Parallelization of the QC-lib Quantum Computer Simulator Library

Ian Glendinning and Bernhard Ömer

October 21, 2003

Outline

- Introduction
- Quantum Bits, Registers and Gates
- QC-lib
- Parallelization
- Performance Measurements
- Related Work
- Conclusion

Introduction

- What is a Quantum Computer?
- What Makes Quantum Computers Different?
- Quantum Algorithms
- Quantum Computer Simulators

What is a quantum computer?

- A device that processes information using physical phenomena unique to quantum mechanics
- What makes quantum computers so exciting?
  - They can solve hard problems
  - The execution time of the best known classical algorithm for prime factorization scales exponentially with the no. of digits
  - Shor's quantum algorithm for prime factorization scales roughly quadratically, *exponentially faster!*

What Makes Quantum Computers Different?

- The quantum analogue of a bit is a two-state quantum system such as an electron's spin or a photon's polarization, a *qubit*
- A qubit can exist not only in the classical 0 and 1 states, but also in a *superposition* of both
- A quantum register with N qubits can be in a state that's a superposition of all values in the range 0 to $2^N - 1$
- Quantum operations act on *all* 2^N values *simultaneously*! This is known as quantum parallelism
- However...
  - Measurement gives only one of the 2^N values, at random

Quantum Algorithms

- The probabilities of measuring the different values can be manipulated by operating on a quantum register with *quantum gates*, the analogue of logic gates
- Quantum algorithms consist of sequences of quantum gate operations and optionally measurements
- Algorithms exist that are able to exploit quantum parallelism, and leave an output register in a state where the probability of obtaining the answer to the problem is very close to one, giving an advantage over some classical algorithms

Quantum Computer Simulators

- Why are quantum computer simulators needed?
  - Hardware not currently available outside research labs
  - Currently the only way to experiment with more than 7 qubits
  - Debugging programs: examination of the quantum state
- 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 Bits, Registers and Gates

- Representation of Qubits, Registers and Gates
- Conventions and Notation
  - Qubits
  - Two-Qubit Registers
- Examples of One and Two-Qubit Gates
  - The Hadamard Gate
  - The Controlled-NOT Gate

Representation of Qubits, Registers and Gates

- The state of a qubit can be represented as a two-dimensional complex unit vector
- The state of an n-qubit register can be represented as an 2^n-dimensional complex unit vector
- Any n-qubit gate (operator) can be represented as a $2^n \times 2^n$ unitary matrix, i.e. a complex matrix U with the property that $U^\dagger U = I$
- The operation of a gate on a quantum register is implemented by matrix multiplication

Qubits

The qubit states corresponding to classical values 0 and 1 are called the *computational basis vectors* and are conventionally defined as:

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

So a general qubit state can be written:

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

subject to the normalization condition $|\alpha|^2 + |\beta|^2 = 1$, where $\alpha, \beta \in \mathbf{C}$
A *measurement* of the state of a qubit always results in a value of either $|0\rangle$ or $|1\rangle$, with a probability of $|\alpha|^2$ for $|0\rangle$ and $|\beta|^2$ for $|1\rangle$

Two-Qubit Registers

The state of a two qubit register can be represented as a four-dimensional complex unit vector.

The computational basis vectors correspond to the classical binary values 00, 01, 10 and 11, and are conventionally defined as:

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

A general two-qubit state is a linear combination of the basis vectors, subject to the normalization constraint that it has length 1.

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 ($|c,t\rangle$):

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

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

QC-lib

QC-lib is a C++ library for the simulation of quantum computers at an abstract functional level. Its 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

It can be used as the back end for an interpreter for the QCL Quantum Computation Language

QC-lib Classes

- **bitvec** - arbitrary length bit vectors which represent basis states
- **term** - a basis vector with a complex amplitude
- **termlist** - the internal representation of a quantum state
  - Only terms with non-zero amplitudes are stored
  - Linear array in combination with a hash table
  - The array and hash table are dynamically doubled in size if the array fills up

