Patrick Trinkle

CMSC611

Homework

Fall 2008

Collaboration with Mike Corbin to clarify what it is some of these things are asking.

3.11

a – For a branch-prediction buffer implementing a given type of predictor, what characteristic of any program guarantees that prediction accuracy as a function of increasing buffer size must eventually become constant. Hint: examine the contents of and compare the prediction accuracy of branch-prediction buffers of different sizes on a simple code fragment such as the following…

Since there’s an entry for each branch, and whether it was taken or not, and possibly some history there to whether it was taken previously by some pattern; the more entries you have the more branches you can handle—but after a point the program will only have so many branches, and if the buffer gets beyond this many of the entries; the extra entries are wasted; so you lose the benefit after a bit.

This can be seen when you examine the increase in accuracy going from x entries to some much larger value (infinite). If you have more entries than branches, then the benefit remains the same.

b – Within the precision of the measurements, the SPEC89 benchmarks nasa7, tomcatv, and gcc were the only programs to have less branch misprediction when buffer size increased from 4096 entries to infinite. Based on the answer to part (a), what quantitative measure can you infer about the machine instruction count of the executable codes for these four benchmarks?

Well, nasa7, tomcatv, and gcc are only 3 benchmarks; so I’ll just work with that.

If their mispredictions decreased when the branch-predictor buffer increased form 4096 to infinite we can infer that there is a significantly larger amount of branch instructions in these benchmarks than the others. This makes sense because gcc compiles code; which has to perform many loops to scan through the code as well as look for possible optimizations.

c – Can you infer anything similar to the result in part (b) about the instruction counts of the other seven benchmarks?

The other seven benchmarks must have fewer branch instructions and therefore would see little if any improvement from having an infinite amount of prediction buffers.

d – How might an optimizing compiler improve prediction accuracy for the other seven benchmarks in part (c) and when would this be and not be possible?

An optimizing compiler might actually be able to code up sections into groups that have few branches; so few entries will be necessary; because normal instructions don’t tend to stall much and are very fast; whereas mispredicting a branch can cost quite a lot. This wouldn’t be possible if the program required a lot of iterations.

For 5.4; a-e, see attached handwritten sheet.

f – If you want to make your system run faster, which part of the memory system would you improve? Graph the change in overall system performance holding all parameters fixed except the one that you’re improving. Parameters you might consider improving include L2 cache speed, bus speeds, main memory speed, and the L1 and L2 hit rates. Based on these graphs, how could you best improve overall system performance with minimal cost?

I calculated the improvement from changing the following: decreasing L2 miss rate, decreasing L2 initial access time, decreasing Main Memory initial access time, increasing L2 bus speed, and increasing Main Memory bus speed. The biggest improvement was from decreasing the L2 miss rate. Good returns also came from decreasing the initial access times. Increasing the bus speed was nice, but you still take quite a hit from the initial access latencies. Decreasing the miss rate might be a bit difficult and more expensive, by adding more of a buffer, and increases the footprint of the die. Increasing the bus speeds will increase the heat signature, but might be relatively inexpensive. A good way to improve performance of this system would be to reduce the initial access time. Might also be relatively inexpensive.

Note: The X axis is speedup % (not pure speedup). Also, some of the values have been rounded.