Quinn Magendanz

Lab 2 Questions

1. **It’s absolutely pitiful to code like that.** Your TA has written the lab handout in a slightly suboptimal way, in terms of simulation speed. Figure out the inefficiency and fix it. Explain your change. Does it make a big difference in speed?

The inefficiency is that the handleBranch function increments the correct global counter variable using nested if statements. Adding these branches is slower than just incrementing each of the global counters simultaneously (since that can be parallelized and there is no need for speculation).

takenCorrect += prediction & direction;

takenIncorrect += prediction & !direction;

notTakenIncorrect += !prediction & direction;

notTakenCorrect += !prediction & !direction;

This makes a very small difference in speed since, relative to the bottleneck of the program (the predictor), a few extra branches are of little consequence.

1. **Hardware? Don’t talk about hardware.** Explain the operation of your implementation. Detail the general algorithm that you used and explain why you chose it. Why does the algorithm that you finally chose outperform the other algorithms you tested and those discussed in class? Describe how you allocated the storage bits of your branch predictor. Don’t forget to cite your sources.

The most optimum predictor that I found was the local history predictor. The local history predictor is made up of a table of shifting registers and a two-bit predictor. On each branch instruction, the address is used to index into the local history table and look at the branch history for that branch. The branch history is then used to index into the two-bit predictor to see how that pattern should decide the speculation. My implementation uses 12 bit local histories, a 2048 entry local history table, and a 4098 entry 2-bit predictor for a total of 32768 bits of static memory. Keeping a 12-unit history for each branch gives sufficient context to predict correctly with 95% accuracy.

This predictor outperformed the other predictors that I tried because the test suite contains relatively small programs, so the local history table is able to store most/all of the programs’ branches in its tables with few collisions, even with the static size constraints. Other predictor implementations such as tournament predictors used up more space on additional tables, leaving insufficient space to store enough data to beat the local history predictor. Other predictors, such as gshare, do not have enough specific detail about each branch to be as accurate.

1. **Are you kidding me? Hardware?** We gave you unlimited computation to develop your branch predictor. Could your implementation be built into a processor? Explain why or why not. Discuss which elements of your algorithm would cause problems if implemented in silicon.

My implementation can definitely be built into a processor. It will only take a table of registers. The way I made the predictor in C++, however, allows for flexible sizing of registers, which is not possible in hardware.

1. **I’m just hoping we can implement a software model.** The branch predictor you implemented is likely to be missing several of the elements of a real branch predictor. For example, in a real machine several predictions might be made before an update occurs. Why might that happen? Explain the consequences of this on your predictor in terms of impact on prediction and hardware changes that might be added to address that impact. Don’t neglect the impact on misspeculation recovery.

Pipelining may cause consecutive branch predictions to be made before the correct path is identified and the predictor is updated. We can assume that not having one update will not detrimentally impact the branch prediction rate of another branch in the next few instructions. However, every mispredict, we will need to keep the resulting computation separate from the commit, kill the instructions following the mispredicted branch in the pipeline, and restore the state to that of the state following the branch.

Exceptions will also require recovery. To handle this, we should have a lazy update approach with the branch predictors. We will only commit the predictor updates when the branch is done and no longer has a chance of misspeculation or exception.