The Simulation-based Gold-Standard framework for verifying HDL branch predictors

Katy Thackray[1], Karl Mose[1]

[1]Department of Computer Science, The University of Cambridge

Abstract

Faults in a branch predictor's programming will not cause ISA tests to fail or the processor to deadlock, instead they are likely to cause subtle effects, such as underperformance. Exhaustive testing is near infeasible and low impact deviations from the desired functionality are likely to go undetected. Extremely rare cases could cause a context-sensitive branch predictor to strongly underperform. This paper presents a framework for testing branch predictors written in optimized HDL for RISC-V processors. The SGV framework can functionally verify a HDL branch predictor against a high level gold standard branch predictor efficiently using billions of trace instructions. A deviation from the outputs of the two models allow for an instant halt and internal state can be logged to trace the deviation. Verifying on a diverse set of traces can ensure with high confidence that the HDL model is functionally equivalent to the gold standard model. The framework is used to verify a TAGE based based branch predictor in Bluespec SystemVerilog.

Introduction

Branch predictors are ubiquitous and a necessary component in high-performance processors to enable high levels of IPC [1] (instructions per cycle). Branch predictors are an evolving field, and modern branch predictors can be highly complex.

Generally, state of the art branch predictors have been published in the form of a high level software programming language implementation. These languages provide the benefits of being able to describe behaviour at a general level without worrying about low-level implementation details, as well as being easily parameterisable. The translation of these to hardware is non-trivial [2]. Despite this, relatively little work has been done on verifying the behaviour of branch predictors implemented in hardware. Often, good performance across a series of benchmarks is considered enough to verify the performance of a branch predictor. However, small errors in implementation can lead to subpar performance, and even small decreases in accuracy can lead to measurable differences in IPC [1].

This paper proposes a more principled approach to developing branch predictors, which we term the Simulation-based Gold-standard Verification framework (SGV) its main purpose is to test for exact functional equivalence between a high and low level model. It makes two extensions. 1) It allows two branch predictors to run in lockstep to compare their output, and 2) It introduces an interface for connecting Bluespec-based branch predictors to the high-level simulator. This enables easy comparison between branch predictors implemented in an HDL and in software. By efficiently bridging the HDL model to the high level simulator, billions of instructions can be used to test at a high volume. The power in this method allows for deviations to be rapidly spotted. A case study is given where SGV is used to debug an implementation of TAGE [3] against a pre-written gold standard model for the RISC-V Toooba [4] processor. We propose that this framework will be greatly beneficial for testing branch predictors in RISC-V processors.

Importance of Correctness

Incorrect branch predictors will not cause an erroneous implementation of the ISA, instead less accurate predictions will be provided and therefore a lower IPC. As the processor will continue to function if a branch prediction is incorrectly implemented, rarely-seen deviations between the predictor's behaviour and the intended model may seem unimportant, however, the context of the predictor could be incorrectly altered such that many future predictions will be affected. Unfortunately, branch predictors can be incredibly context-sensitive, meaning generally low-impact undetected issues in the programming could potentially be disastrous in some contexts. As an example, one of the most widespread branch predictor designs, TAGE, is highly sensitive to small deviations in the branch history. Even if faults don't propagate, as discussed in [1], a small increase in mispredictions can noticeably decrease IPC.

The SGV framework

We present a method for testing predictors written in BSV with a focus on the pure prediction functionality,

rather than cycle accuracy. The SGV framework intends to find divergence in the prediction function by sequentially asserting outputs are the same under billions of instructions, with functionality to log and trace bugs triggering the assertion. While tree structures, folded histories and pipelining may be used in the actual hardware implementation of the predictor, the gold standard model does not require this. A key insight is that a gold standard model is not just simpler because it can be written in a higher level language, but particularly it doesn't need to be optimised and can instead focus on a purely correct implementation of the desired functionality.

The SGV framework bridges the gold standard model and the HDL model using a simulator with the functionality to read in traces, request predictions from both models, and send subsequent updates. SGV is implemented on top of Champsim[5], a trace-based simulator, to verify the models against each-other on each prediction. With SGV, Champsim simulates and compares multiple branch predictors. It compares their outputs and halts on a deviation.

Using the BSV's BDPI library a Bluesim simulation incorporating the branch predictor can be bridged to the Champsim prediction interface using an OS FIFO, allowing for predictions to be run on both models. Because Champsim does not simulate wrong paths, branch instructions could be buffered to greatly improve efficiency making the performance penalty of the SGV framework almost negligible.

Using the SGV framework in development

The SGV framework is effective in the later stages of development after some testing has already taken place directly in the HDL. For our TAGE case study, we used numerous testing traces. To use the SGV framework it was necessary to implement a C++ based version of TAGE that is more faithful to a hardware-based implementation such as implementing an LFSR shift register for random number generation.

Faults found using the SGV framework

Using SGV, we were able to quickly find divergence in behaviour between the software and hardware based predictor. A few thousand predictions into the simulation the framework detected a fault which incorrectly left shifted the PC passed into the training data. Another divergence revealed that the hardware implementation, when choosing the first table as the provider, would incorrectly also mark the first table as the alternative table.

A subtle bug was found 85,000 branch instructions into the simulation. The root cause was the fact that the BSV TAGE predictor was decrementing the useful counter of all entries above *and including* the provider entry, when it should only be the entries above the provider entry. This issue caused a divergence in the state around 20,000 instructions into the simulation despite manifesting in a wrong output 85,000 instructions in. By searching for specific indices in the logs we could trace back the allocations and uses of this entry to find this cause. It is highly likely this small discrepancy would've remained undetected.

Future work

As the framework is an extension of the Champsim prediction interface, it's currently limited to an unrealistic assumption that wrong path instructions are not predicted on. The SGV model also does not model other possible issues in a pipelined processor such as out of order updates and makes the assumption of immediate updates. The SGV framework could be extended to include more realistic assumptions.

Conclusion

In this work, we have introduced SGV, a framework for verifying hardware-based branch predictors against gold-standard models implemented in software. We demonstrate the effectiveness of SGV by verifying a Bluespec System-Verilog implementation of the TAGE branch predictor for Toooba, an open source RISC-V processor, showing how it can be used to incrementally identify and correct implementation bugs.

References

[1] Chit-Kwan Lin and Stephen J. Tarsa. "Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions". In: *2019 IEEE International Symposium on Workload Characterization (IISWC)* (2019), pp. 228–238. URL: https://api.semanticscholar.org/CorpusID:195069130.

[2] Alex Saveau. "Branch Prediction in Hardcaml for a RISC-V 32im CPU". In: *ArXiv* abs/2312.10426 (2023). URL: https://api.semanticscholar.org/CorpusID:266348507.

[3] André Seznec and Pierre Michaud. "A case for (partially) TAgged GEometric history length branch prediction". In: *J. Instr. Level Parallelism* 8 (2006). URL: https://api.semanticscholar.org/CorpusID:6159849.

[4] *Toooba: Open-source RISC-V CPUs from Bluespec, Inc.* https://github.com/bluespec/Toooba.

[5] Nathan Gober et al. "The Championship Simulator: Architectural Simulation for Education and Competition". In: *ArXiv* abs/2210.14324 (2022). URL: https://api.semanticscholar.org/CorpusID:253117130.