Chapter 2

QUANTUM COMPUTER SIMULATION

Chapter 1 discussed quantum computing in non-technical terms and in reference to simple, idealized physical models. In this chapter we make the underlying mathematics explicit and show how one can simulate, albeit inefficiently, the behavior of a quantum computer on an ordinary (classical) digital computer. Such simulation is necessary for the "fitness evaluation" steps of the methods for automatic quantum computer programming that will be described later in this book.

1. Bits, Qubits, and Gates

In classical computing the fundamental unit of information is the *bit*, which can exist in one of two states (conventionally labeled "0" and "1"). Bits can be implemented as positions of gears or switches, levels of charge, or any other conditions of any physical systems that can be easily and unambiguously classified into one of two states. Computations consist of sequences of operations, conventionally referred to as "gates," that are applied to bits and to collections of bits. The physical medium in which the bits and the gates are embedded may influence the computer's size, energy requirements, or "clock rate," but it has no impact on the fundamental computational power of the computer. Two computers with the same storage capacity (in bits) and the same set of supported operations (gates) can be considered equivalent for many purposes.

In quantum computing the fundamental unit of information is the *qubit*, which can also exist in one of two "computational basis" states (conventionally labeled using Paul Dirac's "bra-ket" notation as $|0\rangle$ and $|1\rangle$). But unlike the bit, the qubit can also exist in a *superposition* of $|0\rangle$ and $|1\rangle$ represented as $\alpha_0|0\rangle + \alpha_1|1\rangle$, where α_0 and α_1 are complex numbers such that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. The alphas here are the *ampli-*

tudes described in Chapter 1, and one can square the absolute value of an alpha to determine the probability that a measurement will reveal the corresponding state; for example $|\alpha_0|^2$ is the probability that measurement of the qubit will find it in the $|0\rangle$ state.

The physical medium underlying a quantum computer, like that underlying a classical computer, is generally not relevant to discussions of the computer's fundamental computational power. All that is necessary is that the medium supports the units of storage (qubits) and the relevant operations (quantum gates). A wide variety of proposals has been developed for implementing qubits and quantum gates, including schemes based on optics, ion traps, and the manipulation of nuclear spins in nuclear magnetic resonance devices. All of these schemes present engineering challenges, and many are under active development. We will not be concerned with the details of any of them in the present book, because the computational properties in which we are primarily interested are captured by the abstract view of the quantum computer as a collection of qubits, on which we operate by means of mathematically specified quantum gates. Automatic programming techniques similar to those described later in this book, but built on models of particular implementation schemes, may be be useful for exploring limits or opportunities of the corresponding implementations.

In classical computing the representation of an n-bit system is simply the concatenation of the representations of n 1-bit systems. For example the state of a 5-bit register might be represented as 10010. In quantum computing the representation of a multi-qubit system is more involved, because the individual qubits are not independent of one another. Indeed, qubits in a quantum computer can become "entangled" with one another, and this entanglement underlies several interesting quantum algorithms (Jozsa, 1997; Bennett, 1999). The nature of quantum entanglement is a subject with an enormous literature and a rich history, some of which bears directly on questions about quantum computation. A few suggested entry-points into this literature are (Bell, 1993), (Deutsch, 1997), (Albert, 1992), and many of the essays in (Hey, 1999).

To represent the complete state of a multi-qubit system one must in general store a complex amplitude for each *combination* of basis values ($|0\rangle$ and $|1\rangle$) over the entire system. So, for example, the state of a 3-qubit register might be represented as $\alpha_0|000\rangle + \alpha_1|001\rangle + \alpha_2|010\rangle + \alpha_3|011\rangle + \alpha_4|100\rangle + \alpha_5|101\rangle + \alpha_6|110\rangle + \alpha_7|111\rangle$, where the squares of the absolute values of the alphas sum to 1.

