RETROSPECTIVE:

A Study of Branch Prediction Strategies

James E. Smith

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
jes@ece.wisc.edu

In 1979, I took a leave of absence from the University of Wisconsin to work at the Control Data Corporation in the Twin Cities. In truth, my intention at the time was to abandon an academic career; I felt awkward teaching computer design — never having worked on the design of a computer. And I was doing research in fault-tolerant computing as a theoretical enterprise, which seemed contradictory.

The project I joined at CDC was developing the high end processor of a new product line — the Cyber 180 series. The processor was code-named "Theta", and would become the Cyber 180/990 when it was officially announced. The Cyber 180 architecture was a 64-bit virtual memory system with all the bells and whistles that were fashionable at the time. It was built around register-register instructions, but also had a number of complex instructions to support a heavy duty protection system, business data processing, and a memory-to-memory vector instruction set. With the same hardware, it also had to support the older 60-bit Cyber 170 instruction set, and switch seamlessly between the two modes — potentially at the procedure call level.

My manager was Jim Stockard, and I took technical direction from Ron Hintz, the chief architect. At the time I joined, the project had been underway for awhile, and was beginning to fall short of performance goals. My main responsibility was to carry out performance studies and suggest performance features to the designers.

The technology was based on ECL gate arrays — about 200 gates per chip. It had been developed for the Cyber 200 supercomputers, which ran at a 20 ns clock cycle. The Theta clock cycle was 16 ns, however, and in retrospect, this was probably a touch too aggressive. The complex instructions, aggressive clock cycle, and the packaging technology led to very long pipelines. It took about six pipe stages to fetch an instruction, decode it, and generate what were called "micrands", ready to be issued (this process was very similar to what is done today in aggressive x86 implementations). Instructions issued in order, at most one per cycle.

With a pipeline this long, conditional branches were a performance problem. The original pipeline design used a very simple static prediction strategy where all branches were predicted taken (or maybe not taken — I don't recall which). Instructions following a branch had to wait for the branch to be resolved before they could issue.

During discussions with Tom Lane, the designer responsible for part of the instruction pipeline, the possibility of branch prediction came up. Tom had done a small study, using a cache-like table with single-bit entries. Tom also pointed me to Shustek's thesis [1] — a real classic that suggests a number of static prediction strategies.

I considered several prediction methods, most of which are given in the paper. It was evident that dynamic prediction was better than static. For studying performance, most of the benchmarks I was using (kernels, actually) were heavily loop-dominated. It was evident that using a single history bit led to two mispredictions at loop terminations. Solving this problem with a two-bit saturating counter seemed like a good thing to do, and simulations showed a performance advantage. The parts available for implementing the predictor table were 16 x 4 register file chips, so a two-bit table entry cost no more than a one-bit table. I studied larger counters, but two bits worked better than three or four.

Hardware was at a premium, and early on I realized that the table could be indexed like a cache, but tags were unnecessary — there was already a way of recovering when mispredictions

were made. I believe the final implementation consumed four 16x4 register chips (a 64 entry table) and a single gate array for updating. The counters described in the paper used two's complement arithmetic with the sign bit being used to indicate taken/not taken. However, the final implementation used the now-common 0-to-3 integer counter.

When I joined the Theta project, I took over the performance simulator from Paul Higgins; it had been developed by Paul and Dick Olson of CDC in Canada. The simulator was written in ASPOL and was trace-driven. We had a pile of 9-track trace tapes — including both Cyber 170 and 180 codes. The benchmarks were FORTRAN kernels and one nasty system benchmark called the SWL-profile ("swill" for short).

The benchmarking in the paper was very basic by today's standards — yet the paper made it into the "Performance Analysis" session at the conference. Many of the kernels were only a few lines of code. I included the more difficult-to-predict kernels, but even those were pretty small. In doing the study, the SWL-profile was also simulated, and it showed the biggest performance improvement, but these results were not included in the paper.

In conjunction with branch prediction, the actual Theta design also included a scheme for "conditional issue" ("speculative execution" today). As mentioned above, instructions issued in order, and branches took 5 cycles to resolve (only some simple integer instructions were faster). When a branch issued it reserved all result bus slots up until the time it finished. Then, following instructions could conditionally issue and start executing prior to the resolution of the branch. The reserved bus slots inhibited any conditionally issued instruction from writing its result register before the branch was resolved. A branch misprediction would invalidate all conditionally issued instructions and start over.

Overall, with branch prediction and conditional issue, the FORTRAN benchmarks improved by about 5%, and SWL about 10%. The SWL improvement was the real clincher, because the Theta project was furthest behind in SWL performance goals.

This work was done in the context of a development project, so there wasn't much time for academic investigation. The only exception is the small study I did using larger counters and assigning confidence levels depending on the count values. The concept was to speculate more aggressively when the confidence level was higher. I continued to carry this idea around for about 15 years before working on the study that appeared in Micro-29 [2], two years ago.

This was my first ISCA paper; the conference was held in the Twin Cities, which provided additional motivation for writing the paper. One of my clearer memories of the conference is the invited speech Jim Thornton gave. I also recall meeting David Kroft, another CDC employee who developed a non-blocking cache for a different Cyber 180 machine being designed in Canada.

A number of Cyber180/990 machines were eventually built and sold; I believe several went to Europe, where CDC did a good business at the time.

References

[1] L. Shustek, *Analysis and Performance of Computer Instruction Sets*, Ph.D. Thesis, Stanford University, 1978.

[2] E. Jacobsen, E. Rotenberg, J. E. Smith, "Assigning Confidence to Conditional Branch Predictions", *29th Int. Symp. on Microarchitecture*, Dec. 1996.