QC-lib Classes

- **quState** - user class for quantum states
  - Subclasses `quBaseState` to store actual state information and `quSubState` to allow subregisters to be allocated
  - `quBaseState` contains two termlists, one for the current state and one to accumulate the result of an operation on it
- **opOperator** - user class for quantum operators
  - Subclass `opMatrix` represents an n-qubit operator as a $2^n \times 2^n$ complex matrix, storing the non-zero elements of each row in an array of lists
  - But most operators are simpler, e.g. only working on a few qubits, and **opOperator** has subclasses for a number of special cases, e.g. general single-qubit operations and CNOT

Example C++ program using QC-lib

```
void H(quState& qs) {      // Implement Hadamard transform
  opBit H1(1,1,1,-1,sqrt(0.5)); // define Hadamard gate H
  for(int i=0; i<qs.mapbits(); i++) { // loop over qubits
    quBit qbit(i,qs);           // create one-qubit substate
    H1.apply(qbit);             // apply H to the qubit
  }
}
quBaseState qs(20); // allocate 20-qubit register qs
H(qs);              // apply the Hadamard transform to qs
```

Parallelization of QC-lib

- The parallelization is being carried out using the MPI message passing interface
- Parallelization Strategy
  - Distribute the representation of the quantum memory
  - Each processor stores a subset of the terms
  - Each processor runs the same program, and 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

Data Distribution Scheme

The data distribution scheme is that on 2^n processors, the n least significant bits of a basis state form a distribution key, which determines which processor its corresponding term is stored on.

For example, the processor allocation of basis states for a four-qubit register on four processors P_0–P_3 is:

P_0	$\|0000\rangle$, $\|0100\rangle$, $\|1000\rangle$, $\|1100\rangle$
P_1	$\|0001\rangle$, $\|0101\rangle$, $\|1001\rangle$, $\|1101\rangle$
P_2	$\|0010\rangle$, $\|0110\rangle$, $\|1010\rangle$, $\|1110\rangle$
P_3	$\|0011\rangle$, $\|0111\rangle$, $\|1011\rangle$, $\|1111\rangle$

Single Qubit Operators

Consider a general single qubit operator

$$U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$$

operating on a single qubit with state

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

For simplicity, assume that there are just two processors, and the qubit in question is the least significant one, so it determines the data distribution.

Communication Pattern for 1-Qubit Operators

After the operation of U locally on each processor, terms are created that are not owned by the processor, so communication is needed:

$$\begin{array}{c} P_0 \\ P_1 \end{array} \boxed{\begin{array}{c} \alpha|0\rangle \\ \hline \beta|1\rangle \end{array}} \xrightarrow{U} \boxed{\begin{array}{c} \alpha U|0\rangle \\ \hline \beta U|1\rangle \end{array}} = \boxed{\begin{array}{c} \alpha u_{11}|0\rangle + \alpha u_{21}|1\rangle \\ \hline \beta u_{12}|0\rangle + \beta u_{22}|1\rangle \end{array}} \xrightarrow{\text{Comm}} \boxed{\begin{array}{c} (\alpha u_{11} + \beta u_{12})|0\rangle \\ \hline (\alpha u_{21} + \beta u_{22})|1\rangle \end{array}}$$

