

Simulating Quantum Computers

Benjamin Collins

It has been shown that if built, computers that take advantage of the unique effects of quantum mechanics would be superior to classical electric computers in both short-term memory storage and parallelism. At this stage in the development of the technology, not many standards are set and there are differences among researchers regarding how best to implement hardware for quantum computer. Also, quantum algorithm design is in its infancy with few well-known and useful algorithms published. In order to speed the design of algorithms and implementations for quantum computers tools must be used; simulations on classical computers are one such tool. Simulations of quantum computers are a fairly difficult problem when the limitations posed by classical hardware are considered. For example, the best known algorithm for factoring a binary number of N

bits requires a number of steps on the order of $O\left(\exp\left(\left(\frac{64}{9}\right)^{1/3} N^{1/3} (\ln N)^{2/3}\right)\right)$ [1]. Shor's

algorithm for prime factoring on quantum computers is designed to run in a polynomial number of steps, or on the order of $O(N^\alpha)$ for some $\alpha < \infty$, but when simulated on a classical computer, it is ultimately limited to the previous running time. However, these difficulties are either circumvented (massive parallelism can improve the complexity class of a simulation [2]) or ignored (in the case of Shor's algorithm, the problem is usually restricted in size to small numbers) and simulation plays a very important role in both quantum computer implementations and algorithm designs.

Physicists and engineers find that simulating the physical effects due to quantum interactions in their designs can help them discover many things that would otherwise require a great deal of time for experimentation, recording, and analysis. These types of simulations focus heavily on entanglement and the potential benefit of its effects, such as quantum teleportation. Most do not have access to the equipment needed to experiment, and simulation provides the best alternative. Precision is important for such simulations to be effective; if decoherence in an ion trap computer or the randomness associate with measuring qubits is to be studied, then a trustworthy representation of the system must be used. In most simulators available, there are one of two models used to represent quantum states; a vector representation as described by Nielson and Chuang [3] and Quantum Bayesian Nets [4].

The vector representation is simply a tool for expressing the state of a qubit, but it is good at expressing entangled states. This representation also uses the *Dirac* notation of kets and bras. A typical expression of a superposition might be expressed as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. If expanded into standard vector notation, this might look

like $|\psi\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$. Quantum Bayesian Nets are another thing altogether.

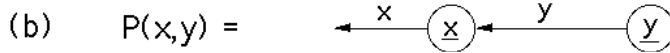
Quantum Bayesian Nets (QBs) maintain the theory of Classical Bayesian Nets (CBs) with one notable difference: QBs assign complex amplitudes to each node instead of probabilities like in CBs [4].



Figure 1, Possible CBs with 2 nodes, x and y . From [4].

a) Represents
 $P(x, y) = P(y | x)P(x)$

b) Represents
 $P(x, y) = P(x | y)P(y)$



From figure 1, it is easy to see how CBs can present an intuitive way to represent complicated probability distributions. The details of CBs are outside the scope of this paper; graph theory in particular is left out.

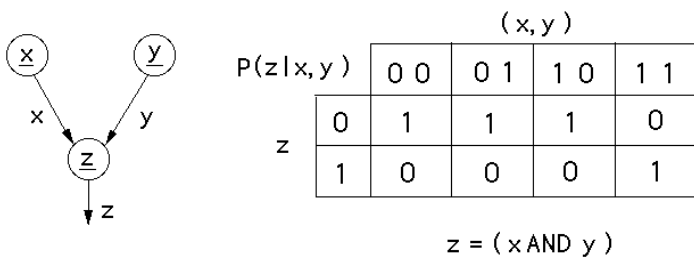


Figure 2, CB of classical AND gate. Variables x, y, z are binary: they can take on $\{0,1\}$. From [4].

QBs are different than the CBs shown in figures 1 and 2 in that they use complex amplitudes at the nodes instead of probabilities, but the concept is essentially the same. It allows for a model that can be visualized in an intuitive way, but can also be used to model computations, as is done in Quantum Fog [5].

Some examples of implementation simulators include QCE [10] (the Quantum Computer Emulator) and Quantum Fog. QCE is specifically focused on simulating an NMR implementation of a quantum computer. It is designed for the Win32 platform. It employs a graphical user interface and allows the creation of micro-instructions (based on effects of magnetic field). Micro-instructions are then used to construct short programs with which some algorithms can be tested. QCE uses the vector representation. Quantum Fog is a commercial simulator, created by the Artiste company [5]. It also employs a graphical user interface, but is designed for the Macintosh platform and employs Quantum Bayesian Nets to represent the underlying quantum states.

Simulation is as valuable to designers of quantum algorithms as it is to designers of implementations, if not more so. Simulation is somewhat easier in this paradigm because the quantum mechanical information can be idealized, or even ignored. Only the effects and features of a quantum computer are considered. Also, these kinds of simulators tend to be more collaborative, open and abstract than implementation simulators. Some of these simulators are complete environments, including a well-defined programming language and its interpreter/compiler like QCL [6]. Others are simply programming libraries that allow the investigator to use a particular existing language to express his algorithm in, like QDD [7] or Open Qubit (Open Qubit is a dead project) [8]. As in implementation simulations, there is more than one way to represent quantum states in algorithmic simulations. The standard vector representation is widely used, but others such as Boolean Decision Diagrams (BDDs) and Multiplicative Binary Moment Diagrams (*BMDs)[11] also have a place in algorithmic simulation.

