Review of 'Branch Runahead: An Alternative to Branch Prediction for Impossible to Predict Branches'

Luc Jones *lxj19*, AbdelQader AlKilany *aqa20*

November 2022

1 Introduction

In this paper, the authors offer a solution to improve branch predictors called Branch Runahead. This new method is implemented in hardware only and serves to improve performance by computing most branch predictions ahead of time while keeping the area and energy costs of the unit low.

2 What problem is being addressed?

In most branch predictors, the prediction is entirely based on the historic outcomes that this branch has taken[1]. This paper attributes this to the stagnant progress made in the reduction of branch mispredictions in regard to data-dependent branches, and claims this is the cause of modern branch predictors being history-based. The paper claims that data-dependent branches whose outcome is based on a value recently loaded from memory, are not correlated with any previous branch predictions. The paper also says that these data-dependent branches bottleneck the rest of the processor - slowing it down as the processor must pay the misprediction penalty very often for these types of branches.

The paper also addresses the problem that other solutions such as pre-computation are ineffective as it requires a trade-off between timeliness and complexity as they focus on heavy-weight, compile-time approaches.

3 What are the main claimed contributions?

Before jumping into the claims this paper makes, it is important to express in a simplified way a few of the technical terms. First, the light-weight dependence chain can be thought of as all the operations that are needed to compute the branch condition. However, if one of the operations takes a long time to finish (such as floating point operations), the sequence of operations is not considered a light-weight dependence chain. This is because the operations are required to complete fast. Second, the difference between guard branches and affector branches is quite important. Although both affect control flow, guard branches are branches that determine whether another branch should be predicted or not, whereas affector branches are branches that have an impact on the condition of another branch [2]. Lastly, the dependence chain engine is a hardware unit that is dedicated to running the dependence chain instructions which has a system in place to prevent it from diverging from the main thread.

Firstly, the paper claims that this is the first time dynamically generated, light-weight dependence chains are scheduled in a way that allows them to run with the same context as the main thread to compute the results of branch instructions. This means that the light-weight dependence chains will follow the control flow that the main thread follows, instead of following its own control flow and getting further and further away from the main thread, which will make the predictions worse. Secondly, this paper marks the importance of correctly classifying guard and affector branches. This paper also introduces a new method of merge point prediction. The main contribution of this paper is the introduction of the Branch Runahead system which dynamically finds dependence chains and pre-computes them on the Dependence Chain Engine (DCE) while making sure these dependence chains do not diverge from the main thread. The last contribution of this paper is the utilisation of three different ways to compute dependence chains.

4 What evidence is offered to support the claims?

First, the paper talks about shortcomings of previous research papers surrounding the topic to show they are the first to look at their specific configuration for computing branch predictions ahead of time.

The next contribution was showing the importance of classifying branches. The paper presents a graph showing that most of the SPEC benchmarks have over 80% of dependence chains with either an affector or a guard branch, justifying their claim.

For the merge point prediction, they claim that the accuracy of their new method is 92%, while previous methods could only get up to 78% accuracy. The main addition is that they added a Wrong Path Buffer (WPB) to retirement, which they use to store the Reorder Buffer (ROB) when there is a flush and later compare the instruction in the WPB to the correct path instructions. If they find instructions that match, then they have identified the merge point. This works because there are over a hundred instructions in the WPB, meaning that when you find matching instructions, it's very likely they have not yet been executed by the main thread.

The main contribution of this paper is discussed thoroughly and provides a deep insight into the results of the simulations. They first present three different ways of executing the dependence chain: non-speculative initiation, independent-early initiation and predictive initiation (this also happens to be the last contribution made). The non-speculative initiation works by simply waiting for each dependence chain to finish computing before starting the next. The independent-early initiation allows chains that do not require the previous chain's results to start before these chains have finished computing the results, allowing for parallelism in the DCE in some cases. The predictive initiation allows all chains to start computing before the previous ones finish by predicting results required for the next chain. This maximises chain-level parallelism in the DCE.

This paper then introduces the microarchitecture of the DCE which includes registers, a cache, reservation stations, ALUs, a TAGE-SC-L and a prediction queue (Figure 7 in the paper). This is the unit responsible for predicting the branches. They then introduce a Hard Branch Table (HBT), to track hard-to-predict branches, which are defined as branches that have a history of being mispredicted. The HBT also tracks affector and guard branches to allow better results. This is how Branch Runahead knows which branches to compute ahead of the main thread.

