# EVALUATION OF L1 RESIDENCE FOR PERCEPTRON FILTER ENHANCED SIGNATURE PATH PREFETCHER

An Undergraduate Research Scholars Thesis

by

ALEXANDER STAGGS

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Paul Gratz

May 2020

Major: Computer Engineering

# TABLE OF CONTENTS

# ABSTRACT

Evaluation of L1 Residence for Perceptron Filter Enhanced Signature Path Prefetcher

Alexander Staggs
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Paul Gratz
Department of Electrical and Computer Engineering
Texas A&M University

Rapid advancement of integrated circuit technology described by Moore's Law [1] has greatly increased computational power. Processors have taken advantage of this by increasing computation rates, while memory has gained increased capacity. As processor operation speeds have greatly exceeded memory access times, computer architects have added multiple levels of caches to avoid penalties for repeat accesses to memory. While this is an improvement, architects have further improved access efficiency by developing methods of prefetching data from memory to hide the latency penalty usually incurred on a cache miss. Previous work at Texas A&M and their submission to the Third Data Prefetching Championship (DPC3) [2] primarily consisted of L2 cache prefetching. L1 prefetching has been less explored than L2 due to hardware limitations on implementation. In this paper, I attempt to evaluate the effect of L1 residence for Texas A&M's Perceptron Filtered Signature Path Prefetcher (PPF) [3]. While an unoptimized movement of the PPF from the L2 to the L1 showed performance degradation, optimizations such as using the L1 data stream to prefetch to all cache levels and updating table sizes and lengths have matched L2 performance.

# NOMENCLATURE

| | |
|---|---|
| SPP | Signature Path Prefetcher |
| PPF | Perceptron Prefetch Filter |
| L1 | Level 1 Cache |
| L2 | Level 2 Cache |
| LLC | Last Level Cache |
| IPC | Instructions Per Cycle |
| DPC3 | Third Data Prefetching Championship |

# CHAPTER I

# INTRODUCTION

As processor execution speeds have outstripped memory access times in computers, architects have continually developed novel ways of increasing memory access efficiency to take full advantage of these processor performance gains and to prevent stalling. To try to hide memory latency of repeat accesses, processors contain data caches smaller and faster than main memory which hold recently used data. By doing so, repeat accesses only pay the latency penalty of the faster cache, rather than the slow main memory latency. To further improve computer performance and attempt to hide the latency penalty of first time accesses to a memory location, methods have been developed to guess these first accesses in advance and pull the respective data into the caches before their first requests in a field called "prefetching." If a prefetch guess is accurate and the data is prefetched before the access from the cache, the latency of going from the cache to DRAM is completely avoided.

Many new innovations in data prefetching are submitted to Data Prefetching Championships (DPCs). In the second of these competitions, a team from Texas A&M University submitted their Signature Path Prefetcher (SPP) [4]. At the most recent, the Third Data Prefetching Championship (DPC3) [2], they enhanced this SPP with a Perceptron Prefetch Filter (PPF) [3] to allow for a separation in mechanisms for gaining coverage and accuracy in prefetching.

The rules of The Second Data Prefetching Championship (DPC2) [6] required submissions to reside only in the L2 cache. The rules of DPC3 opened the possibility of L1, L2, and Last Level Cache (LLC) prefetching. Out of all submissions to DPC3, the only submission to remain primarily

L2 resident was the PPF enhanced SPP. In this thesis, I evaluate the possibility of performance increases from an L1 resident SPP enhanced with PPF.

The submission to DPC3 was optimized for L2 prefetching. A minimally modified version of the DPC3 submission was moved to the L1 cache but showed a performance degradation. This thesis details attempts to optimize PPF to work in the L1 cache and show performance impacts. This is not only an evaluation of how best to build PPF in the L1 cache, but also a test to see if the implementation challenges of designing a hardware constrained L1 resident PPF are worth pursuing. While an initial unoptimized PPF moved to the L1 cache showed a performance degradation, optimizations to learning thresholds and reconfiguring SPP parameters have shown an L1 resident PPF to be at least equal in performance to an L2 resident PPF.

# CHAPTER II

# BACKGROUND

## Modern Computing Problems

Modern computers are designed using the Von Neumann Architecture, where a central processing unit (CPU) reads and execute commands from a memory unit and stores results back in the memory unit. A trend in semiconductor technology advancement known as Moore's Law [1] notes that the number of transistors that can be fit into a given area of an integrated circuit chip double approximately every 2 years.

Processors have taken advantage of this by adding more complex structures that can execute instructions more quickly while memory has gained increased storage space. Increased storage space in memory generally comes at the tradeoff of longer access times. Additionally, in modern computers, processors are executing instructions so quickly that they often must stop execution while waiting for incoming data from DRAM. This problem is exaggerated by growth described by Moore's Law such that the problem is only getting worse and newer methods for improving memory performance are needed. This problem between memory and processing speeds is described as the "Memory Wall" [5].

