Michael Dandrea

C09 Writeup

This assignment was to make a simple program to find a masked benign error in a set of data where one bit is randomly flipped. To do this, I made a loop with a number of trials given by the user (like you said to do when Cade and I went to your office). Inside the loop, I set a and b to a random number between 0 and (2^n) -1. Then, I make c equal to a \* b and flip a random bit in b, making it b prime. Then, I multiply a with b prime to make c prime. If c and c prime are equal, then a masked benign error occurred. The code to do this is below:

A screen shot of a computer screen

Description automatically generated

When -p is given, the results are printed to the screen.

I ran this code with the following command:

./simple\_mbe\_test -n <number of trials> -p > logfile.csv | tee (where <number of trials> was 10000 for the chart provided below)

This command made created “logfile.csv” which is used in the python program included here called graph.py. This program makes a histogram for the data from the test. It is provided below for when <number of trials> is 10000 for simplicity:

A graph of a graph showing a number of values

Description automatically generated

The numbers above the bar show how often an mbe occurred for a given bit position.

The other thing the c program outputs is another csv file called results.csv which has all of the trials done (unless you do make clean). There is another python program called chart.py that uses results.csv to make a line graph showing the probability of an mbe for a given size. There is chart below with test sizes of from 1000 to 1000000000, adding a 0 every time for simplicity:

A graph with a line

Description automatically generated

As we can see, as the input size gets larger the probability for an mbe starts to average out around 3.13%.

Here is a screenshot of the program running where -n is 1000 for simplicity:  
A screen shot of a computer

Description automatically generated

As there is nothing being malloc’d or freed, there is no reason to run it through valgrind to check for memory leaks.

A sample workflow for the programs would be:

make clean all

./simple\_mbe\_test -n <number of trials> -p > logfile.csv | tee

python graph.py

(repeat step 2 and 3 if wanted for trials from 1000 to 1000000000 adding one 0 each time)

python chart.py

graph.py automatically takes logfile.csv as an input, which is overwritten every time the program is run. If a graph is wanted from it, it must be created before doing another run of the program. chart.py automatically takes result.csv as an input, which is appended every time the program is run, so this only needs to be done at the end.