RETROSPECTIVE: Selective Value Prediction

Brad Calder*, Glenn Reinman†, Dean Tullsen‡

*Google, bcalder@google.com
†UCLA Computer Science, reinman@cs.ucla.edu
‡UCSD Computer Science and Engineering, tullsen@ucsd.edu

I. Technical Backdrop

The mid 1990s saw the introduction of the first out-of-order microprocessors. Prior to their introduction, the execution cost of an instruction was directly proportional to the latency of that instruction (modulo a cycle or so due to superscalar effects). I.e., a single cycle instruction added about 1 cycle to execution time, a 50-cycle load miss added about 50 cycles. In out-of-order machines, that was no longer true. Many instructions had no impact on execution time (e.g., if they were executed a few cycles slower, performance wouldn't change), others would have their full latency exposed, and some fell in between. This high variance in importance, or criticality, was a new phenomenon. However, the processor pipeline had no ability to distinguish between instructions whose latency impacted performance and those that did not.

In fact, in a theoretical machine with infinite capacity for parallel execution, only instructions on a single dependence path matter to execution time. All other instructions would have no impact. Even better, with register renaming, only true data dependencies contributed to that path. Our work introduced the idea that it would be valuable to know which instructions were on that critical path.

While we, and others, would pursue that concept in much more general ways, in this first work we found the relatively new idea of value prediction to be a perfect testbed for focusing in on the critical path. This is because, rather than just accelerate an independent instruction, value prediction had the ability to break true data dependencies.

Value prediction broke true data dependencies by predicting the result of an instruction's computation before the instruction finished execution. There was significant gains possible on a correct prediction of the right instructions, but prediction was not without its risks. An incorrectly predicted instruction would not only waste power and processor utilization, but it would also require architectural state recovery to return to correct execution.

Value prediction had tremendous potential - but would it be beneficial to value predict every instruction? Imagine a dependence chain comprising a load instruction, followed by an add instruction, followed by another load instruction, and ultimately followed by a store instruction. Predicting all four of these instructions would have diminishing returns - and worse, a single misprediction could undue any benefit from correctly predicting the other instructions. Conventional wisdom held that long-latency loads were the highest priority for prediction, and in fact predicting the first load allows the two loads to execute in parallel, providing a large gain. However, correctly predicting the intervening add instruction also allows the two loads to proceed in parallel, with the same benefit. Thus, the add's importance is not derived from its latency, but its position on the critical path.

II. Contributions of the Paper

Most of the work that followed the initial papers on value prediction concentrated on more accurate predictors. This paper took a very different approach. We started with the assumption that we had limited resources, in particular limited prediction table size. As a result, we needed to be selective about which instructions actually used, or wrote into, the value tables. Additionally, we saw value in limiting the number of predictions outstanding in the machine – too many, and the likelihood of misprediction would quickly nullify any gains from the correct predictions. Viewed differently – a prediction of an instruction not on the critical path provides no performance gain, but a misprediction of that instruction gives performance loss.

The paper looks at various ways to filter instructions writing into the value table, or consuming table entries to make a prediction. These include the most standard approaches (all instructions, all loads), but also includes skipping instructions whose destination is not sourced within the processor's instruction window, and several instantiations of "on the critical path". One version considers any instruction on the longest path within the current window of unexecuted instructions (i.e., decode to issue). Another marks any instruction at the head of the longest path (of a certain length in cycles) at any time in execution. Once marked, an instruction becomes a candidate for value prediction each time it enters the pipeline.

One interesting early result was that even in an infinite table with oracle confidence – so there is no value to selectivity either in placing values in the table or in making predictions (since there are no mispredicts) – applying a path based filter provided essentially the same performance as having no filter. This validated the contention that there was virtually no gain to value predicting instructions not on the critical path.

In the presence of limited table size and possible mispredicts, the value of selectivity comes into play. In that case, filtering instructions (either those that write to the value table or those that use those values) with path-based criteria consistently outperformed filtering for long latency (i.e., loads). Note that this paper did not speculate on how to practically detect the criticality of instructions; that was left to later work.

The other issue we focused on addressing was how to have confidence in the prediction. We explored value prediction mechanisms that were based on a stride pattern (a fixed delta between subsequent values) and based on a repeating pattern of values (a context predictor). Other prediction mechanisms could certainly be employed, and could benefit from the techniques proposed in our paper.

In addition, we designed a set of confidence counters: structures that could predict the success of the value predictors for a given instruction based on historical data. Even in the face of infrequently predictable instructions, the confidence counters could accurately determine when to use the prediction - the crucial insight was that even if the value predictor could not capture a particular access pattern with high accuracy, as long as we could accurately capture when the value predictor would be right or wrong in its prediction, would help to avoid the misprediction penalty. Prediction is not a requirement for functional correctness, and judicious application of prediction is crucial in mitigating its deleterious effect on performance.

III. Impact of the Paper

This paper introduced the idea that if we could identify instructions that were on the critical path, we could treat them differently and make better use of limited pipeline resources. It also identified the fact that limited instruction window size (as defined by any of several specific structures like the instruction queue, reorder buffer, physical register file, etc.) necessitated a more local view of the critical path. That is, an instruction on the theoretical global critical path could be prevented from entering the window because a prior instruction has not yet executed (rendering the instruction queue full) or committed (rendering the reorder buffer full). Thus, that instruction and instructions it depends on are now part of the critical path.

These two concepts were critical to an entire research area devoted to identifying those instructions on the critical path – critical path prediction. Our paper [5] introduced the idea of critical path prediction, exploiting clues within the pipeline to identify instructions likely to be on the critical path. Later that year, Fields and Bodik [2] sought to carefully track the critical path itself, accounting for all of the structural hazards that could create artificial dependencies and place an instruction on the critical path.

A number of papers followed from both groups and others that continued to refine this idea of effectively tracking the critical path, and finding new ways to exploit that information in the pipeline.

IV. Current State

Value prediction has been used to provide performance gains as a feedback-directed optimization using Value Profiling [1] and evaluating the potential path for optimizing the critical path [3], [4], [6] of the program to improve execution.

Value prediction as a hardware optimization was a promising technology that never saw its way into real processor designs, likely a victim of the architectural ”energy crisis”, as power and energy became first class design constraints. Under those constraints rampant speculation, just because the processor had nothing better to do, fell out of favor. To our knowledge no processor takes significant advantage of the variance in instruction criticality, and that variance is higher than ever in today’s large-window processors.

We still consider this a missed opportunity. Value predicting the critical path was always the best target for this, but it was far from the only one. Some of those potential optimizations decreased speculation or power (e.g., slower functional units for instructions with slack) – optimizations that still match up with today’s design priorities.

References

[1] B. Calder, P. Feller, and A. Eustace, “Value profiling,” in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, p. 259–269.

[2] B. Fields, S. Rubin, Rastislav, and Bodík, “Focusing processor policies via critical-path prediction,” in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, 2001, p. 74–85.

[3] R. Muth, S. K. Debray, S. Watterson, and K. D. Bosschere, “Alto: A link-time optimizer for the compaq alpha,” *Software - Practice Experience*, vol. 31, no. 1, p. 67–101, jan 2001.

[4] A. Srivastava, A. Edwards, and H. Vo, “Vulcan binary transformation in a distributed environment,” 2001.

[5] E. Tune, D. Liang, D. M. Tullsen, and B. Calder, “Dynamic prediction of the critical dependence path,” in *Proceedings of the 7th International Symposium On High Performance Computer Architecture*, 2001.

[6] E. Tune, D. Tullsen, and B. Calder, “Quantifying instruction criticality,” in *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, 2002.