## Data Prefetching

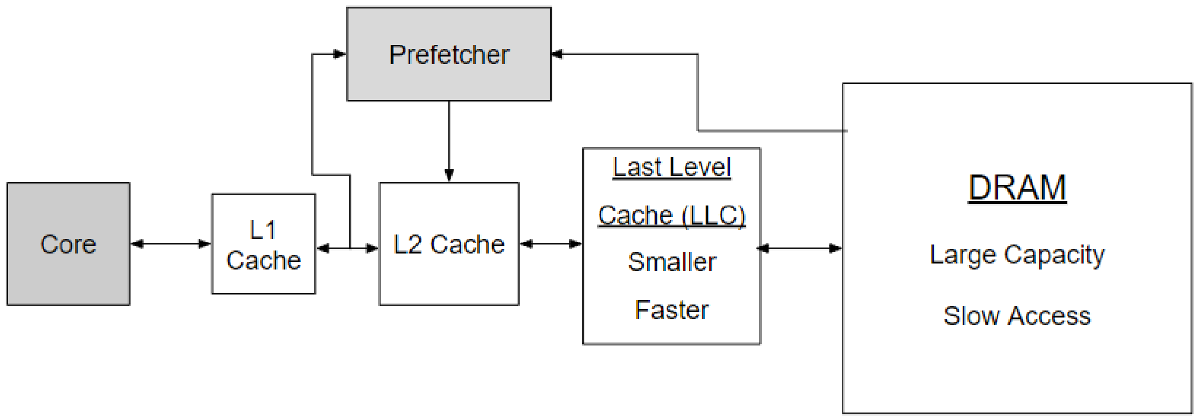

**Figure 1:** Caching and Prefetching Diagram

To help reduce this performance degradation caused by the Memory Wall, computer architects have devised systems of caching, where small and fast blocks of memory are placed near the processor to hold on to recently used pieces of data. As shown in **Figure 1**, modern computers typically implement 3 levels of caches between the Core and the Main Memory or DRAM. These caches take advantage of temporal locality, a trend where recently used pieces of data are likely to be used again soon. In a cached processor, the program can call for a load to a piece of data not already in the cache. This is called a cache miss, and the program must then wait to retrieve data from main memory.

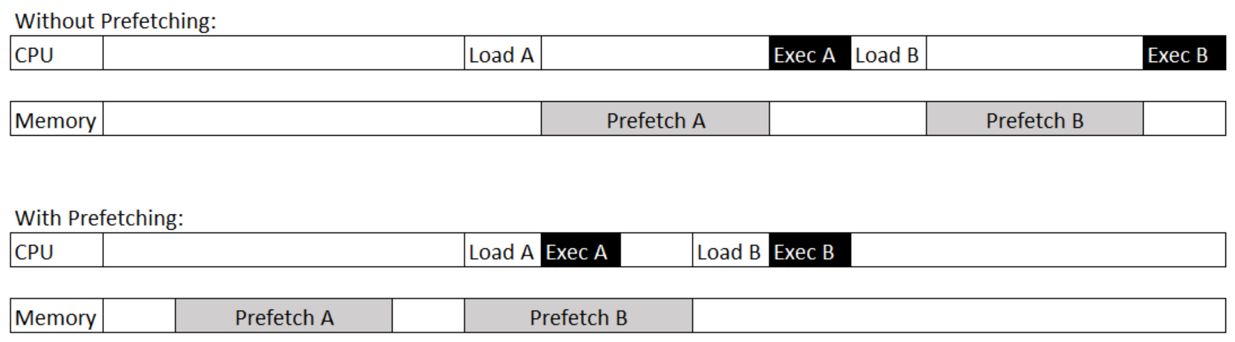

**Figure 2:** Prefetching Timing

These cache misses incur a large latency penalty, as the time to retrieve data from main memory can take dozens or hundreds of process or cycles, which wastes compute time and slows down program execution speed. To hide this latency, modern architects have developed methods for adding hardware to the processor to predict what data will be needed to be prefetched into the cache before the program looks to load it.

**Figure 2** shows how the timing of a prefetch hides this latency. In the top "without prefetching" timelines, the work for prefetching in the memory is performed after the load instruction. The time between "Load A" and "Exec A," the execution instruction, is a stall, where the processor isn't working. The lower example "With Prefetching" shows how a good prediction can load the memory before it is requested, meaning that the time between the load and the execute can be shortened so that there are few or no stall cycles.

**Figure 1** shows a diagram of an L2 resident prefetcher, as it takes information from the access stream to the L2 and uses it to make fetches to the L2 and LLC. A perfect prefetch, where the data is brought into the cache right before it is needed effectively removes the entire latency penalty usually incurred on a cache miss.

**Signature Path Prefetcher (SPP)**

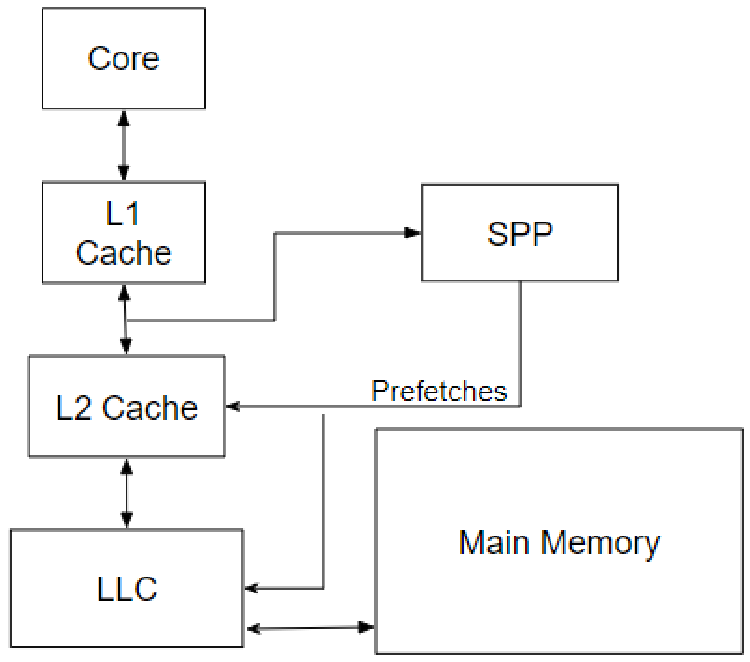

**Figure 3:** SPP Structure

The Signature Path Prefetcher (SPP) [4] was Texas A&M's submission to the Second Data Prefetching Championship (DPC2) [6]. Submissions to DPC2 were only L2 resident prefetchers with no L1 or LLC prefetchers. L2 prefetchers are those trained by L2 cache accesses, or L1 cache misses. As shown in **Figure 3**, SPP receives data from accesses to the L2 cache and uses it to generate prefetches to the L2 cache and LLC. Prefetches pull data from Main Memory to the cache.

