CPRE 581 Project Proposal

Jake Hafele, Gregory Ling, Thomas Gaul
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50014

I. Abstract

Branch predictor designs have often sought for higher prediction accuracy's to achieve a higher IPC value and better throughput for instruction flows. As new designs and architectures are proposed, the main center of focus in terms of improvement is prediction accuracy, with little room to include variations in terms of area, power, or timing constraints. By comparing previously implemented predictor designs such as global, tournament, 2-level, and GShare predictors using the open-source tool ChipYard, we will be able to analyze the varying constraints to determine an optimum branch predictor based on varying area, power, and timing constraints.

II. Introduction

From as early as 1981, researchers have investigated the topic of branch prediction in the context of high level architecture designs. J. E. Smith [1] proposed multiple strategies including static predict if taken designs and dynamic designs which maintain a table of most recent branches or a 2-bit counter to predict strongly taken or not taken. For each of these compared designs, the main determination of branch predictor performance was in terms of prediction accuracy, where a higher prediction accuracy would lead to less discarded instructions or latency on a branch misprediction. In following works, such as Yeh and Patt's [2] two level predictor analysis, multiple varying prediction schemes are compared between global, per-address, and per-set schemes for each branch prediction level, with again prediction accuracy as the only factor for performance. While Yeh and Patt did only analyze branch predictor tables with less than 512K bits, that was the only consideration to hardware complexity. For many of these papers proposing new architectures, analyzing the impacts of branch predictors in terms of area, power, or logical delay fall to the side in terms of performance comparison.

In terms of complexity, branch predictors have become increasingly advanced to satisfy the high demand of issuing multiple instructions per cycle in modern processors. In newer technology nodes, it can be seen that as technology nodes decrease in size, shortened clock cycles and larger wire delays will become a much more prevalent issue [3]. As the clock period decreases, this can lead to more architectural units impacting the critical path and delay of the design, which can lead to the hardware complexity and delay impact of branch predictors becoming more significant. It can also be seen that increasing the delay of a branch predictor to improve prediction accuracy is never a good trade off [4], leading to the motivation that the comparison of delay between modern branch predictor implementations should be further investigated. While the referenced paper does use Gshare as a baseline prediction model, three new architectures are proposed, instead of following up work on previously implemented prediction designs.

When considering options for improvement for branch prediction or many other areas of improvement throughout a processor, bigger correlates to at least a marginal improvement [5]. Thus, when making the optimal theorized processor, many sections would be of infinite size. However, when it comes to implementing a design in hardware, certain decisions need to be made due to hardware limitations. Branch predictors with a BTB that consists of a size of a small cache can take up to 10% of CPU power [6]. Now, different branch prediction strategies have varying power consumption and size requirements. Each of these prediction strategies has a different size where their benefits per size increase fall off. With the limited power capabilities of a processor, power to improvement must be considered as increasing the power consumption of different segments of the processor yields different levels of improvement. In the work we explored, they investigate the miss rate and performance of many branch prediction algorithms, but rarely do they compare the power cost to performance and not across multiple approaches.

Similar to power, the area required by different strategies differs and increases as the size of each strategy increases [7]. With the decline in Moore's law, relying on being able to pack more into the same area is no longer an option either, so architecture designers need to be wise about how they spend their area to maximize their performance benefits. When looking at other area-intensive performance improvements, it may be found that the area is better spent increasing cache sizes or instruction window size [5]. Like with power, the concern of area required by branch prediction strategies is not investigated by many papers but is a crucial consideration when implementing these in a design.

III. Related Work

To recontextualize branch predictors in terms of area, power, and timing constraints, we must analyze previously published branch predictor designs, as well as analyzing other existing works that consider other factors of performance besides just prediction accuracy. By analyzing previous prediction schemes, we will get a better idea of how to model them in ChipYard, and gain further knowledge in how to present our results. By using more modern works related to other constraints such as area, power, and delay, we will also learn more about what is done and how we can recontextualize prediction accuracy between each of the three additional factors for performance.

