EEL 4768 Project 2

G-Share Global Branch Predictor

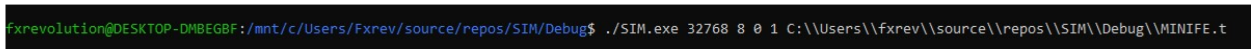
Felix Henriquez

README

Use the following instructions to run the simulator on your system.

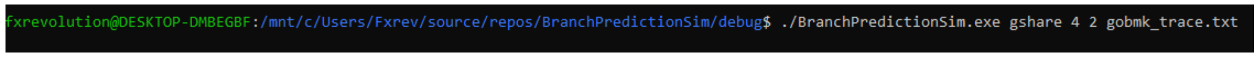
- Using a Windows or Linux command line terminal, change directory to the folder containing the executable file BranchPredictionSim.exe (i.e. "…/BranchPredictionSim/Debug/")
- Once in the folder execute the program in the following format:
  - **./BranchPredictionSim.exe Gshare <M> <N> <Trace_File path>**
- The Trace File Path can be the file's full path with the file name as shown below:

  NOTE Please use \\\\ instead of \\ because the string will not ignore escape characters



```
fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/SIM/Debug$ ./SIM.exe 32768 8 0 1 C:\\Users\\fxrev\\source\\repos\\SIM\\Debug\\MINIFE.t
```

- If the trace file is in the same folder as the executable the file path can be just the filename and extension.



```
fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 4 2 gobmk_trace.txt
```

- The trace file must contain input lines in the following format
  - **Address Taken/Not Taken**
  - 0x123456789abcdef t
  - 0xfedcba987654321 n
- The program was created using Visual Studios and the solution can be viewed on the platform with minimal configuration.
- To view the code in Visual Studios go to the project folder "BranchPredictionSim" and open the solution file "BranchPredictionSim.sln" using VS 2017

The Simulator outputs the following values to the console:

M N Miss Percentage

Objective:

To create a simulation of a gshare global branch predictor that can be configured to use any number of M bits from the PC address and N bit global history register, were N <= M.

Design Specification Plan:

This branch predictor is implemented using a C++ vector. The vector acts as the container for Branch Prediction Table. The simulator follows these steps to determine the miss percentage:

1. Initiate all indexes in the prediction table with Weakly Taken value.
2. With a given address, remove the unused bytes by right shifting two bits.
3. Get the M number of bits by modding the address by 2^M.
4. Gets the index by right shifting the Global Prediction History (M-N) bits and XOR with M bits.
5. Check the index in the prediction table:
    - If the bit counter is greater than Weakly Taken, the Prediction is Taken, else, Not Taken.
    - If the actual result is Taken, increment the bit counter, saturate at Strongly Taken.
    - If Not Taken, decrement the bit counter, saturate at Strongly Not-Taken.
    - If Prediction doesn't match actual result, increment the miss counter.
    - If the actual result is Taken, shift the GPH register right 1 bit and input 1 for MSB.
    - If Not Taken, just shift the GPH register right 1 bit.
    - Increment the Total Trace Counter.

Using vectors seemed to be the simplest approach. Vectors are great for dynamic memory allocation. Since the index is a hash of the Program Counter and the Global Prediction History register, there is O(1) insertion and access to the bit counter at that index. In the interest of code readability and to reduce debugging time, vectors are also the better choice as the container design.

Screenshots:

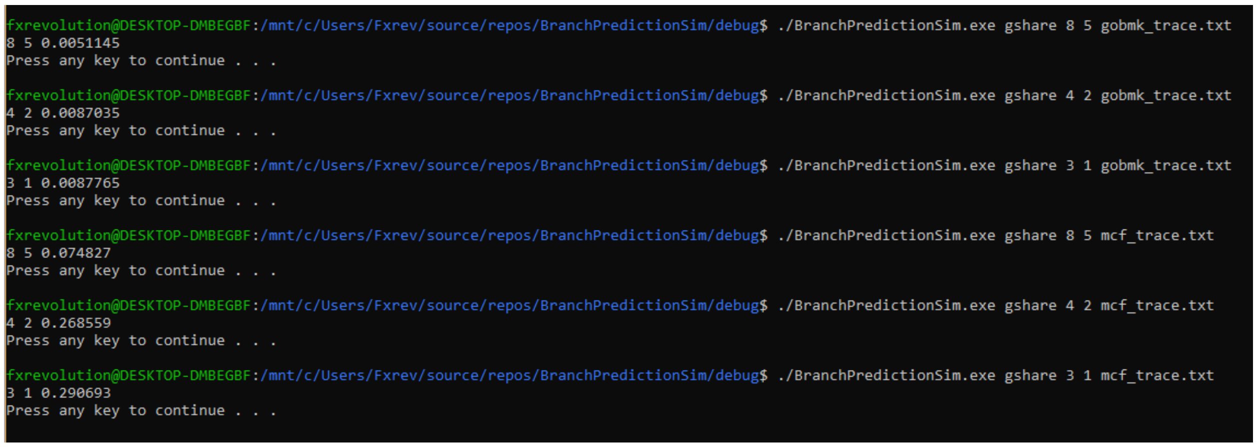
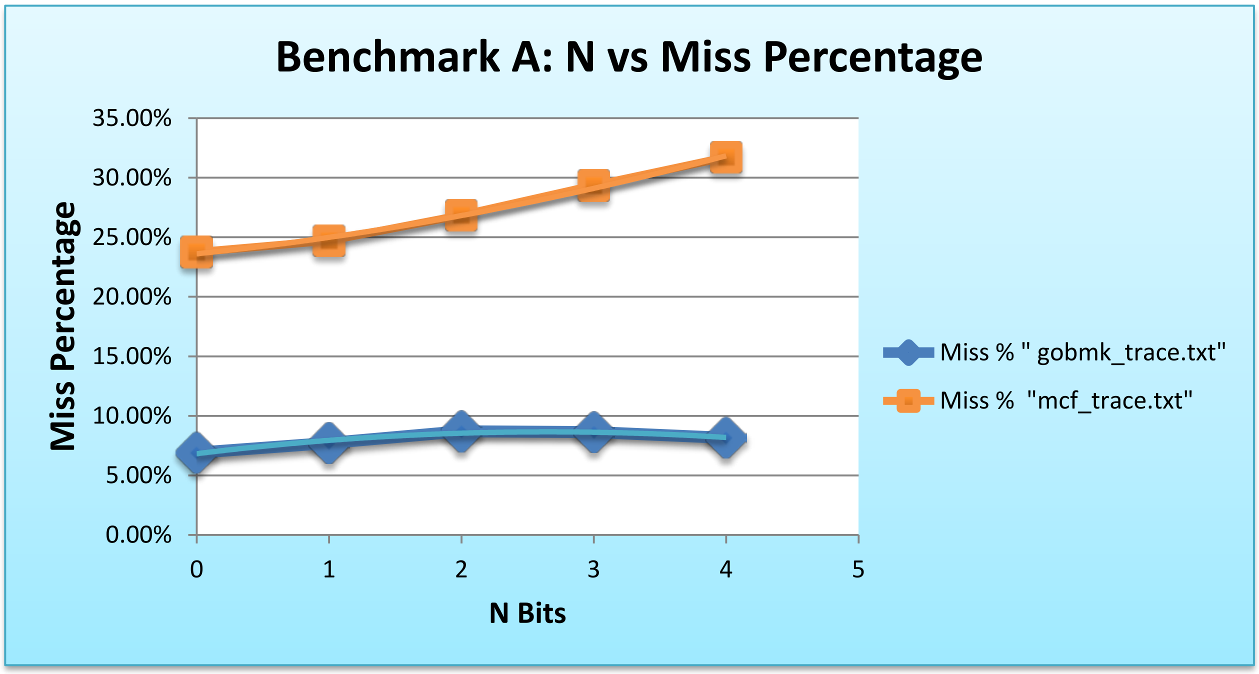

```
fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 8 5 gobmk_trace.txt
8 5 0.0051145
Press any key to continue . . .

fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 4 2 gobmk_trace.txt
4 2 0.0087035
Press any key to continue . . .

fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 3 1 gobmk_trace.txt
3 1 0.0087765
Press any key to continue . . .

fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 8 5 mcf_trace.txt
8 5 0.074827
Press any key to continue . . .

fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 4 2 mcf_trace.txt
4 2 0.268559
Press any key to continue . . .

fxrevolution@DESKTOP-DMBEGBF:/mnt/c/Users/Fxrev/source/repos/BranchPredictionSim/debug$ ./BranchPredictionSim.exe gshare 3 1 mcf_trace.txt
3 1 0.290693
Press any key to continue . . .
```

Figure 1 Validation Runs

Summary:

The following table is data gathered using the gobmk_trace and the mcf_trace files to analyze how the miss prediction ratio of these workloads changes with increasing N bit correlation. M is fixed at 4; N is increased from 0 to 4.