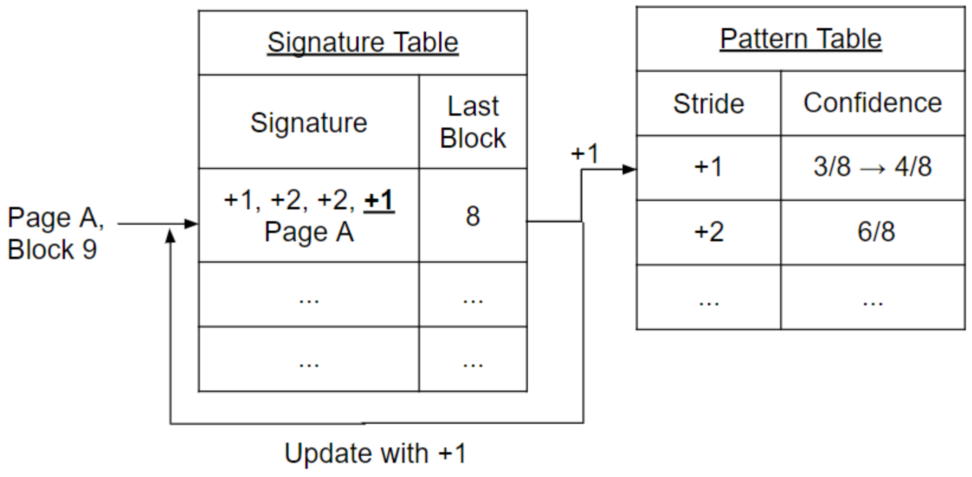

**Figure 4a:** SPP Confidence Update

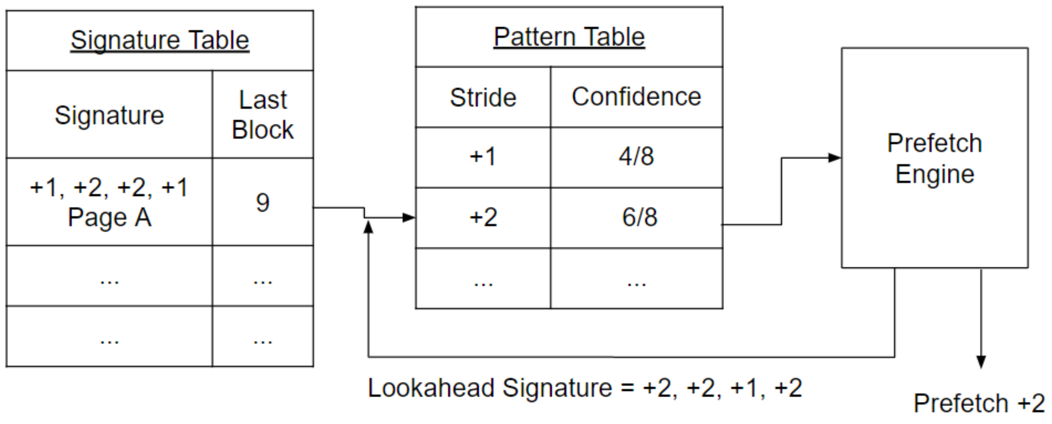

**Figure 4b:** SPP Prefetch

SPP predicts future accesses by constructing a signature table (ST) indexed by the physical page number stores signatures corresponding to previous access patterns. **Figure 4a** shows how the signature is used to index into a pattern table (PT), which stores future access patterns and a confidence for each pattern. Actual accesses to memory and prediction results train SPP by incrementing pattern table values.

**Figure 4b** show how prefetches are generated. If the access to the pattern table shows a confidence higher than a set threshold, then a prefetch is made according to the pattern. Otherwise no prefetch is made. If the prefetch was useful, the confidence is incremented. If not, it is decremented. Using the first generated prefetch, SPP also speculates the next prefetch by indexing the signature of the previous accesses concatenated with the previous prefetch. This occurs recursively if the products of all confidences in the speculated path is above the threshold.