Quantum gates can be formalized as matrices, with the application of a gate to a quantum computer state implemented as the multiplication

of the gate's matrix times a column vector containing the state's amplitudes. What sense does this make? Let us first look at a classical version of this idea. Consider a 2-bit classical register. Such a register can be in one of four possible states, namely 00, 01, 10, or 11. Suppose, for reasons that will seem perverse until we generalize to the quantum case, that we wish to represent the state of this register not using the two bits themselves, but rather by recording individually the "amplitudes" for each of the four possible states. Since the register is classical it cannot be in a superposition — it will always be in one particular state. The amplitude corresponding to the actual state of the register will be 1, and all of the other amplitudes will be 0. We will write the amplitudes in the form of a column vector in binary order; that is, the number on top will be the amplitude for the 00 state, the next one will be the amplitude for the 01 state, and so on. So the four possible states of this 2-bit classical register will be represented as:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

What classical operations can be performed on such a register? Although they can be built in various ways from Boolean primitives, all allowable operations have the effect (if they have any effect at all) of changing the state of the register from one of these four states to another. And any such operation can be represented as a matrix, consisting only of 0s and 1s, which, when applied to a state vector (via matrix-vector multiplication), produces another valid state vector. For example, consider the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This matrix will have no effect when applied to 00 or 01, but it will transform 10 into 11 and 11 into 10. That is, it will act as a "`NOT`" operation on the right-most bit *if and only if* the left-most bit is 1. For this reason this is often called a "controlled `NOT`" or "`CNOT`" gate. All permissible transformations of the 2-bit register can be represented similarly, using 4×4 matrices containing only 0s and 1s. Not all such matrices are permissible — only those that are guaranteed to produce valid classical state vectors (containing one 1 and the rest 0s) when applied to valid classical state vectors.

Quantum computation can be viewed, mathematically, as a generalization of this classical matrix model. The first generalization is that the amplitudes in the state vectors are no longer required to be 0 or 1. Each amplitude can be any complex number, as long as the squares of the absolute values of the amplitudes sum to 1.[1] Similarly, the set of permissible operations (matrices) is expanded to include any matrix that meets the condition of *unitarity*, which can be expressed (in one formulation) as the requirement that:

$$U^{\dagger}U = UU^{\dagger} = I$$

Here U is the matrix in question, $U^{\dagger}$ is the *Hermitean adjoint* of U (obtained by taking the complex conjugate of each element of U and transposing the result), and I is the identity matrix. The multiplication of a vector of amplitudes by any unitary matrix will always preserve the "summing to one" constraint described above. Although there are infinitely many such unitary matrices, a small finite set suffices for quantum computational *universality* in the same sense that the `NAND` gate suffices for classical computation (Barenco et al., 1995).

In this book we use a selection of quantum gates similar to that used elsewhere in the quantum computing literature. We use the `CNOT` gate described above, along with the simpler 1-bit `Quantum NOT` or `QNOT` gate with the matrix:

$$QNOT \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

We also use a family of 1-qubit "rotations" parameterized by an angle θ, with matrices of the form:

$$U_{\theta} \equiv \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

Another 1-qubit gate, called `Square Root of NOT` or `SRN` provides a good example of the non-classical power of quantum gates. We use a version of `SRN` with the following matrix (which is also equivalent to $U_{-\frac{\pi}{4}}$):

$$SRN \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

[1] Squaring does not obviate the taking of the absolute value, because some amplitudes will be complex and have negative squares.

When applied to a qubit that is in either the $|0\rangle$ or the $|1\rangle$ state, it leaves the qubit in an equal superposition of $|0\rangle$ and $|1\rangle$ — that is, it appears to randomize the value of the qubit, since a measurement after the application of the gate will produce 0 or 1, each with 50% probability. But this is not simple randomization, as the qubit's history can still influence its future behavior. A second application of `SRN` to the qubit will leave it, deterministically, in the opposite of the state in which it started — that is, measurement will produce 0 if the intial state was $|1\rangle$, or 1 if the initial state was $|0\rangle$.[2] So two applications of `SRN` produce the effect of `QNOT`, which is why `SRN` has the name that it does.