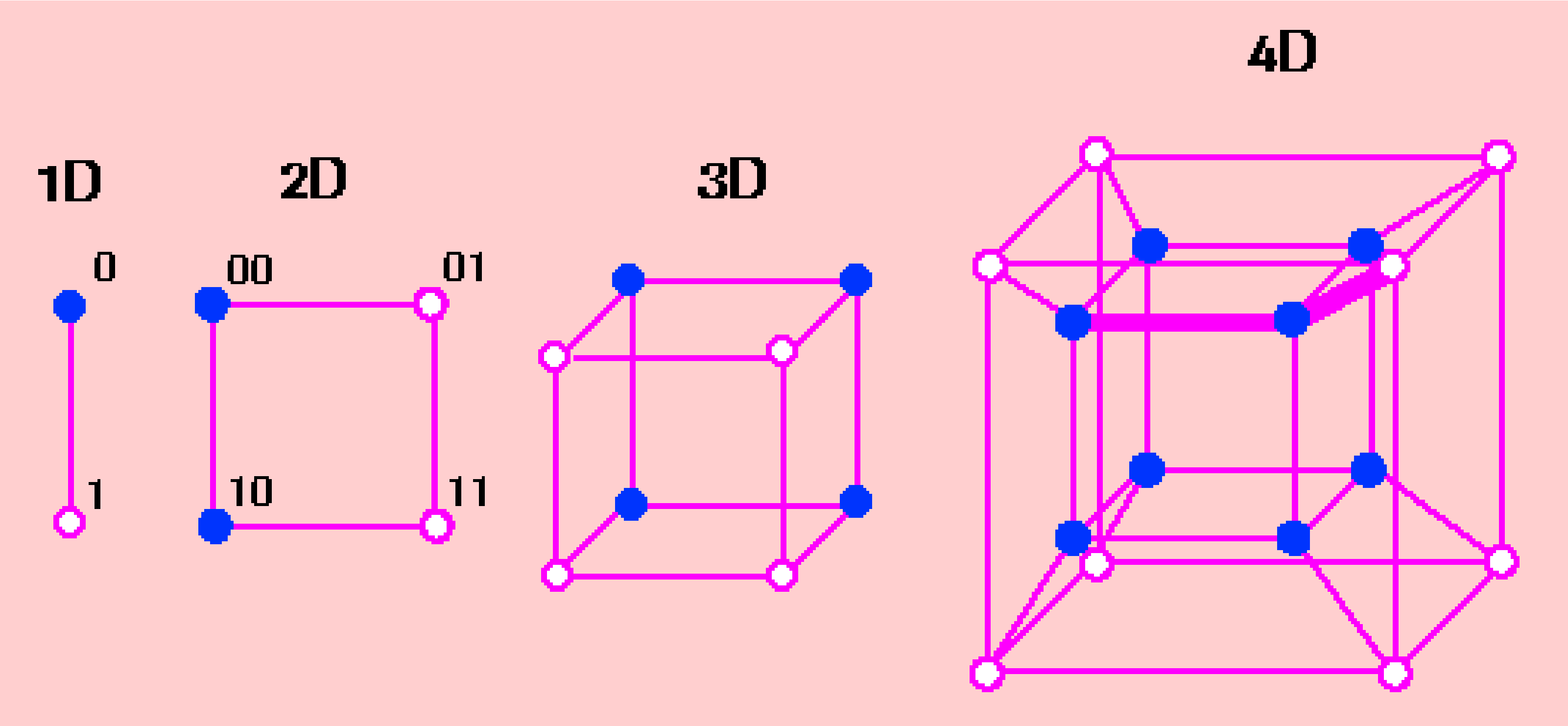
When a single qubit operator is applied to the ith qubit of a register, it is applied to the ith qubit of every term in the superposition state of the register, leaving the other qubits in each term unchanged. E.g.

$$\begin{array}{c} P_0 \\ P_1 \end{array} \boxed{\begin{array}{c} \alpha U|010\rangle \\ \hline \beta U|101\rangle \end{array}} = \boxed{\begin{array}{c} \alpha u_{11}|010\rangle + \alpha u_{21}|011\rangle \\ \hline \beta u_{12}|100\rangle + \beta u_{22}|101\rangle \end{array}} \xrightarrow{\text{Comm}} \boxed{\begin{array}{c} \alpha u_{11}|010\rangle + \beta u_{12}|100\rangle \\ \hline \alpha u_{21}|011\rangle + \beta u_{22}|101\rangle \end{array}}$$

Communication Pattern for 1-Qubit Operators

- For each term in the initial state, at most two terms are created, and if the qubit is in the distribution key, one new term is 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
- The basis vectors of an n-qubit register define the corners of an n-dimensional hypercube, and an operation on the ith qubit generates data movement in its ith dimension

Hypercubic Communication Pattern



Parallelization of Single-Qubit Operators

The representation of the distributed quantum memory has been encapsulated in class `quBaseState`, which has an extra `termlist` to accumulate remote terms. The action of an operator is:

1. Apply the operator locally to each term, accumulating the resulting local and remote terms separately
2. Exchange remote terms with remote processor using `MPI_Sendrecv()`
3. Merge received terms into local term buffer
4. Swap local term buffer with the `termlist` for current state so new terms become the current state, as in sequential version

Current Status

- Although the CNOT gate is a two qubit gate, it only modifies a single qubit, so we were able to use the same parallelization strategy
- The general single qubit operator and CNOT have been implemented - together universal for quantum computation
- Not necessarily the most efficient way to implement other operations - optimized versions of other operators are possible

Performance Measurements

Performance measurements made on a Beowulf cluster with 16 nodes, connected by Gigabit Ethernet, each with:

- A 3.06 GHz Pentium 4 processor
- 2 GByte of 266 MHz memory
- Linux 2.4.19 operating system

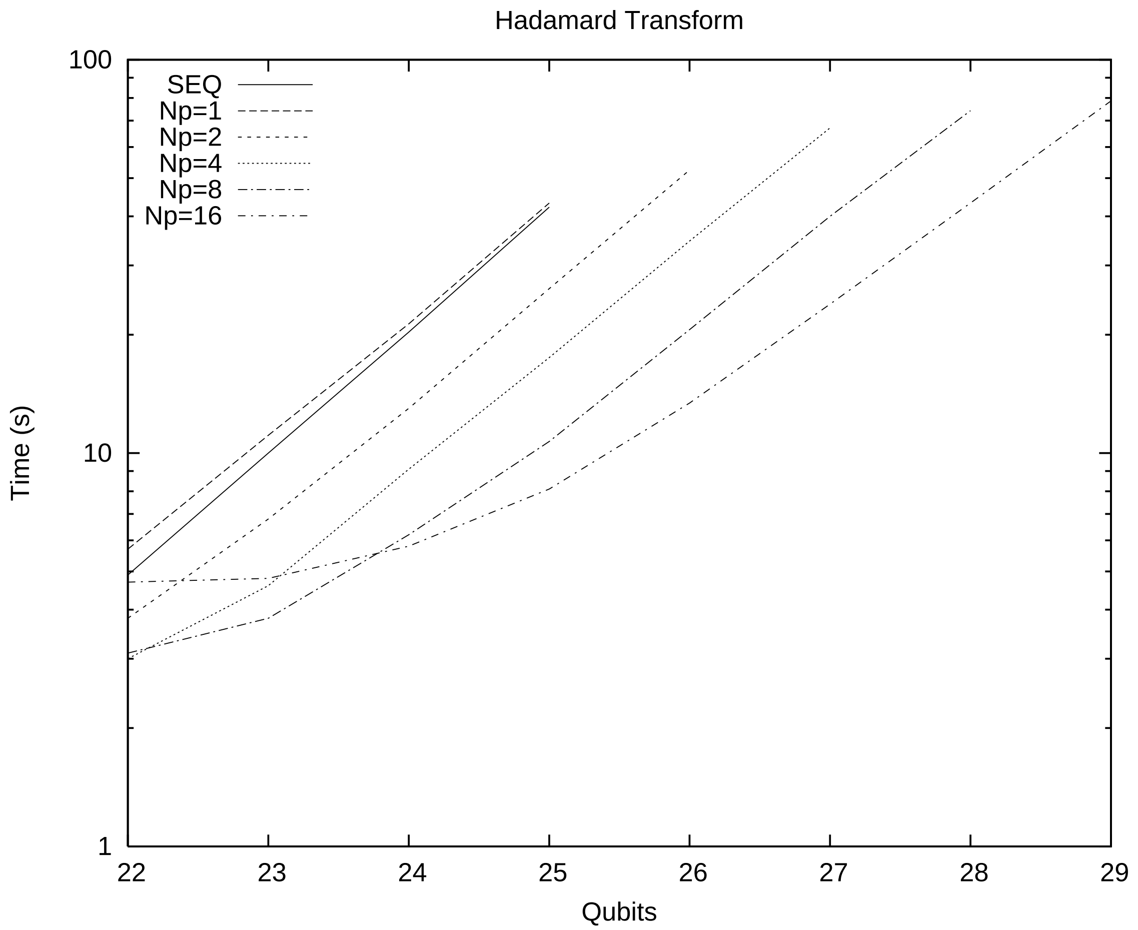
Measurements were made for two test cases:

- Hadamard Transform
  - Hadamard gate applied to every qubit in a register
- Grover's Algorithm
  - Unstructured search in time $O(\sqrt{N})$, classically $O(N)$

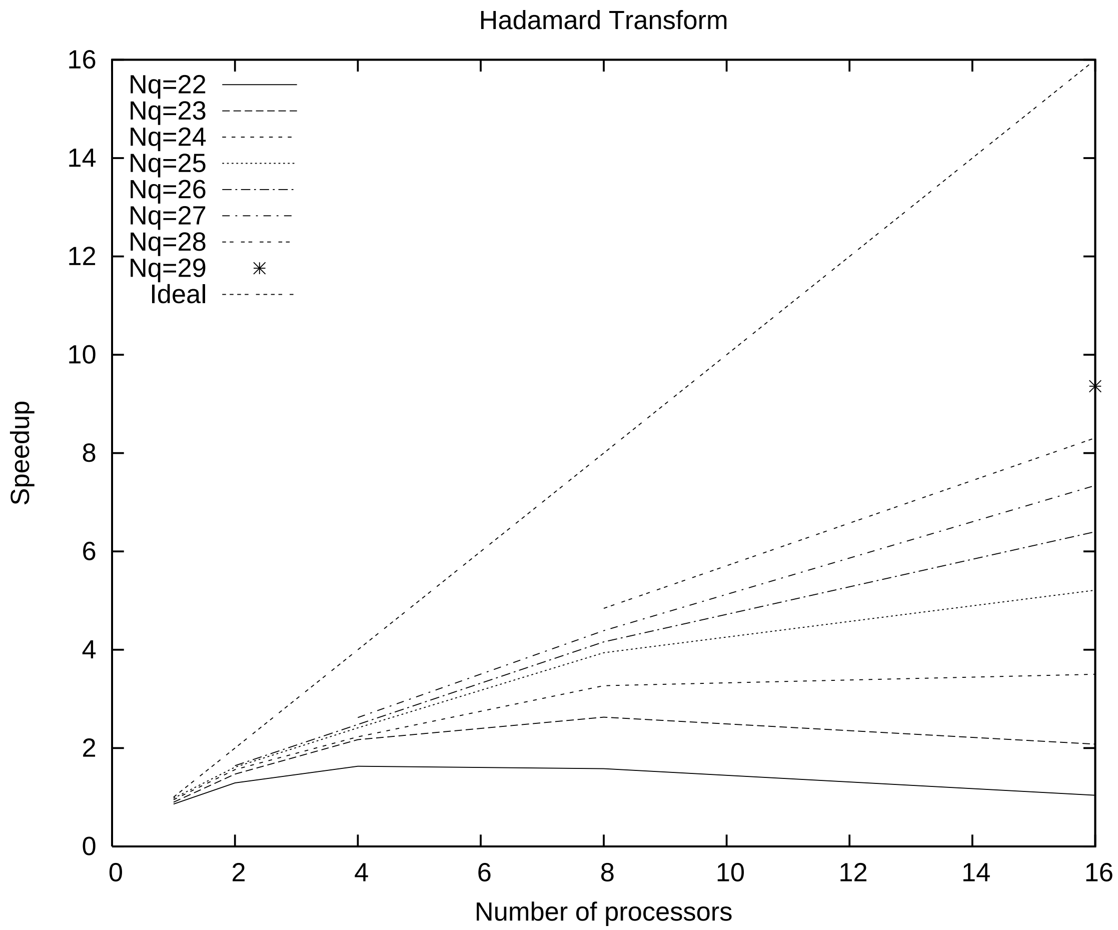
VCPC

EUROPEAN CENTRE FOR PARALLEL COMPUTING AT VIENNA

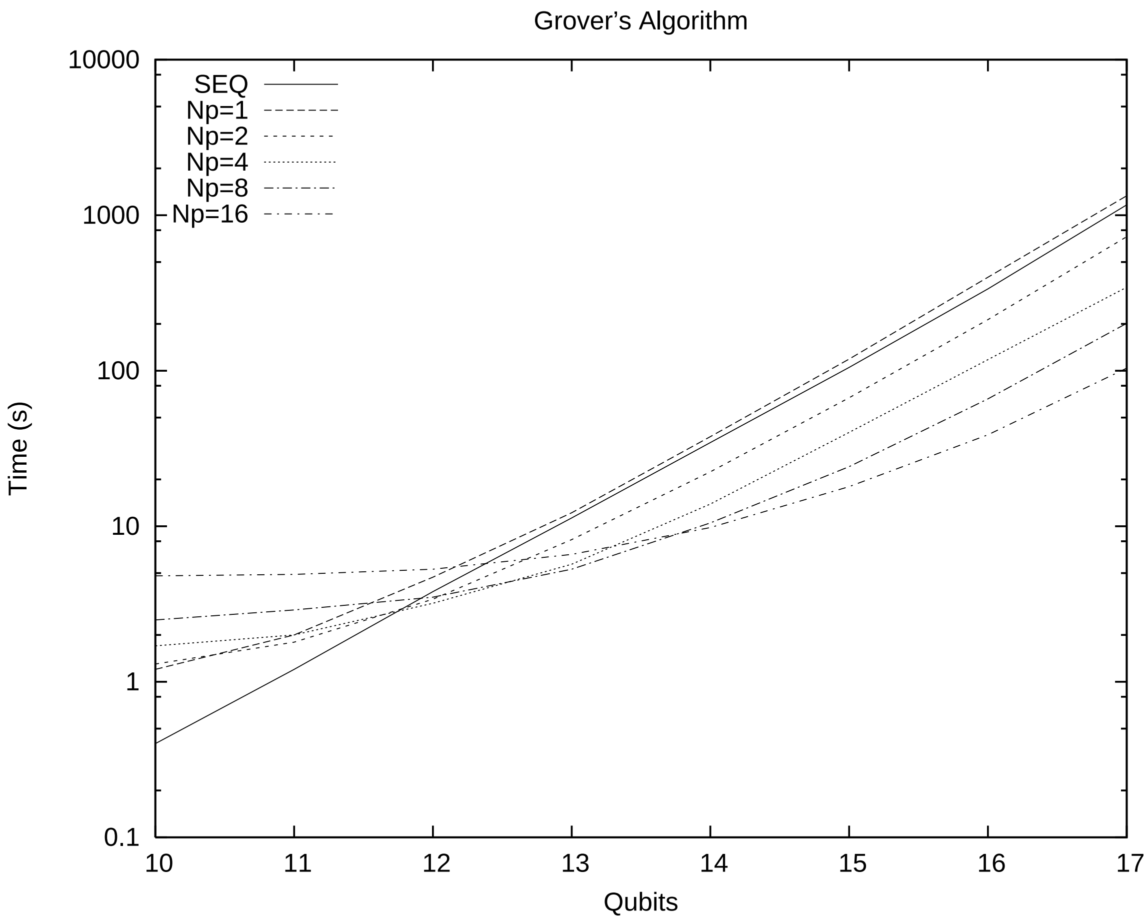
Hadamard Transform



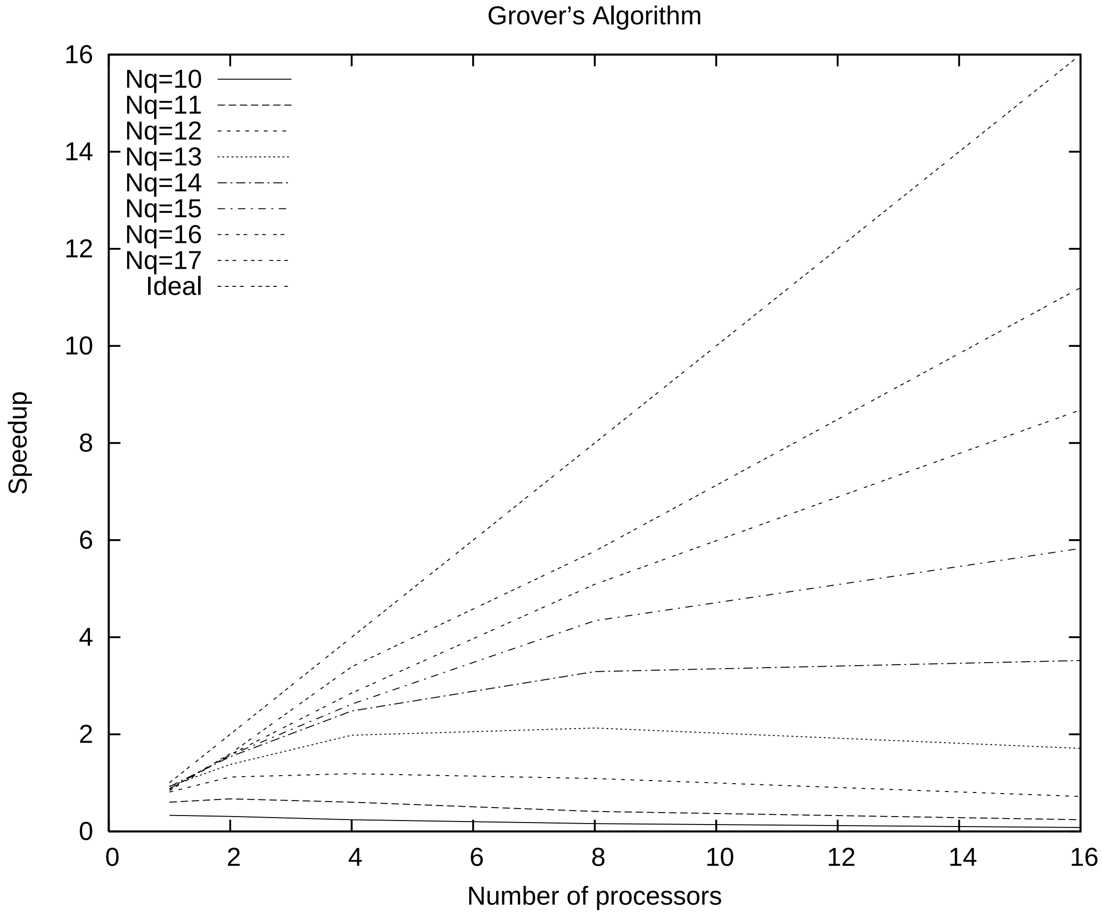
Hadamard Transform



Grover's Algorithm



Grover's Algorithm



Summary of Performance Measurements

- Hadamard Transform
  - Speedup of 9.4 on 16 processors for largest problem size (29 qubits)
  - Up to 25 qubits on 1 processor (64 bytes per term)
  - Up to 29 qubits on 16 processors
- Grover's Algorithm
  - Speedup of 11.2 on 16 processors for largest problem size (17 qubits)