**Perceptron Prefetch Filtering (PPF)**

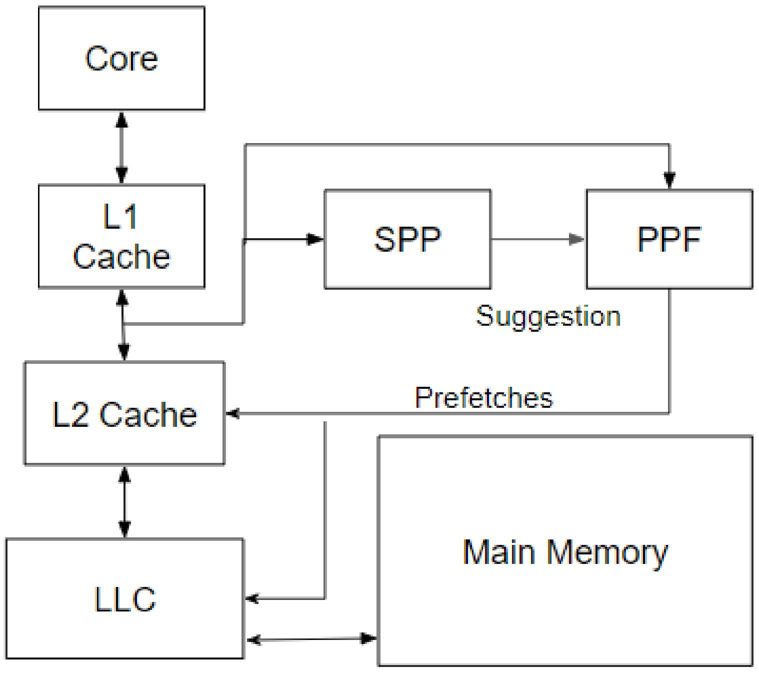

**Figure 5:** PPF Structure

Texas A&M's submission to DPC3 was the Perceptron Prefetch Filter (PPF) Enhanced SPP [3]. This filter is used to separate the accuracy and coverage components of prefetching into separate components. As shown in **Figure 5**, PPF takes information from L2 cache accesses and SPP to generate prefetches. SPP no longer generates prefetches directly but instead suggests prefetches to PPF. SPP is tuned to aggressively suggest prefetches to increase coverage. PPF filters out low confidence suggestions to increase accuracy. The confidence mechanism of SPP is no longer used to determine if a prefetch should be made but is used as a feature for the perceptron.

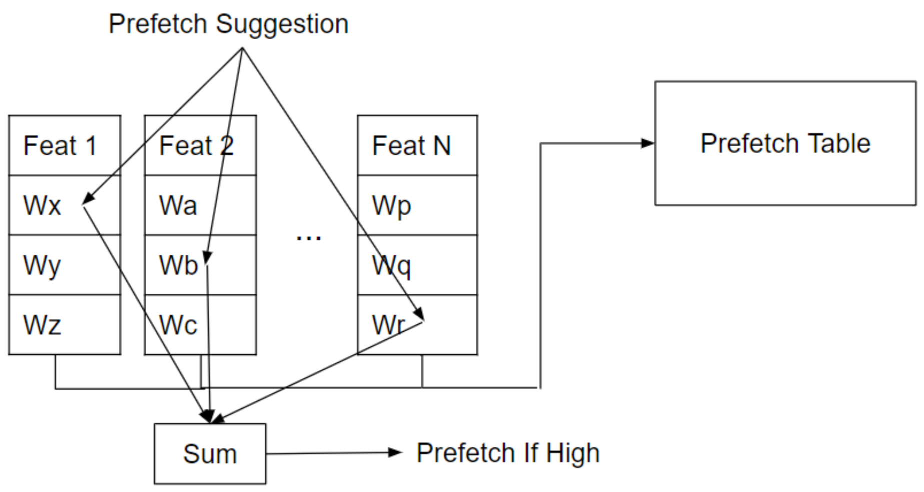

**Figure 6a:** Perceptron Filter Operation

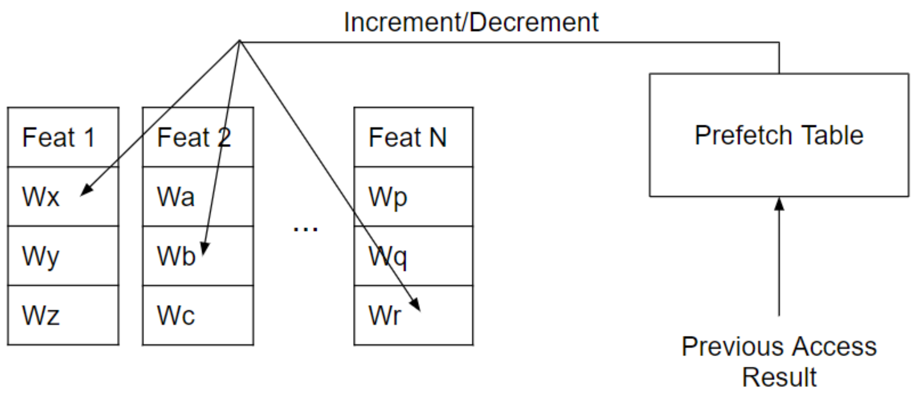

**Figure 6b:** Perceptron Learning

PPF takes data from the prefetch, including the prefetch location, signature, SPP confidence, etc. to build hash tables. These tables are shown labeled "Feat N" in **Figure 6**. The tables are indexed by the feature value and store a confidence value, as shown in **Figure 6a**. For a given prefetch, PPF determines the confidence by summing the output of the hash tables. This perceptron confidence then determines if a prefetch is made using a given threshold. PPF learns by adjusting table weights based on the correctness of previous predictions. This is displayed in **Figure 6b**.

## The Third Data Prefetching Championship (DPC3)

Of the other prefetchers submitted to DPC3, PPF was the only submission that was primarily L2 resident. Other prefetchers submitted to this competition were either primarily L1 resident [8,9] or contained complex prefetching mechanisms at multiple levels [7,10]. By comparison, PPF only included a next line prefetcher in the L1 cache, which is comparatively simpler. The primary avenue at improving PPF that my research tries to test is what performance gains can be brought out of L1 residence.

# CHAPTER III

# METHODS

## ChampSim Simulator

Evaluation of prefetching structures were performed using the ChampSim simulator. It is a trace-based simulator used in DPC3 [2]. Traces are recordings of instructions seen by the processor while a program is running. These instructions have the information about the time and location of memory accesses while the program is running. From this information, the simulator is able to run through the instructions and simulate if execution could be improved if a prefetcher were present.

An L1 prefetcher is much harder than an L2 prefetcher to implement in a real-life processor as the timing constraints are very tight. Memory accesses to the L1 must be fast enough to be accessed in one cycle, so this time dictates the clock rate of the prefetcher. ChampSim is not built to simulate many of these hardware constraints, rather giving an idea of how effective the algorithm for prefetching is. This is beneficial to this study as an L1 resident PPF [3] can be built and evaluated in effectiveness to get an idea of performance gain to help decide if a more low-level hardware implementation of it is worthwhile.

## Prefetcher Implementation

Prefetchers and ChampSim are both written in C++. The prefetchers use functions that are part of ChampSim to perform prefetching operations and to get information about the current state of the machine. The process of using this information to determine a location in memory to prefetch is written as a normal C++ program and then returned to the simulator. My implementations of improvements are modifications to the implementation of PPF submitted to DPC3.

