ECE 510—Intro to Computational Intelligence
Quantum Project (Perkowski)
Eric Paul (12/3/2007)

Introduction

When solving complex problems, parallel processing is often used to speed up the solving time. Quantum Computing is one example of parallel processing. It is modeled after quantum phenomena that are well known, yet my understanding of said phenomena is marginal at best. However, given the computational models, in this exercise we will go through the process of designing a quantum computer system for a specific application: the graph coloring problem.

Instead of trying to tackle a complex problem and realize an equally complex solution, we will design a system that realizes the basis form of a graph. The idea here is that we can map the design process to a more complex graph once we understand how the quantum computation methodology can be applied to the basis case.

Problem Description

The problem chosen for this exercise is the vertex-coloring problem from graph theory. A good coloring is defined as assigning colors to each of the vertices such that no adjacent nodes (two nodes sharing an edge) are the same color. There is a well known theorem—the "four-color theorem"—first put forth by P.J. Headwood in 1890. Headwood stated that for any planar graph, the minimum number of colors is at most four. Figure 1 shows an example of both a planar and non-planar graphs.

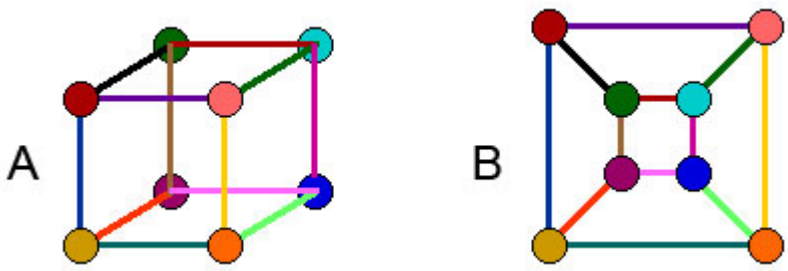

Figure 1: Planar & Non-planar graphs

In figure 1A, we see a 3-dimensional graph that is colored with eight colors, the same number of nodes that appear in the graph. The four-color theorem says nothing about a graph of this type. The same graph can be redrawn, as in figure 1B, in a planar fashion. Again we see that the number of colors is eight; the four-color theorem tells us that the minimum number of colors for vertex coloring—the chromatic number for this graph—is at most four.

Both of these graphs are relatively simple in their structure, meaning that an exhaustive test would be feasible and possible to find the chromatic number. However for much larger and complex

graphs, some sort of computational tool would be required. We could use a classical Boolean circuit to discover the chromatic number for some arbitrary graph, but the testing would need to be done sequentially (one color combination and count after another), and would take a long time for a large, complex graph. The parallel nature of quantum computing would help us determine the chromatic number due to its ability to test all possible combinations at once.

We wish to highlight the design process for building a simulated quantum computer, so for the purposes of simplicity and clarity we will develop a quantum circuit that discovers the chromatic number for the basis graph: a two-node, single edge graph (fig 2). The idea is that we could expand the design process to a larger, more complex graph.

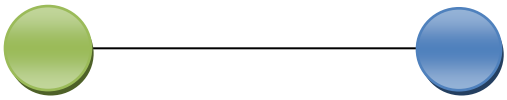

Figure 2: Two-Node Graph

In figure 2, it can easily be seen that the chromatic number is the same as the maximum number of colors: two.

Oracle Design

The quantum oracle is a device that can answer our question of what the chromatic number is for the 2-node graph; it is the heart of the quantum computer. For any quantum computer, the "width" of the circuit is determined by the number of inputs, the number of work bits required for the computation, plus one for the oracle bit. In our case, the maximum number of colors is two, which can be encoded in one bit per node (one line per node). No work bits (ancilla) bits are required for this simple problem, so the width of our circuit is three lines.

In order to design the quantum circuit, we start with the classical Boolean version. The circuit must have two inputs and one answer bit for both input and output. We just need to discover if the color for node 1 is different than node 2, so the first comparison is simply an XOR of the inputs. In order to transfer this answer to the oracle (answer) line, we XOR the previous operation with the discrete value zero. This simple circuit is shown in figure 3.

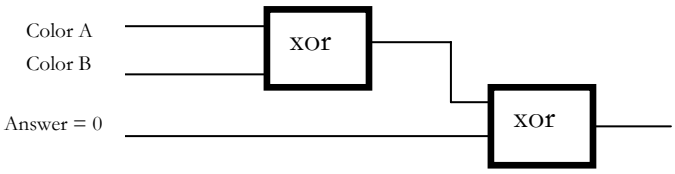

Figure 3: Boolean Implementation

In this configuration, the answer bit is initialized to zero. We now wish to transform this implementation into a quantum circuit. As noted earlier, the width of the circuit is constant through all stages, so all quantum gates will need to be 3 lines wide.

The quantum version of the 2-input XOR gate is just a Feynman gate, so the quantum circuit will be structured as in figure 3.

