15740 Discussion Topics:
Branch Prediction

Alex Nizhner Yang Gu

October 8, 2003

In the rest of this document, we summarize the following papers on branch prediction in superscalar processors:

- Paper 3. Timothy Heil, Zak Smith, and James E Smith. Improving Branch Predictors by Correlating on Data Values, 1999

1 Paper 1

The authors point out that, despite a proliferation of various hardware and software branch prediction schemes, no means for the structured analysis of their accuracy exists. In other words, there is a general lack of understanding in the community of why certain schemes are better than others and how they could be improved further. The authors propose to address this problem via the introduction of a generalized theoretical framework for the analysis of branch prediction schemes, the internals of which they subsequently develop.

For the purposes of the present discussion, a branch prediction scheme is defined as a comprehensive mechanism that partitions a stream (i.e., a trace) of branch executions into classes and assigns each class to a unique predictor. All known branch prediction schemes, then, can be specified in terms of a set of predictors—simple mechanisms used for predicting the direction of a given branch instance, and a divider—the strategy under which the branch execution stream is partitioned for assignment to distinct predictors. (The authors note that the goal of partitioning an execution stream is to make the isolated sub-streams more predictable individually than the unified original stream.) The GAs scheme illustrated in the Patterson and Hennesy text, for example, is characterized as follows: the divider strategy is specified as the assignment of each
branch execution to a predictor identified by the concatenation of the low-order bits of the branch PC and the branch history vector; the set of predictors is simply a table of traditional 2-bit saturating counters.

The authors compare and contrast several existing schemes in terms of divider parameters of global history representation, aliasing, and cross-procedure correlation, and the predictor parameter of adaptivity. The results serve to reinforce and formalize the seemingly intuitive notions of the adverse effect of aliasing on prediction accuracy, the stronger correlation of path history to branch direction compared to pattern history, the benefits of cross-procedure correlation in static schemes, and others.

2 Paper 2

Previous work on categorizing and characterizing predictors has been based on the high-level constructs form earlier research on branch prediction mechanisms. We are therefore losing the chance to find fresh new components of predictors. Another drawback is that while we know many good building blocks for branch predictors, this may not be the case for other prediction problems. This paper introduces an algebraic-style description language for predictors and also a parser that can understands the predictor and a variety of functions. Starting from low-level primitives and use genetic programming to stochastically search the design space, the model allows for automatic manipulation, including generating simulators and automated synthesis.

This method does better job than most of the well-known human discovered predictors except gshare. As a result, a lot of interesting components were generated and a relatively simple indirect jump target predictor significantly better than existing ones was created.

This paper presents a new idea on building predictor starting from low-level and it shows its advantage over some of the existing predictors. However, the logical complexity of the automatically-created predictor brings inefficiency. It may not be an easy work to manually identifying the pieces of the solution to resolve the "introns". We still need to find way of better encapsulating good offsprings.

3 Paper 3

The authors point out that all known branch predictors use control information—that is, some combination of PC bits and branch history—in order to make predictions. Examples include schemes such as GAs and gshare which combine a global branch history string with the low-order bits of the branch PC in order to correlate the branch being predicted to past program behaviour. Whereas
such branch-history prediction is often quite effective, there are program constructs that yield unusually high misprediction rates—consider frequent loops whose counts exceed the maximum length of global history string, or massive switch statements indexed by successive unrelated inputs. In the latter case, the outcome of a given branch may not be at all correlated to the program’s history.

The authors suggest correlating on data values in addition to branch history in order to reduce the misprediction rate in such cases. The chief observation they make is that the data on which a code fragment operates frequently encode more information useful to a branch predictor than branch history—for instance, a loop counter that is nearly zero is a reliable indicator that the current iteration of the loop is the last. Such data are typically readily available in the operand registers of compare-and-branch instructions and the like. The authors then discuss the implementation of the Branch Difference Predictor, a scheme that correlates branch directions to the past values of the difference of branch operands.

In addition to a conventional predictor such as gshare which is expected to correctly predict a large proportion of branches, the BDP employs a cache-like structure termed the Rare Event Predictor for “difficult” branches. The per-branch value history list is used to validate entries in the REP; that is, when the base predictor first mispredicts, the branch instance and the current value history are recorded as the source of an authoritative prediction should the branch re-occur with the same value history.

It turns out that a single difference value per branch is typically sufficient to achieve satisfactory prediction rates. The Branch Difference Predictor is shown to achieve significantly better accuracy than conventional schemes. Correlating on data values in this fashion is the first step toward capturing information other than the PC and branch history to raise prediction accuracy; it is conceivable that other program state can be captured as well, improving the performance of the Branch Difference Predictor.