## Evaluation

Prefetcher performance is generated by the simulator as instructions per cycle (IPC). This metric records the number of instructions the processor has executed during the program and divides it by the number of clock cycles needed to execute it. To compare prefetcher implementations to one another, the speedup over a no prefetching baseline is used, shown in **Equation 1**.

$$Speedup = \frac{Prefetcher\ IPC}{No\ Prefetching\ IPC}$$

**Equation 1:** IPC Speedup

Better prefetchers are those with a higher speedup. The IPC speedup of a prefetcher over a set of multiple program traces is used test performance across different workload types. These workloads are the same used in DPC3. The geometric mean of the speedup of a prefetcher over all traces is calculated and used to compare to other prefetchers.

**Residence Changes**

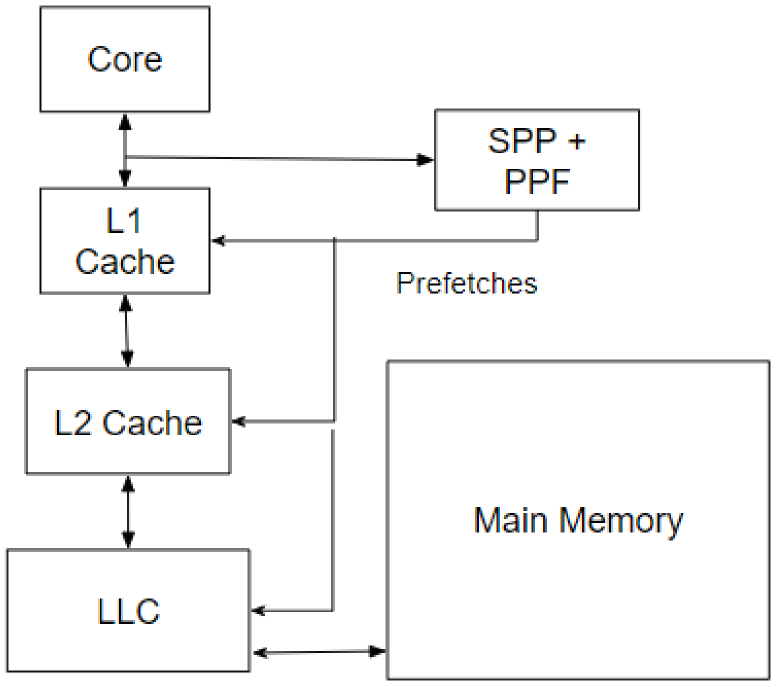

**Figure 7:** L1 Resident PPF Structure

My initial attempts at improving PPF were to move it from the L2 cache to the L1. Compared to the L2 PPF shown in **Figure 5**, the L1 resident PPF in **Figure 7** takes information from the L1 access stream, rather than the L2, and prefetches to all three levels of the cache, rather than just the largest two. Following the trend set by other prefetchers submitted to DPC3, my initial assumption was that there may be some performance benefit to be gained by moving the prefetching engine to the L1. L1 resident prefetchers are able to see all demand accesses generated by the program, while L2 prefetchers only see accesses that miss in the L1. This give a much larger data stream from which PPF can learn from.

The original L2 PPF filled to either the L2 cache or LLC depending on confidence in a prefetch. Two thresholds are set to be compared against the sum of the perceptron feature weights shown on **Figure 6a**. If the sum of all perceptron weights is above the high threshold, a L2 prefetch is generated. If the sum is below the high threshold but above the low threshold, an LLC prefetch is made. If the sum is below the low threshold, no prefetch is made. Moving PPF to the L1 prompted me to add an additional threshold to PPF so that it can fill to all 3 levels of the cache.

## Prefetcher Tuning

The movement PPF to the L1 cache and the addition of another threshold meant that the old perceptron thresholds were likely not suited for the new configuration. To determine appropriate new thresholds, runs of the new PPF configuration were run with varying values for the thresholds. The thresholds were systematically tested from the lowest to the highest perceptron sum value with a difference of 10 per test. This wide step test was used to find a high performing region where a more narrowed test could be performed. Within this region a more fine-tuned test could be performed.

In addition to threshold testing, tests were performed to see if changing some parameters of the PPF tables would produce a positive benefit. Not all tables in the original PPF were of the same size. Initial tests scaled all tables to be larger and smaller by the same scaling factor without changing how large one table is proportionally to a second. This limited how small we could make the largest tables, so all tables were then set to the same, largest size and then tests were performed to see how small this size could be before performance was severely degraded.

Another attempt at tuning the L1 PPF was to modify SPP parameters. As the density of the information stream given to the L1 cache is much greater than that of the L2, the amount of history needed to record all the accesses in a recent time period would be greater. A search for an optimal SPP signature length was run from one entry (3 bits), doubling every test to 96 bits.

# CHAPTER IV

# RESULTS

## Residence Changes

Initial tests to see how an L1 resident prefetcher would perform without great tuning changes consisted of varying the fill levels. While the prefetcher would still receive the L1 access stream, it would prefetch to 2 of the 3 fill levels.

