QOperAv, a Code Generator for Generating Quantum Circuits for Evaluating Certain Quantum Operator Averages

Robert R. Tucci
P.O. Box 226
Bedford, MA 01730
tucci@ar-tiste.com

October 26, 2010

Abstract

This paper introduces QOperAv v1.5, a Java application available for free. (Source code included in the distribution.) QOperAv is a "code generator" for generating quantum circuits. The quantum circuits generated by QOperAv can be used to evaluate with polynomial efficiency the average of $f(A)$ for some simple (that is, computable with polynomial efficiency) function f and a Hermitian operator A, provided that we know how to compile $\exp(iA)$ with polynomial efficiency. QOperAv implements an algorithm described in earlier papers, that combines various standard techniques such as quantum phase estimation and quantum multiplexors.

1 Introduction

We say a unitary operator acting an array of qubits has been compiled if it has been expressed as a Sequence of Elementary Operations (SEO), where by elementary operations we mean few-qubit (like 1 and 2-qubit) operations such as CNOTs and single-qubit rotations. SEO's are often represented as quantum circuits.

This paper introduces[1] QOperAv v1.5, a Java application available for free. (Source code included in the distribution.) The name "QOperAv" is an abbreviation of the phrase "Quantum Operator Average", and is pronounced like the street, "Copper Av.". QOperAv is a "code generator" for generating quantum circuits. The quantum circuits generated by QOperAv can be used to evaluate with polynomial efficiency the average of $f(A)$ for some simple (that is, computable with polynomial efficiency) function f and a Hermitian operator A, provided that we know how to compile $\exp(iA)$ with polynomial efficiency. Such averages arise, for example, in an algorithm described in Ref.[6] for evaluating partition functions with a quantum computer.

Apart from its usefulness as a code generator, QOperAv is interesting in that it required very few lines of code to write, because it relies on classes that form part of a large class library that had been written previously. This class library has been used previously to construct many other applications (for example, QuanSuite, QuSAnn, Multiplexor Expander, and Quibbs).

QOperAv implements an algorithm discussed in Ref. [6]. The quantum circuit generated by QOperAv includes some quantum multiplexors. The Java application Multiplexor Expander (see Ref.[4]) allows the user to replace each of those multiplexors by a sequence of more elementary gates such as multiply controlled NOTs and qubit rotations. Multiplexor Expander is also available for free, including source code.

2 Input Parameters

The quantum circuit generated by QOperAv is described in detail in Ref.[6]. Using the notation of Ref.[6], the circuit depends on the following inputs:

N_B**:** This is a positive integer.

$N_{B\underline{j}}$**:** This is a positive integer.

γ**:** This is a positive real.

[1] The reason for releasing the first public version of QOperAv with such an odd version number is that QOperAv shares many Java classes with other previous Java applications of mine (QuanSuite discussed in Refs.[1, 2, 3], QuSAnn discussed in Ref.[4], Multiplexor Expander discussed in Ref.[4], and Quibbs discussed in Ref.[5]), so I have made the decision to give all these applications a single unified version number.

Δt**:** This is a positive real.

for $p = 0, 1, 2, \ldots, N_{B\underline{j}} - 1$**, a quantum circuit for** $\exp(i2^p A\Delta t)$**:** We call the unitary operator $\exp(iA\Delta t)$ an "atom" and the N_B qubits it acts on, the atom qubits. The demonstration version of QOperAv uses as an atom the circuit for an N_B-qubit quantum Fourier transform, and it raises the atom to the 2^p-th power by placing the atom inside a LOOP that repeats 2^p times, but both this particular atom and this method of raising the atom to a power can be changed easily by subclassing the class of QOperAv that defines this. In particular, rather than raising the atom to a power by repeating the atom circuit, the user could raise the atom to the 2^p-th power by replacing the parameter Δt by $2^p\Delta t$ in the atom circuit.

a quantum circuit for V**:** The unitary operator V acts on the N_B atom qubits. The demonstration version of QOperAv uses for V the circuit for an N_B-qubit quantum Fourier transform, but this can be changed easily by subclassing the class of QOperAv that defines this.

function $f : \mathbb{R} \to \mathbb{R}$**:** The demonstration version of QOperAv uses $f(\xi) = e^{-(0.1)\xi}$, but this can be changed easily by subclassing the class of QOperAv that defines this.

Let $N_S = 2^{N_B}$ and $N_{S\underline{j}} = 2^{N_{B\underline{j}}}$. The Hermitian operator A is assumed to have non-negative eigenvalues. Furthermore, Δt is assumed to be small enough that

$$A_x \frac{\Delta t}{2\pi} < \frac{N_{S\underline{j}} - 1}{N_{S\underline{j}}} \tag{1}$$

for all eigenvalues A_x of A. Furthermore, we assume that

$$0 \leq \gamma f(\frac{2\pi\ j}{\Delta t\ N_{S\underline{j}}}) \leq 1 \tag{2}$$

for $j = 0, 1, 2, \ldots N_{S\underline{j}} - 1$.

