Final Project Report

Jacob Rumpf and Zach Langer

Q. If you are building a processor and have to do static branch prediction (meaning you have to assume at compile time whether a branch is taken or not), how should you do it? You can make a different decision for branches that go forward or backward.

A. I would always assume that the branch is taken. The result showed that it is more common than not that a branch is taken. This applies to both forward and backward branches. Both take the branch about 80% of the time. This means that the CPU wouldn't have to spend as much time flushing out wrongly predicted branches.

Q. If you are building a 256-byte direct-mapped cache, what should you choose as your block (line) size?

A. If I were building a 256-byte direct-mapped cache, I would use a 32-byte block size. This is because the 32-byte cache consistently resulted in the highest hit rates amongst the various block sizes. This means that it will consistently perform faster than other options since it will spend less time accessing more expensive memory location. In general, it appears that moderately sized block-sizes are better than extreme ones; CPU's with maximum block size(256 bytes) or maximum number of entries(4 byte block size) performed the worse than those that had a decent number of entries and a small block size (16,32,64)

Q. What conclusions can you draw about the differences between compiling with no optimization and -O2 optimization?

A. Compiling with optimizations greatly reduces the amount of dynamic instructions. Memory read/writes and Register read/writes all saw a decrease of over 60%. This drastically increases performance because far fewer operations need to be executed. There was no drastic difference between the percent of branches taken versus not taken. There were some differences in the cache hit rates too. For the best block-sizes, the cache hit rate was slightly better for the unoptimized code. For the worst block-sizes, the cache hit rate was greatly better for the unoptimized code. However it can be concluded that this is negated by the benefit of the reduced instruction quantity because the run time of the optimized code was notably faster.

Important Notes:
Our cache implementation worked perfectly except for the blocksize of 256. For this block size alone, we had incorrect results for shang (but good results for fib and bytetest). We could not pinpoint the source of the error, which occurred even when we tried other cache implementations, but we were always within <1% error.

We also had trouble running the optimization level 1 version of Shang. The program appeared to freeze consistently at line 4974 when it pushed into the stack. Other students had a similar problem that was confirmed by Professor Lupo on Piazza (posted as "Bad Alloc"), so we chose to leave our program as is. If this is just an issue of the program running out of memory when the server is high on users, then it may not appear during instructor grading.