The final 1-qubit gate that we routinely employ is the `HADAMARD` gate, with the following matrix:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This gate is similar to `SRN` except that it acts more like a "square root of identity." It is useful for creating and "decoding" superpositions in a variety of quantum algorithms.

It is sometimes helpful to use a fully-parameterized 1-qubit gate, which can act as any other 1-qubit gate if its parameters are set appropriately. One form for this "generalized rotation," which we call $U2$, is as follows:

$$U2 \equiv \begin{bmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{bmatrix} \times \begin{bmatrix} \cos(\theta) & \sin(-\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} e^{-i\psi} & 0 \\ 0 & e^{i\psi} \end{bmatrix} \times \begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

Other useful 2-qubit gates include the `Controlled Phase` gates, with matrices of the form:

$$CPHASE \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{bmatrix}$$

Finally, the `SWAP` gate, which simply swaps the states of two qubits, is often handy:

$$SWAP \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

[2] Actually, the matrix for two consecutive applications of `SRN` is $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$, meaning that two applications of `SRN` to $|1\rangle$ will produce $-|0\rangle$, although the change in sign has no effect on measurements. *Six* applications would be required to obtain $|0\rangle$.

Specific problems may call for the use of additional gates. For example, many problems are phrased with respect to a "black box" or "oracle" gate, of which we are asked to determine some property. Grover's database search problem is of this sort; we are given a multi-qubit gate that encodes a database, and we are asked to determine which input will produce a "yes" output (which the oracle usually indicates by flipping — `QNOT`ing — a specified qubit).

2. Gate-Level Simulation

There are many approaches to quantum computer simulation. At one extreme one can attempt to simulate, as realistically as possible, the exact interactions involved in a particular physical device, including noise and other effects of imprecision in the design of the physical components. For example, Kevin Obenland and Alvin Despain simulated a quantum computer that manipulates trapped ions by means of laser pulses, modeling imperfections in the laser apparatus as deviations in the angles of rotations (Obenland and Despain, 1998). Alternatively, one could simulate the quantum computer at a higher level of abstraction, ignoring implementation details and working only with "perfect" unitary matrices.

If one wishes to simulate the execution of *arbitrary* sequences of quantum gates then one necessarily faces exponential space and time costs whether one works at the implementation level or at a more abstract level. That is, if the number of qubits in the system is N, then the space and time requirements for simulation will both scale approximately as 2^N.

In order to evolve quantum algorithms, as described in Chapter 7, we must indeed be able to simulate the execution of arbitrary sequences of quantum gates. But since our focus is on the theoretical power of quantum computing, and not on the strengths or weaknesses of any particular implementation, we can conduct our simulations with straightforward matrix mathematics. We will explicitly maintain full vectors of complex amplitudes, upon which we will explicitly conduct large matrix multiplications. We will pay exponential costs for this form of simulation but the simulation techniques will be conceptually simple.

The exponential costs associated with simulation will limit the range of problems to which our automatic programming techniques can be applied. We will generally seek applications that involve only small quantum systems or that produce algorithms that can be "scaled" to various sizes by hand after they have been discovered automatically. Fortunately, there do seem to be many problems for which the simulation costs are not prohibitive.

Simulation shortcuts are possible if one knows in advance that the algorithm being simulated obeys certain constraints — that is, that certain amplitudes will always be zero, or that certain amplitudes will have values that can be quickly re-derived (so that one needn't always store them all explicitly), or that certain types of entangled states will never be produced. Such constraints, combined with clever encoding schemes, can lead to substantial improvements in simulation speed for many algorithms, although exponential costs will still be incurred in the worst case (Viamontes et al., 2002; Viamontes et al., 2003; Udrescu-Milosav, 2003). These types of advanced simulation techniques are not discussed further in this book, but they could certainly be incorporated into the automatic quantum computer programming framework described here, and one would expect their incorporation to increase the reach of the technology.