Related Work

There are many quantum computer simulators but we are only aware of two of them that have been implemented for parallel systems:

- Obenland and Despain (1998)
  - Runs on distributed-memory parallel computers, like our simulator, but is based on a physical model of a particular experimental realization of a quantum computer
- Niwa, Matsumoto and Imai (2002)
  - Like our simulator, does not depend on any physical model, but runs on shared-memory multi-computers rather than distributed-memory ones

Conclusion

- Sufficient functionality of QC-lib has been parallelized to implement simulation of universal quantum computation
- Implemented the Hadamard transform and Grover's algorithm
- Promising speedups obtained, more qubits simulated
- Future work
  - Static control of distribution of qubits by the programmer
  - Parallel implementation of more operators
  - Investigate performance of more algorithms
  - Dynamic redistribution of qubits and load-balancing

Conclusion

- FP6 proposal: Quantum Programming and Simulation Project
  - Objectives
    * Further Development of the Language QCL
    * Simulation of Quantum Computers
    * Quantum computing paradigms and their abstract models
    * Transforming, optimizing and mapping quantum & classical programs onto quantum & classical architectures
  - Preliminary Consortium
    * 8 groups from Austria, France, Germany, Poland, Romania, England
- Further information from http://www.vcpc.univie.ac.at/qc/