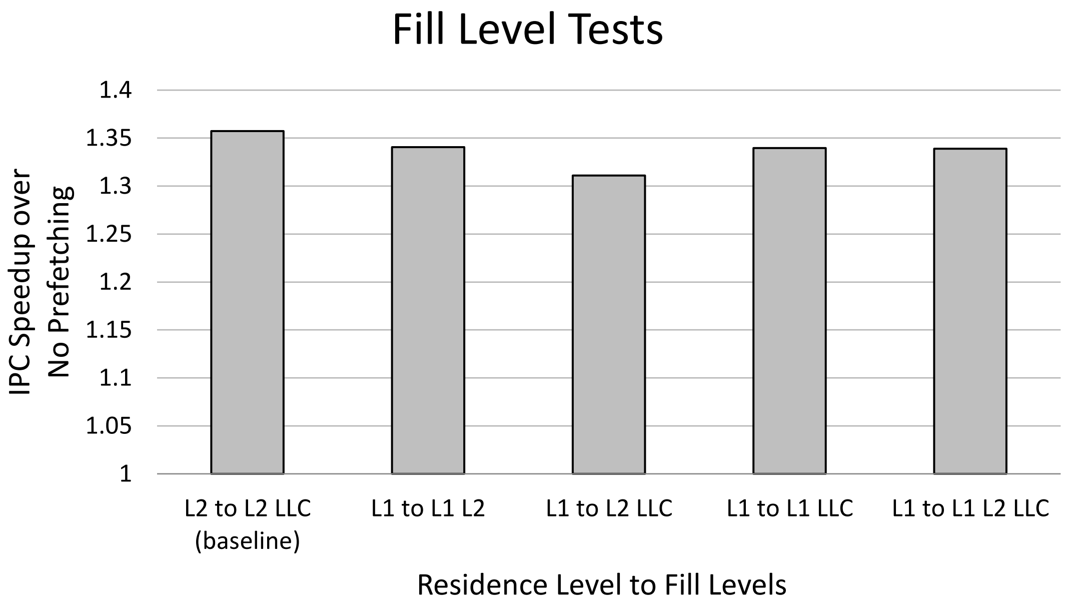

**Figure 8:** Fill Level Test Results

**Figure 8** shows that none of the of the fill level configurations surpassed the baseline L2 PPF, but this is mostly attributable to the parameters of PPF being tuned for an L1 context. L1 to L1 and L2, L1 to L2 and LLC, and L1 to all levels were very close to one another in performance. Because

L1 to all other levels allowed some greater flexibility in tuning thresholds, as there are 3 fill levels rather than 2, it was selected to improve upon. Updates to this prefetcher would likely also cause a performance gain if applied to the other two close performing configurations, but this has not been tested.

**Perceptron Sum Threshold Tests**

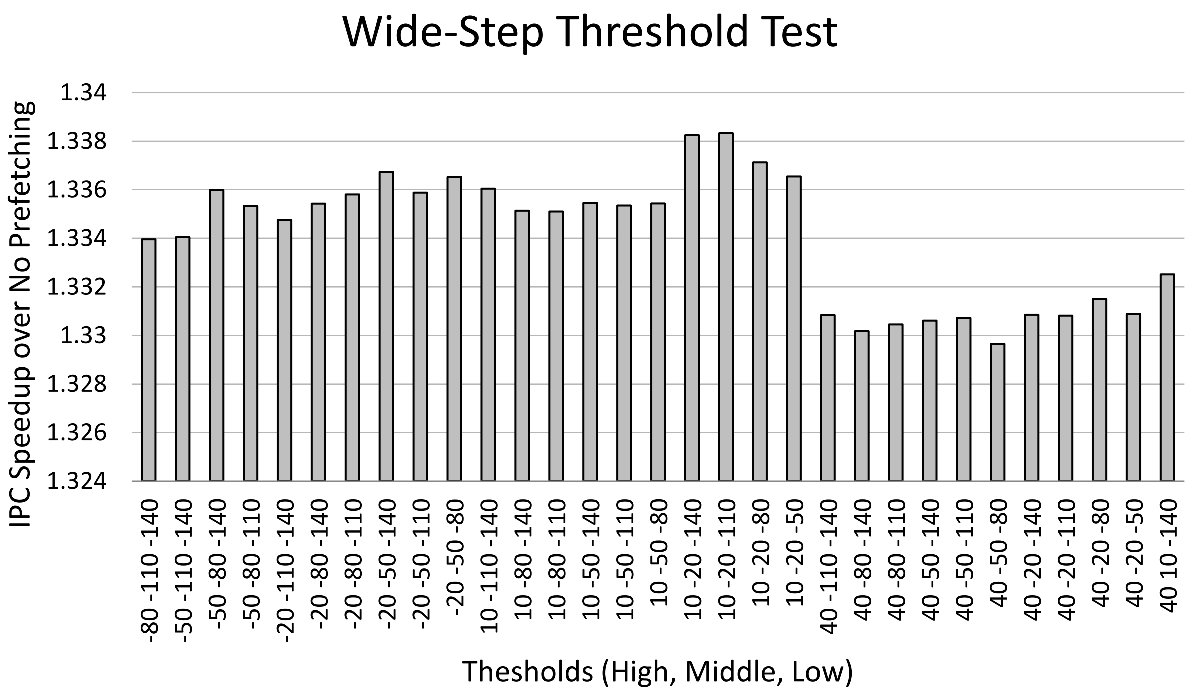

**Figure 9:** Wide-Step Threshold Test

The initial test of PPF Thresholds showed a region of interest where the high threshold is 10 and the middle threshold is -20. High and middle thresholds near these values were then used to make a more specific search with a shorter increment between threshold values. **Figure 9** shows the results of this test.