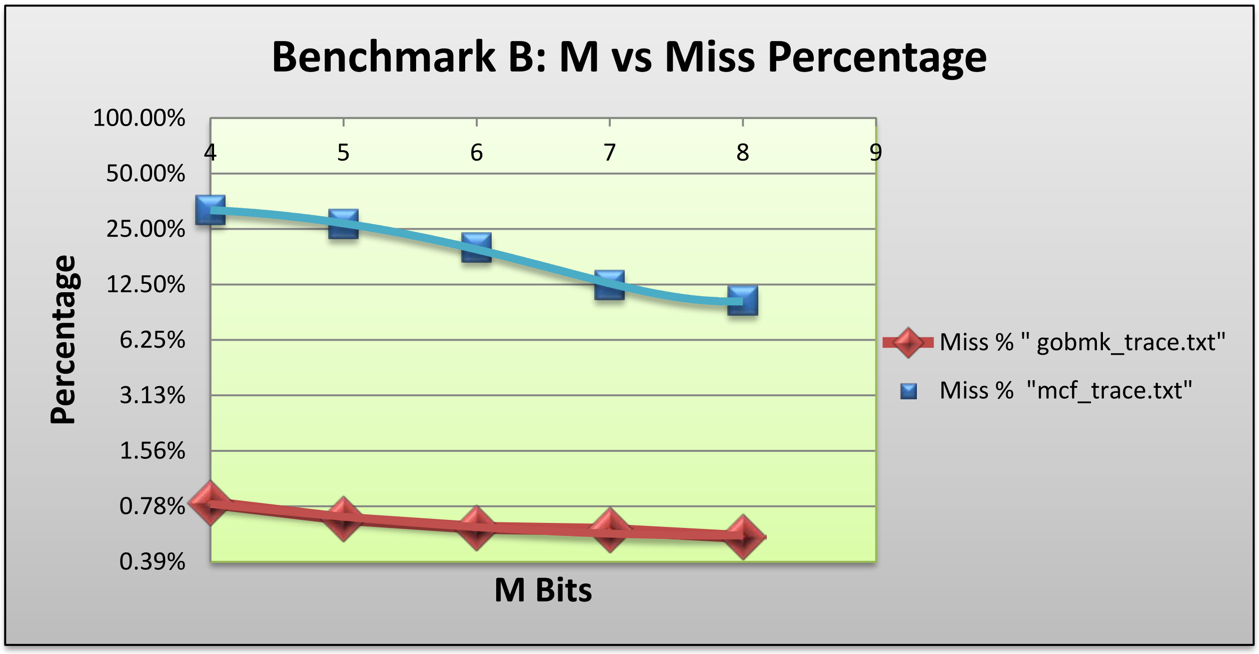
Part A Benchmark, M = 4		
N	Miss % " gobmk_trace.txt"	Miss % "mcf_trace.txt"
0	0.691%	23.76%
1	0.773%	24.71%
2	0.870%	26.86%
3	0.864%	29.36%
4	0.816%	31.72%



The value N determines how many bits of the global history shift register to use. The greater the N the more correlation there is between the branch performance and how the previous branches resulted. We can infer that as N increases for the MCF program, there is less accuracy because the branches are not that related to each other. Using the global history is not helping predict the next outcome. However, for goBMK, there is correlation between the branches. The result of one branch depends on the results of the previous branches. Notice, how using more than half the bits as N bits, increases accuracy.

The following table is data gathered using the gobmk_trace and the mcf_trace files to analyze how the miss prediction ratio of these workloads changes with increasing M bit correlation. N is fixed at 4; M is increased from 4 to 8.