Finally, they go through the results of their evaluation that uses Scarab to simulate three microarchitectures of different sizes and evaluates them against the SPEC CPU2017 integer speed benchmark, the SPEC CPU2006 integer benchmark and the GAP benchmark. This test shows that the Branch Runahead improves the Misprediction Per Kilo Instruction (MPKI) rate by 37.5%, 43.6% and 47.5%, for the three different sizes and the Instructions Per Cycle (IPC) improves by 8.2%, 13.7% and 16.9%. Regarding the impact on energy used, Branch Runahead decreases it on average, despite the extra hardware, mostly thanks to the decrease in overall run time caused by a higher IPC count. Although they do not provided any accurate numbers of the average energy decrease, they present a graph showing that Branch Runahead can decrease the energy used by around 40% at times, but it can also increase it slightly for certain benchmarks.

5 In your opinion, does the experimental work test the claims adequately?

They present multiple different options and variations of their design to implement their strategy: as discussed previously they have the different ways of executing the dependence chain: non-speculative initiation, independent-early initiation and predictive initiation. Three variants of executing the Dependence Chain Engine are used: unlimited storage Big engine, a 17KB Mini engine, and a 9KB Core-Only engine, which shares reservation stations, physical registers, and functional units with the core.

When presenting results, however, they only give results showing only one combination of the DCE with the ways to execute dependence chain. For example when presenting metrics for the different ways of executing the dependence chain, they do not mention which DCE they are using.

This could mean that a consistent DCE may not have been kept for each test and therefore this could have been used to hide limitations of each variation by pairing it with the best variation of the DCE for that specific benchmark. In my opinion, in order to test the claims adequately the paper should have either showed the results for each combination of DCE and Chain initiation methods, or mention which combination they are presenting and why.

There is no reasoning for the benchmarks picked from the individual benchmarks from the suites chosen - this could mean programs were picked to have the most favourable results for them as they could've chosen ones with the exact characteristics they were after - i.e. picking ones with many hard to predict data dependent branches. It would have been beneficial to see how programs such as gcc or go would preform with this design.

The extra space the design takes up is not too large in theory(2.2% of total space - not the increase in total space, so this could be a slight use of statistics to present a lower % than in reality), but since no layout was proposed it is unclear if this will be true in practice as the configuration of the new connections may not work as well as thought.

The benchmarks used had a wide variety of misprediction rates, and the performance statistics generally increase the higher the original misprediction rate originally was. This indicates their design specifically focuses on the problem that was trying to be addressed in this paper and does in fact have great results. Another thing to note is that one of the DCE variations had unlimited size, which is included to show the potential of the design, and the paper does mention that this is unrealistic and is there to show how close the other variations are to the best case scenario. This shows that (assuming the simulation results are accurate) the 17KB design they propose is very impressive as it approaches the maximum performance without taking too much space.

The paper also does map the limitations of the design, as in some cases with mediocre IPC and MPKI improvements the energy consumption increases. This combination means that this design could have adverse effects when running certain programs. In the benchmarks the paper showed that 8 out of the total 18 benchmarks had higher energy consumption with at least one of the three DCE designs. As discussed earlier - this number may not be completely legitimate and could be higher, as again there was no mention about which chain initiation method was used for each, therefore for every test the most energy efficient result could have been chosen. With the already poor energy performance statistics this could easily mean that in most cases this design has an adverse effect on energy performance, instead of a positive one as claimed in the paper.

One untested aspect was whether there were any security risk increases/decreases. Since this method allows data-dependent branches to be predicted without a history based predictor, it means it is harder to train the predictor to make mispredictions, therefore making the CPU not as vulnerable to spectre attacks.

6 Conclusion

In summary, this paper is well written and researched as it comes up with several convincing variations to their method of decreasing data-dependant branch misses. The weaknesses of their design are somewhat shown - although not enough, however, the weaknesses in their evaluation methods are evident as it appears they have excluded information about what combination of their design is being tested. A justification for this decision is not presented. It is vital for the authors to provide very clear information about the results being published i.e. which combinations of DCE and chain execution methods are being used. The paper is very clear about its aims and - according to the published results - preforms exactly how it was intended to, by heavily reducing MPKI by an average of 47.5% and IPC by an average 16.9%. It is still not clear whether this is a useful implementation, as it does reduce performance in a few of the benchmarks, and increases energy consumption in almost half of presented benchmarks (the amount by which is conveniently left out).

7 Work Distribution

Report sections 1, 3 and 4 written by Luc Jones
Report sections 2, 5 and 6 written by AbdelQader AlKilany
All sections were researched by both team members

References

[1] Eddie Collins *Comparison of Branch History and Branch Correlated Prediction Techniques* https://cs.msutexas.edu/ passos/caesar/branch.pdf
Accessed 15/11/2022

[2] Luc Jones 2022 *Example of guard and affector branches* https://github.com/lucjones02/ACA_CW2/blob/main/guard_vs_affector.c
Accessed 15/11/2022