3 Output Files

QOperAv outputs 3 types of files: a Log File, an English File and a Picture File.

A Log File records all the input and output parameters that the user entered into the **Control Panel** (see Sec.4), so the user won't forget them.

An English File gives an "in English" description of a quantum circuit. It completely specifies the output SEO. Each line in it represents one elementary operation, and time increases as we move downwards in the file.

A Picture File partially specifies the output SEO. It gives an ASCII picture of the quantum circuit. Each line in it represents one elementary operation, and time

increases as we move downwards in the file. There is a one-to-one onto correspondence between the rows of corresponding English and Picture Files.

English and Picture Files are used in many of my previous computer programs. I've explained those files in detail in previous papers so I won't do so again here. See, for example, Ref.[5] for a detailed description of the content of those files and how to interpret that content.

4 Control Window

Fig.1 shows the **Control Panel** for QOperAv. This is the main and only window of QOperAv (except for the occasional error message window). This window is open if and only if QOperAv is running.

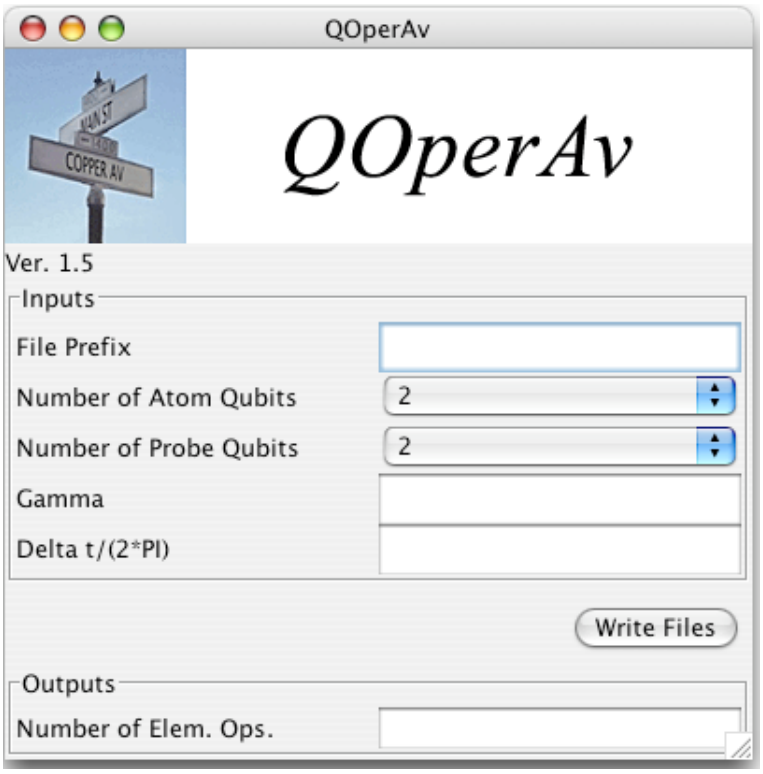

Figure 1: **Control Panel** of QOperAv

The **Control Panel** allows the user to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when the user presses the **Write Files** button. For example, if the user inserts `test` in this text field, the following 3 files will be written:

- `test_qoa_log.txt` This is a Log File.
- `test_qoa_eng.txt` This is an English File
- `test_qoa_pic.txt` This is a Picture File.

Number of Atom Qubits: This equals N_B.

Number of Probe Qubits: This equals $N_{B\underline{j}}$.

gamma: This equals γ.

Delta t/(2*PI): This equals $\Delta t/(2\pi)$.

The **Control Panel** displays the following output text boxes.

Number of Elementary Operations: This is the number of elementary operations in the output quantum circuit. If there are no LOOPs, this is the number of lines in the English File, which equals the number of lines in the Picture File. For a LOOP (assuming it is not nested inside a larger LOOP), the "`LOOP k REPS:`N" and "`NEXT k`" lines are not counted, whereas the lines between "`LOOP k REPS:`N" and "`NEXT k`" are counted N times (because `REPS:`N indicates N repetitions of the loop body). Multiplexors expressed as a single line are counted as a single elementary operation (unless, of course, they are inside a LOOP, in which case they are counted as many times as the loop body is repeated).

References

[1] R.R. Tucci, "QuanTree and QuanLin, Two Special Purpose Quantum Compilers", arXiv:0712.3887

[2] R.R. Tucci, "QuanFou, QuanGlue, QuanOracle and QuanShi, Four Special Purpose Quantum Compilers", arXiv:0802.2367

[3] R.R. Tucci, "Java Application that Outputs Quantum Circuit for Some NAND Formula Evaluators", arXiv:0802.2370

[4] R.R. Tucci, "Code Generator for Quantum Simulated Annealing", arXiv:0908.1633

[5] R.R. Tucci, "Quibbs, a Code Generator for Quantum Gibbs Sampling", arXiv:1004.2205

[6] R.R. Tucci, "Use of Quantum Sampling to Calculate Mean Values of Observables and Partition Function of a Quantum System", arXiv:0912.4402