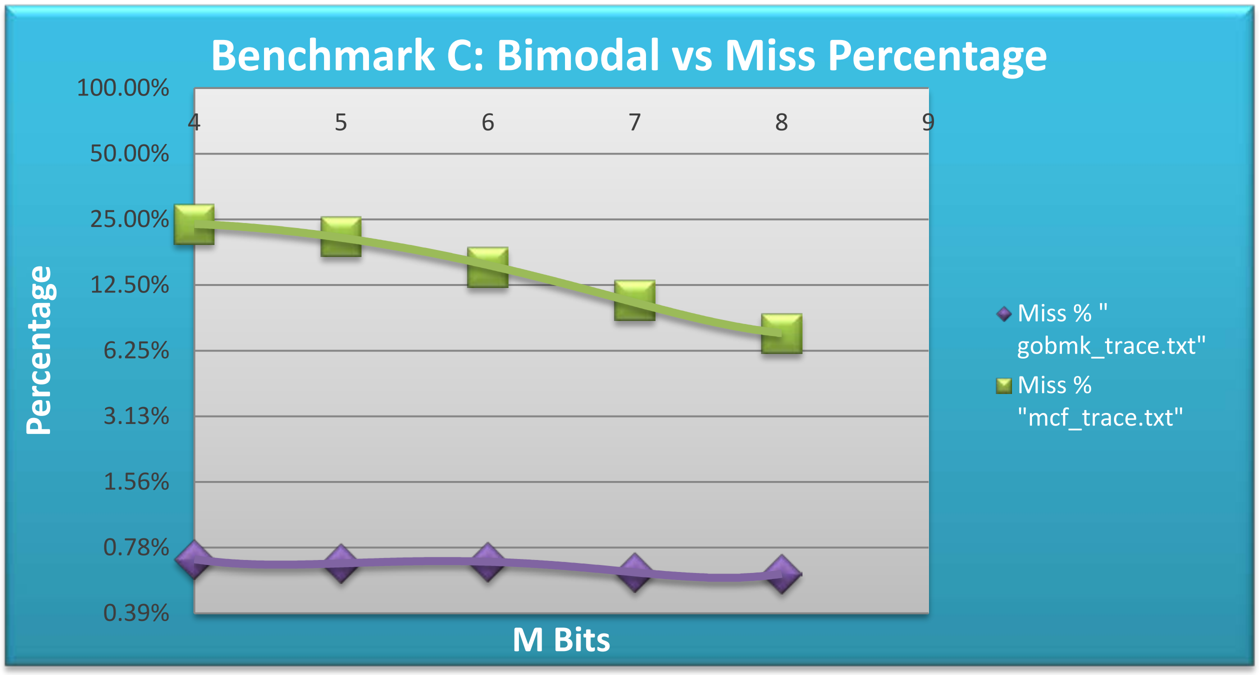
Part B Benchmark, N = 4		
M	Miss % " gobmk_trace.txt"	Miss % "mcf_trace.txt"
4	0.82%	31.72%
5	0.67%	26.56%
6	0.60%	19.81%
7	0.58%	12.40%
8	0.53%	10.22%



The value M determines how many bits of the Program Counter address to use. The greater the M the more correlation there is between the branch's previous state and the next prediction. There is also increased accuracy because the prediction table is larger and the branches are distinguished more, so there is less ambiguity when making a prediction. The larger the M the less likely it is that two unrelated branches map to the same index. MCF is a program that performs better when the predictions are made based more on actual branch history than global history. Notice how miss ration decreases when there is less correlation between PC bits and GHR bits for both programs, but more so for MCF.

The next table is data gathered using the gobmk_trace and the mcf_trace files to analyze how the miss prediction ratio of these workloads changes with increasing M bit as a bimodal predictor. N is fixed at 0; M is increased from 4 to 8.

Part C Benchmark, N = 0		
M	Miss % " gobmk_trace.txt"	Miss % "mcf_trace.txt"
4	0.69%	23.76%
5	0.66%	20.83%
6	0.67%	15.07%
7	0.60%	10.63%
8	0.59%	7.47%



When the prediction is bimodal, the history of other branches is ignored when making predictions. The greater the M the more correlation there is between the branch's previous state and the next prediction. So increasing M, when GHR bits are ignored provides more accurate results for MCF, where there is little correlation between Global Branch results and lackluster performance for goBMK because, as determined previously, goBMK performs better when a little correlation between a branch and its neighbors is considered.

Using the data gathered by these benchmarks, the assumption can be made that the best configuration for the program that requires these traces would be a bimodal predictor with the largest feasible number of M bits, for the MCF program. GoBMK is pretty accurate, even with a small number of M bits. The best performance for a trace such as this would at least use half as many GHR bits as PC bits.

To summarize, no branch predictor configuration is perfect. Increasing the number of PC bits used, as large as possible is a benefit, but the diminishing returns have to be weighed with the cost of a large memory overhead. Increased Global History Register bits used also helps but it depends on the program. Whether or not to use a GShare predictor such as this really depends on the uses cases for the system, but simulation using the known PC traces and results will help pin point a hardware design that will provide the most accurate predictions.