A. Branch Predictor History

One of the earliest works in branch prediction was presented by J. E. Smith [1], which presented multiple static and dynamic branch prediction schemes within the context of an improved prediction accuracy. The complexity of these designs varied between static predictions which would always assume branch taken, or more complex schemes where a branch would prediction would be determined on the previous result of the branch, based on the program counter address for the instruction memory. While each of these models were driven based on their prediction accuracy, multiple different FORTRAN benchmarks were used, which shows that it is important to run test programs with varying instruction sets, to test the full capabilities of each branch prediction scheme. Alongside this, multiple table sizes were considered in terms of size for dynamic branch prediction history results, which could benefit out own ChipYard results.

As authors sought out higher prediction accuracy's, more works were published, such as Yeh and Patt's [2] comparison of dynamic predictors which used multiple variations of two-level schemes. In this paper, three types of predictor schemes were applied, including using a history of the last N branches used (Global), the last N branches of the same branch PC address (per-address), or the last N branches in the same set (per-set). Each of these variations were applied for both the first and second level of the dynamic predictors, leading to 9 total variations to analyze for prediction accuracy, for multiple SPEC benchmarks. One useful piece from this paper that could be examined is the comparison of hardware cost for each of the 9 configurations, based on the number of branch history table entries and number of branch sets. This work also modeled many varying branch prediction schemes based on varying branch history lengths, and was cut off after 512K bits, showing that some size considerations were used when comparing prediction accuracy.

Around a similar time that the above work was published, S. McFarling [8] presented a novel idea of a tournament predictor. The core idea of this design is to use both a local and global branch prediction scheme for the second level predictor, and select between which predictor to use with a 2-bit prediction counter. A 2-bit counter would act as a saturated counter to select the best predictor to use, either the global or local predictor, based on the performance history of each predictor, which would drive the choice counter to either increment or decrement. This work presents its results in terms of branch accuracy, and models multiple configurations of tournament predictors in the same plot, including bimodal, bimodal/Gshare, and local/gshare combinations. Overall, a prediction accuracy improvement improved to 98.1%, from the previously known best prediction scheme of 97.1%. This work could be re-contextualized in terms of area, power, and delay constraints to determine if the added 1% in accuracy improvement can be justified in a real-world application.

B. Branch Predictor Motivation

The overall inspiration for our project is similar to that explored by the paper "Branch Prediction, Instruction-Window Size, and Cache Size: Performance Trade-offs and Simulation Techniques" [5]. In the paper, the authors look at the performance benefit of changing the sizes of their branch prediction unit, instruction window size, and cache size due to the die constraints of a design. In it, they specifically explore a hybrid branch predictor with a global sector, a GAg predictor, and a PAg predictor. In the paper, they find that incorrect branch prediction always ends up being a major bottleneck and tends to be an area where resources are required.

One of the major aspects of our project goal is investigating the delays associated with the different predictor designs, and this idea was explored by "The Impact of Delay on the Design of Branch Predictors" [4]. In their paper, they look at the delays caused by increasing branch predictor size because, up to the writing of their paper, the only variable explored was the predictor's accuracy. The paper specifically looks at the gshare predictor and a hybrid predictor with varying sizes. They find that if they just optimize of accuracy of prediction it has a negative effect overall due to the heavy delays incurred by their size. They found their hybrid predictor functioned well until its delay reached a full cycle, and then everything fell apart. Due to their findings, they suggest all papers investigating branch prediction should also report delays due to the implications they found.

The next major aspects we want to explore in our project is power and area. Power is discussed well in the paper "The Impact of Delay on the Design of Branch Predictors" [6]. They explored the trade-off between the power and accuracy of four hybrid predictors and found that the application is an important consideration due to the varying power consumption. Another paper "Low Power/Area Branch Prediction Using Complementary Branch Predictors" explores power in addition to area, another aspect we want to explore [7]. They cover all topics we want to explore and as a result, propose a complementary branch predictor. It is a delay, area, and power-efficient branch prediction algorithm that only completes complicated predictions for branches that are typically challenging for the processor to guess which yields good prediction accuracy without as much of the costs.

Finally in regard to our interest in exploring adding a hint instruction to “prime” the branch predictor we found the paper Branch Penalty Reduction on IBM Cell SPUs via Software Branch Hinting" [9]. This paper describes branch hint instructions attempted in an IBM processor. Some limitations they ran into were a minimum distance between the hint instructions and the real branch instruction which required them to use NOPs to fill the gap, and the need to restructure nested loops to allow for enough space to fit the branch hint instructions before the actual branches. Even with these limitations, they appear to have gotten some improvement from this design.

