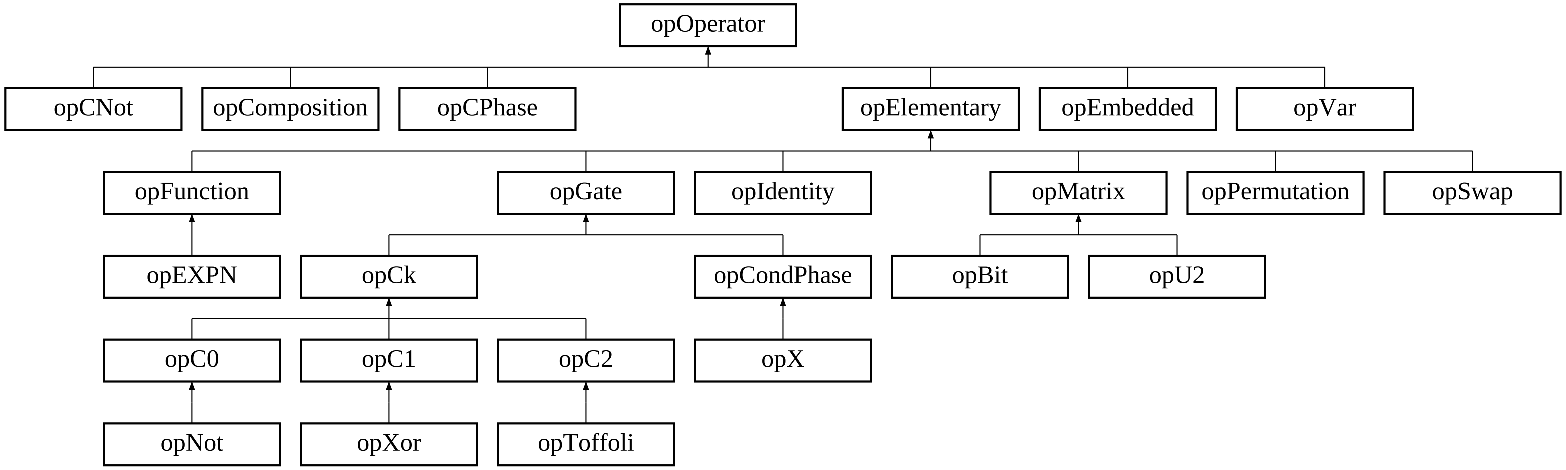
- One termlist for the current state and one to accumulate the result of an operation on it

Operators

- Any n-qubit operator can be represented as a $2^n \times 2^n$ complex matrix, and class `opMatrix` implements this
- However, most operators are simpler, e.g. only working on a few qubits, and QC-lib provides classes for a number of special cases

Class opOperator

- Inheritance Diagram



Example C++ Progam using QC-lib

```
// Implement Hadamard transform

quBaseState qs(20);              // allocate 20-qubit register
opBit H(1,1,1,-1,sqrt(0.5)); // define Hadamard gate H

// loop over qubits in register
for(int i=0; i<qs.mapbits(); i++) {
  quBit qbit(i,qs);              // create one-qubit substate
  H.apply(qbit);                 // apply H to the qubit
}
```

QCL

- QCL (Quantum Computation Language) is a high-level architecture-independent procedural programming language for quantum computers
- The interpreter `qcl` uses QC-lib as a back end, so QCL programs can be developed and run without a physical quantum computer
- Syntax derived from classical languages like C or Pascal
- The same Hadamard example in QCL:

```
qureg qs[20];
H(qs);
```

Example QCL Progam - DFT (Coppersmith)

```
operator dft(qureg q) { // main operator
  const n=#q;             // set n to length of input
  int i; int j;           // declare loop counters
  for i=1 to n {
    for j=1 to i-1 {      // apply conditional phase gates
      V(pi/2^(i-j),q[n-i] & q[n-j]);
//      if q[n-i] and q[n-j] { Phase(pi/2^(i-j)); }
    }
    H(q[n-i]);            // qubit rotation
  }
  flip(q);                // swap bit order of the output
}
```

Parallelization

- Strategy:
  - Distribute the representation of the quantum memory
  - Each processor stores a subset of the terms
  - SPMD - each processor running the same program
  - Operators are applied only to local terms (owner computes)
  - The result of the local operations in general includes terms that are not owned by the processor, which need to be communicated to their owning processors
  - Implement communication with MPI

Data Distribution Scheme

Least significant bits determine processor. E.g. for four processors:

Basis State	Processor
$\|0\mathbf{00}\rangle$	0
$\|0\mathbf{01}\rangle$	1
$\|0\mathbf{10}\rangle$	2
$\|0\mathbf{11}\rangle$	3
$\|1\mathbf{00}\rangle$	0
$\|1\mathbf{01}\rangle$	1
$\|1\mathbf{10}\rangle$	2
$\|1\mathbf{11}\rangle$	3

Single-Qubit Operators

Consider a 3-qubit register and four processors, with the register initially in the state $|000\rangle$, which is stored on processor 0:

$$\begin{aligned}(H|0\rangle)|00\rangle &= (1/\sqrt{2})(|0\rangle + |1\rangle)|00\rangle \\ &= (1/\sqrt{2})(|000\rangle + |100\rangle)\end{aligned}$$

Both new states are still on processor 0 - no communication needed.

$$\begin{aligned}|0\rangle(H|0\rangle)|0\rangle &= |0\rangle(1/\sqrt{2})(|0\rangle + |1\rangle)|0\rangle \\ &= (1/\sqrt{2})(|000\rangle + |010\rangle)\end{aligned}$$

$|010\rangle$ is on processor 2, so it's amplitude needs communicating.