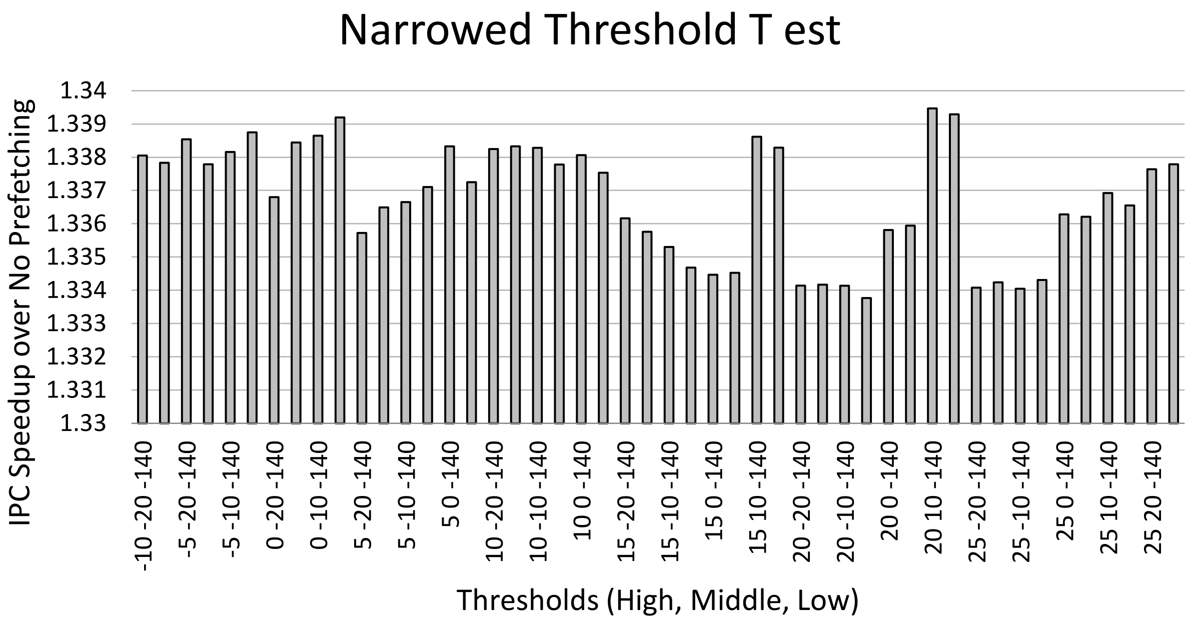

**Figure 10:** Narrowed Threshold Test

From the results in **Figure 10**, the top performing threshold values in this region were near a high of 20 and a middle of 10. There were other areas of the chart with different threshold values that achieved similar results. The changes in performance between these thresholds is very small, meaning that changes to thresholds seem to make little impact on overall performance. PPF seems to be insensitive to fine changes in thresholding, so long as the thresholds are not too far to either the high or low extremes for perceptron sums.

## Perceptron Table Size Tests

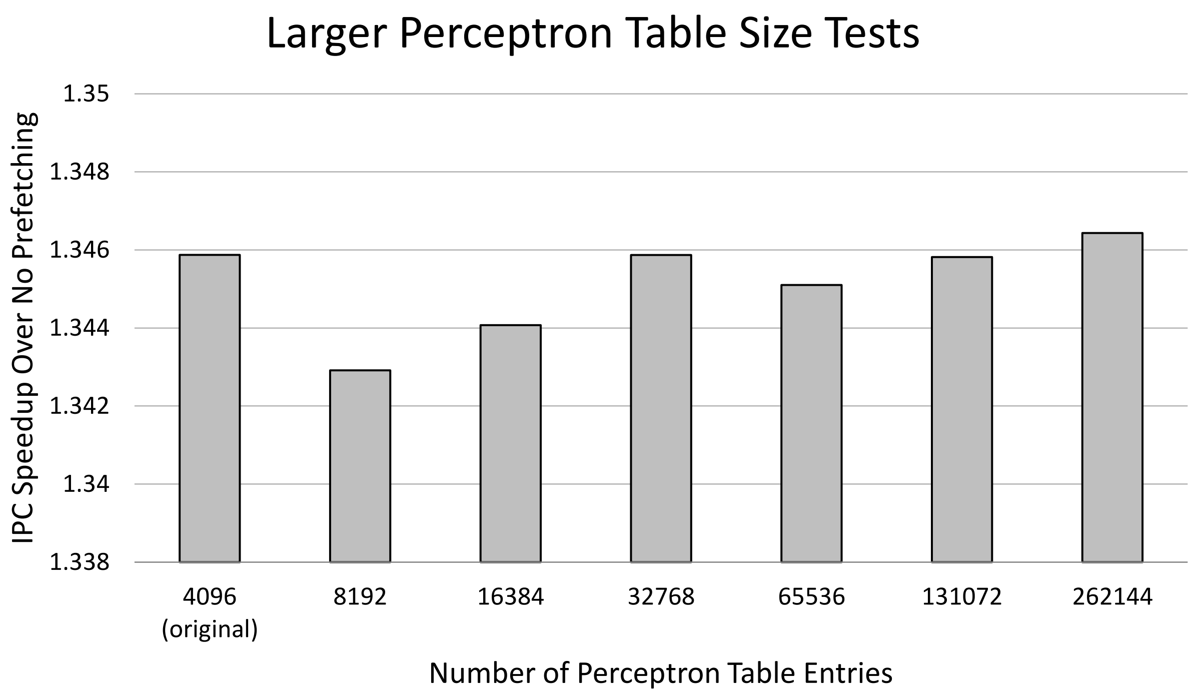

**Figure 11:** Larger Perceptron Table Size Tests

Increasing the size of the perceptron table produced some small variation in performance. There is no clear trend in how performance changes with table size. **Figure 11** shows that the best performing test was with 262,144 entries which is more space to give a table than is practical. These differences were only in the range of fractions of a percent. This led to the idea to test the opposite direction to see if space could be saved in reducing perceptron table sizes. The original table depth of 4096 entries used in the DPC3 submission was chosen as it filled the space budget of the competition rules. Space savings here could be better not only for a practical building of PPF, but also allow for performance benefits in expanding storage of other structures.