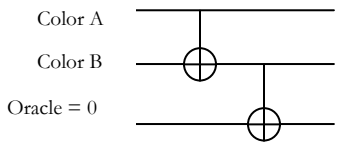

Figure 2: Quantum Oracle

The implementation of the oracle will be realized in matrix form. All gates in the oracle are based on The 2-input Feynman gate and the single-input wire (identity). The matrices for the Feynman gate and a simple wire are

$$FEY = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad [\text{EQ1}]$$

$$WIRE = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \qquad [\text{EQ2}]$$

The Kronecker product can be used to realize these parallel connections. The order of the parallel placement is important in deriving the 3-line wide gates. And in the case of the first gate, we need a Feynman in parallel with a wire. This order is reversed for the second gate. The first gate will be represented with U1 and the second gate with U2:

$$U_1 = FEY \otimes WIRE \qquad [\text{EQ3}]$$

$$U_2 = WIRE \otimes FEY. \qquad [\text{EQ4}]$$

The following product of matrices becomes our oracle:

$$O = U_2 U_1. \qquad [\text{EQ5}]$$

Notice that the product is derived by multiplying the matrices in reverse. Next, we consider how to implement the oracle using the Grover Algorithm.

The Grover Algorithm

The Grover Algorithm is a method by which we use the matrix representation of our oracle to realize the overall quantum circuit. The algorithm is iterative, and we need to generate an oracle "container"—referred to here as the Grover block and implemented as a matrix—that can be applied k times. The value of k is known to be

$$k = \left\lceil \left(\frac{\pi}{4}\right)\sqrt{N} \right\rceil, \qquad \text{[EQ6]}$$

where N is the number of inputs. For our circuit, there are N=2 inputs, so the number of Grover iterations in the quantum circuit is k =2.

Once the Grover block is designed, we can apply the block to the inputs and oracle bits as depicted in figure 3.

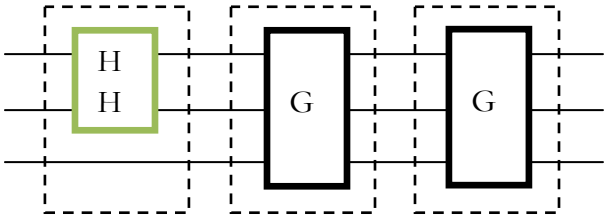

Figure 3: Grover Algorithm Flow

In this diagram, the Grover block is applied to the inputs twice. The dotted boxes represent operator matrices, the "HH" block is a two-element vector of Hadamard gates, and the "G" block is the Grover Block. A single Hadamard gate is the how we model the quantum phenomena of "exploding" the encoded input into the superposition of all possible states for a given input. Its matrix form is

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \qquad \text{[EQ7]}$$

The matrix that represents two Hadamards and a wire in parallel can be calculated with the Kronecker product:

$$H2WIRE = H \otimes H \otimes WIRE, \qquad \text{[EQ8]}$$

WIRE, as in the oracle design, is just the identity matrix. The Grover block in figure 3 is defined by the diagram in figure 4.

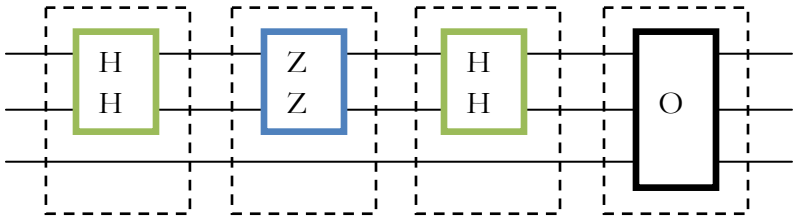

Figure 4: Grover Block Implementation

The "HH" block and the associated matrix is the same as the matrix defined by equation 8 in the Grover block. The "ZZ" block is a two-element zero-state phase shift vector. The single zero-state phase shift operator in matrix form is

$$Z = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}. \qquad [\text{EQ9}]$$

This transformation serves as the model for the quantum bit flipping. The theory behind this phenomena and how it helps us arrive at the desired solution is unclear, but from the literature we know it is necessary.

Going back to figure 4, the matrix that represents two zero-state phase shift gates in parallel with a wire is derived from the Kronecker product

$$Z2WIRE = Z \otimes Z \otimes WIRE. \qquad [\text{EQ10}]$$

Using reverse-multiplication of matrices, the Grover block can be calculated by

$$G = O \cdot H2WIRE \cdot Z2WIRE \cdot H2WIRE. \qquad [\text{EQ11}]$$

Referring to figure 3, we now can derive the overall matrix that we apply the inputs to using reverse multiplication:

$$M = G \cdot G \cdot H2WIRE. \qquad [\text{EQ12}]$$

Lastly we need an input vector; three quantum lines can be implemented using the Kronecker product of 3 zero-quantum-state variables:

$$IN3 = |0\rangle \otimes |0\rangle \otimes |0\rangle. \qquad [\text{EQ13}]$$

Applying this to our operator matrix, the output becomes

$$OUT = M * IN3. \qquad [\text{EQ14}]$$

Simulation Results

The translation of the previously outlined steps was implemented in the following script (gcolor2.m):

```
% Title:    ECE 510 Quantum Project
% File:     gcolor2.m
% Author:   Eric Paul (November 2007)

% This file implements a quantum computer for a two-node planar graph
% coloring problem.

clear all;

%% GENERATE BASIC QUANTUM STATE INPUT VECTOR [equation 13]
IN3 = kron(kron([1;0],[1;0]),[1;0]) % 3 lines: |0>*|0>*|0>

%% IDENTITY MATRIX DEFINITION FOR A WIRE [equation 2]
WIRE = [1,0;0,1];

%% HADAMARD VECTOR
H = 1/sqrt(2)*[1,1;1,-1];   % H -> hadamard basis [equation 7]
H2 = kron(H,H);             % H || H

%% GENERATE INITIAL CIRCUIT: H || H || WIRE [equation 8]
H2_WIRE = kron(H2,WIRE);

%% GENERATE ORACLE
FEY = [1,0,0,0;...  % 2-input Feynman gate [equation 1]
       0,1,0,0;...
       0,0,0,1;...
       0,0,1,0];

U1 = kron(FEY,WIRE);    % Feynman || wire [equation 3]
U2 = kron(WIRE,FEY);    % Wire || Feynman [equation 4]

ORACLE = U1*U2; % XOR [equation 5]

%% GENERATE GROVER BLOCK
Z2 = [-1,0,0,0;...  % 2-input zero-state phase shift gate
       0,1,0,0;...
       0,0,1,0;...
       0,0,0,1];
Z2_WIRE = kron(Z2,WIRE);    % Z2 || WIRE [equation 10]
G_BLOCK = H2_WIRE*(Z2_WIRE*(H2_WIRE*ORACLE)); % [equation 11]

%% GROVER ALGORITHM [Equations 11-14]
IN_H2_WIRE = H2_WIRE * IN3;     % apply Hadamards to input
OUT = IN_H2_WIRE;               % initialize output

% set iteration count for grover loop
% DISCREPANCY: iterations required is 1 more than calculated by equation 9
itr = 3

% grover loop
```

```
for i=1:itr
    OUT = G_BLOCK * OUT;
end

OUT
```

Notice that we have a discrepancy between the number of calculated iterations and the number of iterations that provided the desired output. The formula for calculating the number of required iterations in equation 6 was given, and the derivation is unclear. If the reasoning behind this equation was known, we might be able to discover what causes the discrepancy in the simulation. In any case, the simulation gives the expected results with the iteration count adjustment. The MATLAB output is as follows:

```
>> gcolor2

OUT =

         0
   -0.5000
   -0.5000
         0
         0
   -0.5000
   -0.5000
         0
```

Mapping the output vector into a truth table, and ignoring the 0.5 scaling factor, we find

IN	OUT
000	0
001	-1
010	-1
011	0
100	0
101	-1
110	-1
111	0

The Grover Algorithm tells us that where we see negative values in the truth table, the associated input combination is the realization of the oracle saying "yes" to the design question. In the truth table we see four state combinations that result in good coloring: whenever the lines that represent the node colors are in opposite states.

It is my understanding that we are interested in the rows that correlate to the oracle bit's actual value in the design. In reference to figures 3 and 4, the oracle was designed with the oracle bit set to a value of zero. In the truth table, the oracle being in state 0 upon initialization (input state) and the output representing good coloring is realized by the first two -1 outputs. The shaded rows of the truth table represent this combination of input and output states. So, the interpretation is that the oracle has correctly found that there are 2 valid graph colorings: when the colors for the two input lines are inverses of each other. From this, we can say that the oracle, and the overall quantum computer is functioning as designed.

Conclusion

The purpose of this exercise was to design a quantum computing system to solve the graph coloring problem for the basis graph of two nodes. My understanding of the quantum phenomena we are modeling was, and continues to be, very incomplete. The Grover Algorithm is quite abstract as well, and I'm not sure if it was implemented properly in the solution presented here. Recall that we had to increase the iteration count for the Grover loop by one. Again, my understanding is not complete enough to even guess the cause of the discrepancy or how to fix it. That is, assuming that all other aspects of the implementation are correct. I think a better understanding of the computational models, and how they describe the operation of a quantum computing system, is required to ensure an accurate solution.

However, the computational models covered in class were the basis for this design, and overall results seem to imply that my interpretation of the models function properly for this specific problem. Given more time, it would be interesting to expand this procedure to solve the graph coloring problem for a larger and more complex graph. Furthering this idea, it would be even more interesting to develop a simulated quantum computing system that could test any given oracle. This would mean that we would need a well tested, known-to-be-good Grover Algorithm wrapper for the oracle under test. The amount of development time required for such a general system would surely be more than the time spent here.

In any case, I feel that I have learned something (the rudimentary basics at best) about the quantum computing implementation for parallel processing.