Operator Invocation mechanism

- All operators have an `apply()` member function that takes a `quState` argument
- `apply()` is implemented differently for various operator subclasses
- The largest subclass is `opElementary`, all of whose subclasses apart from `opIdentity` use the same `apply()` function
  - `opElementary::apply()` uses three member functions in class `quState`, namely `opbegin()`, `opadd()` and `opend()`

Operator Invocation mechanism

```
void quBaseState::opbegin() {
  ptermbuf->clear();
}
void quBaseState::opadd(const bitvec& v,const complx& z) {
  ptermbuf->add(v,z);
}
void quBaseState::opend() {
  termlist *pl;
  pl=pterms; pterms=ptermbuf; ptermbuf=pl;
}
```

Communication Structure of One-Qubit Operators

- For each term in the initial state, at most two terms are created, one local, and one remotely owned
- For each processor, all the remotely owned terms are owned by one other processor
- Symmetrically, the remote processor potentially creates terms that are owned by the local one
- Communication is needed between disjoint pairs of processors

Parallel Implementation of One-Qubit Operators

- Accumulate local and remote terms separately, then exchange remote terms with remote processor, and accumulate the received terms
- Representation of the distributed quantum memory encapsulated in class `quBaseState`

Changes to Class `quBaseState`

- `quBaseState` constructor allocates new `mp_ptermbuf` termlist to list of remote terms
- `reset()` changed to initialize quantum memory to $|0\rangle$ only on process zero, its owner. Other processes initially own no terms
- New `dump()` function to print out the state of quantum memory
- `opbegin()` takes new parameter defining the bit to be operated on, and sets new variable `mp_remote_proc` to remote process id
- New member function `mp_owner_proc()` to return the owner processor of a basis state (bitvec object)

New Version of opadd()

```
void quBaseState::opadd(const bitvec& v,const complx& z) {
  if (mp_owner_proc(v) == mp_my_proc) {
    ptermbuf->add(v,z);
  }
  else {
    mp_ptermbuf->add(v,z);
  }
}
```

New Version of opend()

Calls new function `mp_sendrecv` to exchange lists of remote terms

```
void quBaseState::opend() {
  termlist *pl;
  if (mp_remote_proc != mp_my_proc) {
    mp_sendrecv(mp_ptermbuf, mp_remote_proc,
                ptermbuf, mp_remote_proc);
  }
  pl=pterms; pterms=ptermbuf; ptermbuf=pl;
}
```

Issues Encountered

- C++ classes were sometimes a hinderance to efficient use of MPI
- E.g. an array of complex number objects cannot be portably sent or received, as the address of the data is hidden by the object
- Similarly, class `termlist` internally stores terms in a linear array, but the terms are objects
- In this first implementation, portability was achieved at the cost of copying the data out of the objects into a buffer
- It would be more efficient to re-implement `termlist` with a list not using objects, and a `mp_sendrecv` member function

Performance Measurements

C++ Hadamard example program

Qubits	No. Procs	Time (s)
22	1	110.0
22	2	57.6
22	4	30.8
22	8	18.0
19	1	14.8
19	2	9.3
19	4	7.6
19	8	6.0

Storage Requirements

- 64 bytes are needed per term for the sequential code
- So on a single processor with 2GB of memory, assuming 1GB ($= 2^{30}$) available, if all possible basis vectors had non-zero amplitudes, the maximum number of qubits that can be simulated is about 24
- With 16 nodes each with 2GB, the parallel version could represent up to four more qubits, i.e. 30, though there is an extra memory overhead so it's probably more like 3 more
- However, not all algorithms use such a large number of non-zero terms

Conclusions

- Hard to understand C++ code without documentation, even with access to the author!
  - Doxygen tool helped - class hierarchy and code browsing
- C++ classes can make efficient use of MPI harder than for C or Fortran
- A parallel version of the general single-qubit operator has been implemented
- Performance is promising
- Much more to do!

Future Work

- Static control of distributed qubits by programmer
- Multi-qubit operators
- Optimized parallel `termlist` class with `mp_sendrecv`
- Implement Shor's and Grover's algorithms
- Dynamic redistribution of qubits

Further Information

- http://www.stack.nl/~dimitri/doxygen/
- http://www.qubit.org/
- http://arXiv.org/archive/quant-ph
- http://tph.tuwien.ac.at/~oemer/qcl.html
- http://www.vcpc.univie.ac.at/~ian/hotlist/