To perform the full matrix mathematics described in the previous section we must generally expand the compact matrices that characterize the gates to the appropriate size for the complete quantum system being simulated. For example, if we wish to apply a `QNOT` gate to the right-most qubit of a 3-qubit system then it is not enough to multiply two amplitudes by the 2×2 matrix that characterizes `QNOT`. Rather, one must do something that affects *all* amplitudes in the system, effectively multiplying it by the following 8×8 matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

In this case the expansion of the 2×2 matrix to produce the 8×8 appears relatively straightforward, but the process is more confusing when one must expand a multi-qubit gate, particularly when the qubits to which it is being applied are not adjacent in the chosen representation. For example if one wishes to apply a `CNOT` gate in a 3-qubit system, using the right-most qubit as the "control" input and the left-most qubit as the "target" (the one that is flipped when the control qubit is 1), then one must effectively use the following 8×8 matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

How does one construct the needed matrix expansion? First, note that one needn't necessarily construct the matrix (the "tensor product") *explicitly*. In many cases it will suffice to perform an operation which has the same effect as multiplication by the expanded matrix, but which uses only the compact representation of the gate. We call this "implicit matrix expansion."

In other cases one *does* want the explicit representation of the expanded gate, for example because one wants to multiply several expanded gates with one another for storage and later re-application. The choice between implicit and explict matrix expansion presents a trade-off between space requirements and flexibility. With implicit matrix expansion one must store the matrices only in their compact forms, which can be a considerable savings. For example, a 1-qubit gate in its compact form can be represented with only 4 complex numbers, whereas the explicit expansion of this gate for a 10-qubit system consists of $1,048,576$ complex numbers. On the other hand, the expanded forms may be convenient for certain purposes both in the evolution and in the analysis of quantum algorithms. An ideal simulator will therefore provide both options and allow the user to switch among them according to need.

An algorithm for explicit matrix expansion is provided in Figure 2.1, and an algorithm for applying an implicitly expanded gate is provided in Figure 2.2. Source code for these algorithms is included in the distributions of QGAME, a quantum computer programming language and simulation system described in the following chapter; the code for applying an implicitly expanded gate is included in the minimal version of QGAME in the Appendix of this book.

A variety of other approaches to quantum computer simulation exist, some of which are based on alternative conceptualizations of quantum computers (for example, on "quantum Turing machines" or "Feynman computers"). Source code for other simulators can be found in other texts (for example, Williams and Clearwater, 1998) and via internet searches.

To expand gate matrix G (explicitly) for application to an n-qubit system:

- Create a $2^n \times 2^n$ matrix M.
- Let Q be the set of qubit indices to which the operator is being applied, and Q' be the set of the remaining qubit indices.
- $M_{ij} = 0$ if i and j differ from one another, in their binary representations, in any of the positions referenced by indices in Q'.
- Otherwise concatenate bits from the binary representation of i, in the positions referenced by the indices in Q (in numerical order), to produce $i^\star$. Similarly, concatenate bits from the binary representation of j, in the positions referenced by the indices in Q (in numerical order), to produce $j^\star$. Then set $M_{ij} = G_{i^\star j^\star}$.
- Return M.

Figure 2.1. An algorithm for explicit matrix expansion.

To apply gate matrix G (expanded implicitly) to an n-qubit system:

- Let Q be the set of qubit indices to which the operator is being applied, and Q' be the set of the remaining qubit indices.
- For each combination C of 0 and 1 for the set of qubit indices in Q':
  - Extract the column A of amplitudes that results from holding C constant and varying all qubit indices in Q.
  - $A' = G \times A$.
  - Install A' in place of A in the array of amplitudes.

Figure 2.2. An algorithm for applying an implicitly expanded gate.

http://www.springer.com/978-0-387-36496-4

Automatic Quantum Computer Programming
A Genetic Programming Approach
Spector, L.
2007, XII, 154 p., Softcover
ISBN: 978-0-387-36496-4