Binary Decision Diagrams don't deal with probabilities, but they are a good way to represent and operate on Boolean functions. Figure 3 shows an example of a Boolean function and argument ordering. BDDs are a very weak representation of quantum states and cannot be used in the simulation of all algorithms. The advantage of using BDDs, however, is that they can provide an efficient way to approximate larger systems than would otherwise be practical for some algorithms. Each qubit is represented by its own BDD which is collapsed once the qubit is read.

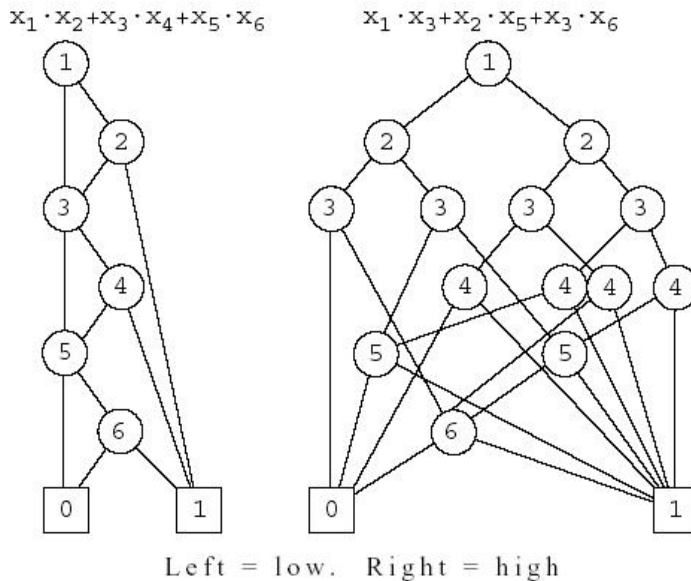


Figure 3. Argument ordering changes the BDD. From [9]. ‘.’ represents AND, ‘+’ represents OR.

The use of *BMDs is also under investigation for the QDD [7] project and perhaps others. *BMDs differ from BDDs in that they represent functions over Boolean variables like the BDDs, but the functions have non-Boolean ranges. They include multiplicative

edge weights to help represent functions.

There are several algorithmic simulators available [12]. Among these are QCL [6], QDD, and Open Qubit. As previously stated, QDD uses BDDs to internally represent the quantum states, however weakly. It is implemented as a C++ library intended for platforms with the Gnu GCC compiler. Open Qubit, unfortunately is a dead project. It employs the standard vector representation for the underlying states, and is also implemented as a C++ library. QCL, or the Quantum Computing Language, is more ambitious. mer [13] defines a fully-featured language for designing algorithms for a quantum computer. The language includes control structures, quantum operators, and built-in quantum types in addition to standard types. He allows the programmer to define his own operators as well as his own structures. QCL uses a vector representation for the underlying quantum states. Several important algorithms have already been demonstrated using QCL, including Shor's factorization algorithm.

There are a variety of simulators, most of which fall into one of two paradigms; simulating implementations of quantum computers such as NMR or ion traps, and creating an idealized environment which can efficiently simulate algorithms. Each simulator has its own representation for quantum states, including the vector representation, Bayesian Nets, Binary Decision Diagrams, and Multiplicative Binary Moment Diagrams. Each has its own advantages and disadvantages in helping investigators construct a functional quantum computer.

References

- [1] mer, Bernhard. *Simulation of Quantum Computers*. Thesis
Department of Theoretical Physics, Technical University of Vienna, 1996
- [2] Obenland and Despain. *Parallel Quantum Computer Simulation*.
http://www.hpcmo.hpc.mil/Htdocs/UGC/UGC98/papers/1e_chal/
Information Sciences Institute, U. Southern Cal., 1998.
- [3] Nielson and Chuang. *Quantum Computation and Quantum Information*
Cambridge University Press, 2000
- [4] Robert R. Tucci. *Quantum Bayesian Nets*.
Int. Jour. Of Mod. Phys. B9(1995)295-337
- [5] Quantum Fog website.
http://www.ar-tiste.com/QFogManual/cover_page.html
- [6] Bernhard mer. *Quantum Programming in QCL*.
Thesis, Dept. of Inst. of Inf. Sys., Technical University of Vienna 2000
- [7] QDD website
<http://home.plutonium.net/~dagreve/qdd.html>
- [8] Open qubit website
<http://www.ennui.net/~quantum/>
- [9] Randal E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*.
IEEE Trans. On Computers C-35-8 (1986) 677 – 691
- [10] Raedt, et. al. *Quantum Computer Emulator*.
Comp. Phys. Comm. 132 (2000) 1-20
- [11] R. E. Bryant. *Verification of Arithmetic Circuits with Binary Moment Diagrams*.
32nd Design Automation Conference, 1995
- [12] Simulation index web site
<http://www.dcs.ex.ac.uk/~jwallace/simtable.htm>
- [13] Bernhard mer. *A Procedural Formalism for Quantum Computing*. Thesis.
Dept. of Theoretical Physics, Technical University of Vienna 1998.