IV. Specify Ideas

The common trend of modern branch predictors is that to achieve a higher prediction accuracy, designs have become more complex. Due to this, the hardware cost will increase, which could lead to a larger area, power, or delay demand. By analyzing different branch predictors of varying levels of complexity, we will gain further insight in to where certain branch predictors could be used in modern applications, such as desktops, mobile devices, or warehouse scale servers. Based on the constraints for each of these applications, different area, power, or delay demands will need to be met.

We are planning to expand on previous branch prediction schemes and compare prediction accuracy within the context of area, power, and timing constraints. By using a similar set of benchmarks we will be able to better compare prediction schemes such as a global, local, tournament, and Gshare predictors. As referenced above, many previous works only identify branch predictor performance in terms of prediction accuracy, and possibly hardware cost. Our goal is to re-evaluate prediction accuracy with a similar set of benchmarks for multiple prediction schemes, and determine if there are any common trends or benefits for using prediction schemes with low power consumption, size, and delay constraints.

ChipYard allows us to generate a base RISC-V CPU core ready for RTL synthesis. We can then add various branch prediction algorithms to the base core and synthesize them to determine power and area metrics, then run SPEC2006 benchmarks on the different soft cores produced. Potential branch predictors we are considering include:

- Null predictor
- Random predictor
- N-bit local predictor
- Global predictor
- Gshare
- Tournament

V. Intended Results

A. Expected Plots

We intend to evaluate the performance of different branch predictor strategies in light of different benchmarks, and tracking the corresponding area and power usage statistics for each. One plot will show a bar graph of power usage statistics reported in synthesis for either Xilinx Vivado or Altera Quartus Prime for all branch predictor implementations tested, and a second plot will show the same for area usage. A third plot will show performance across various benchmarks from the SPEC2006 suite in either cycles or CPI for each branch predictor cross each benchmark.

B. Producing Plots

We will produce the plots using matplotlib in Python using data generated either via RTL synthesis in Xilinx Vivado or Altera Quartus Prime for area and power metrics, or via RTL implementation on an FPGA for cycle counting. We will use ChipYard to create a base RISC-V CPU implementation, then add different branch predictor implementations designed using the Chisel HDL language. These resulting RTL processors can run on an FPGA, and either run SPEC2006 benchmarks bare-metal (if feasible), and as an alternative backup option, boot a minimal linux distribution on the FPGA, and run the SPEC2006 benchmarks within Linux. Bare-metal will give more specific results in terms of cycle counts, but may not be feasible with the SPEC2006 benchmarks.

C. Designed Infrastructure

Our designed infrastructure required to test our branch predictors consist of the FPGA, potentially a small linux distribution to run on the soft core, and a computer to communicate with and record data from the FPGA.

References

[1] J. E. Smith, "A study of branch prediction strategies," in *Proceedings of the 8th Annual Symposium on Computer Architecture*, ISCA '81, (Washington, DC, USA), p. 135–148, IEEE Computer Society Press, 1981.

[2] T.-Y. Yeh and Y. N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, ISCA '93, (New York, NY, USA), p. 257–266, Association for Computing Machinery, 1993.

[3] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger, "Clock rate versus ipc: the end of the road for conventional microarchitectures," in *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pp. 248–259, 2000.

[4] D. Jimenez, S. Keckler, and C. Lin, "The impact of delay on the design of branch predictors," in *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pp. 67–76, 2000.

[5] K. Skadron, P. Ahuja, M. Martonosi, and D. Clark, "Branch prediction, instruction-window size, and cache size: performance trade-offs and simulation techniques," *IEEE Transactions on Computers*, vol. 48, no. 11, pp. 1260–1281, 1999.

[6] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan, "Power issues related to branch prediction," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pp. 233–244, 2002.

[7] R. Sendag, J. J. Yi, P.-f. Chuang, and D. J. Lilja, "Low power/area branch prediction using complementary branch predictors," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–12, 2008.

[8] S. McFarling, "Combining branch predictors," Tech. Rep. Technical Note 36, Western Research Laboratory, June 1993.

[9] J. Lu, Y. Kim, A. Shrivastava, and C. Huang, "Branch penalty reduction on ibm cell spus via software branch hinting," in *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 355–364, 2011.