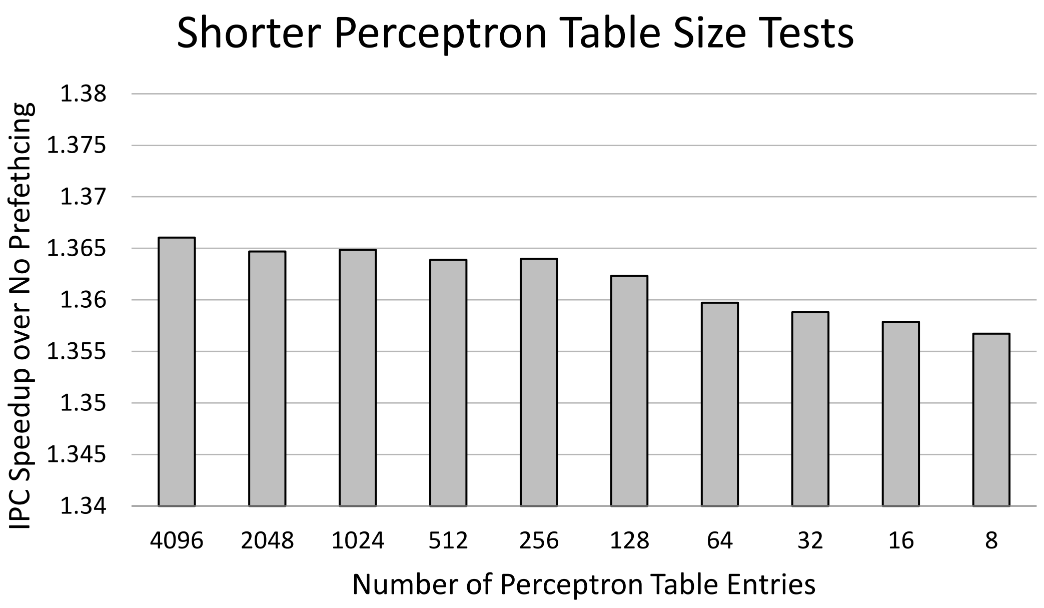

**Figure 12:** Smaller Perceptron Table Size Tests

Remarkably, the performance degradation, shown in **Figure 12**, of significantly reducing the perceptron table sizes is very low for exponentially more space saved. Losing only 1% performance for reducing the tables to only 8 entries deep could prove useful in adapting a prefetcher to use a smaller perceptron table and using this space gain to boost performance in other ways. This test was replicated in original prefetcher submitted to DPC3 and the results closely matched in an L2 context.

**Signature Size Tests**

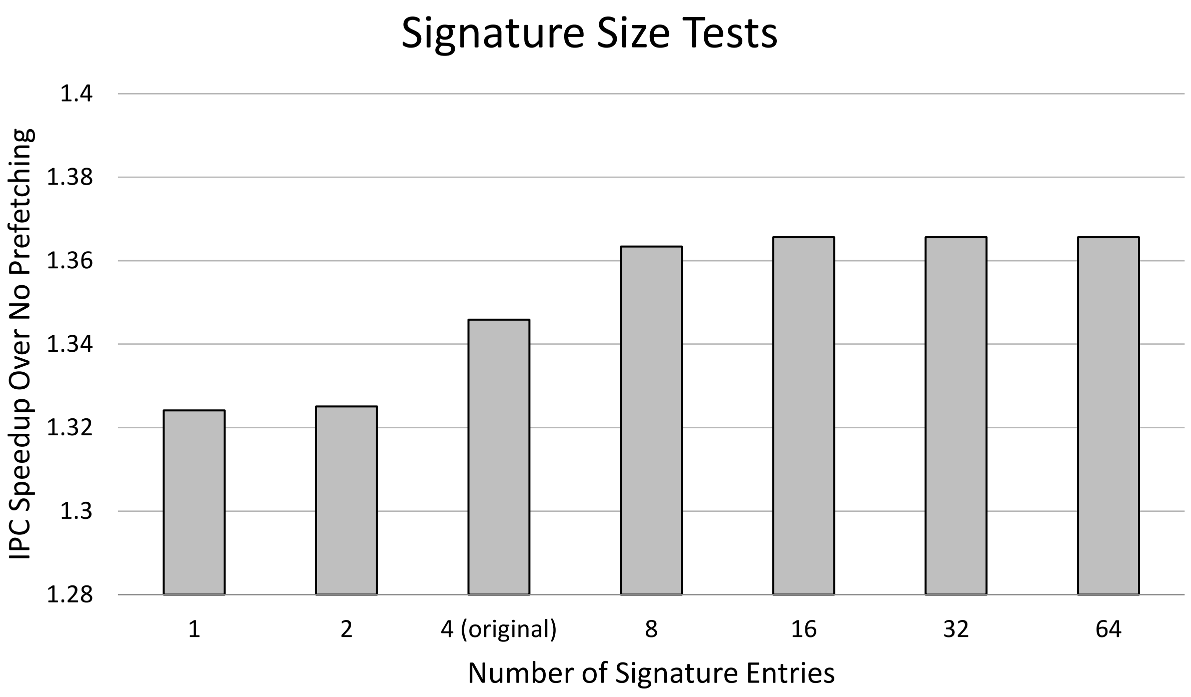

**Figure 13:** Signature Size Tests

Increasing the signature size of the SPP made a significant impact on performance, as shown in **Figure 13**, increasing it by almost 2% over the originally used 4 entry signatures. This increased performance comes at the cost of increased table space but given the results of the perceptron table size tuning, this space can be saved by reducing the length of perceptron tables. This test was duplicated in the L2 and it was found that a signature of 4 entries is optimal for an L2 context. This is likely different because the L1 data stream is much more dense than that which goes to the L2. L2 cache accesses are only L1 cache misses, so a large proportion of requests coming from the processor are filtered out before reaching a L2 PPF.

**Final Comparison**

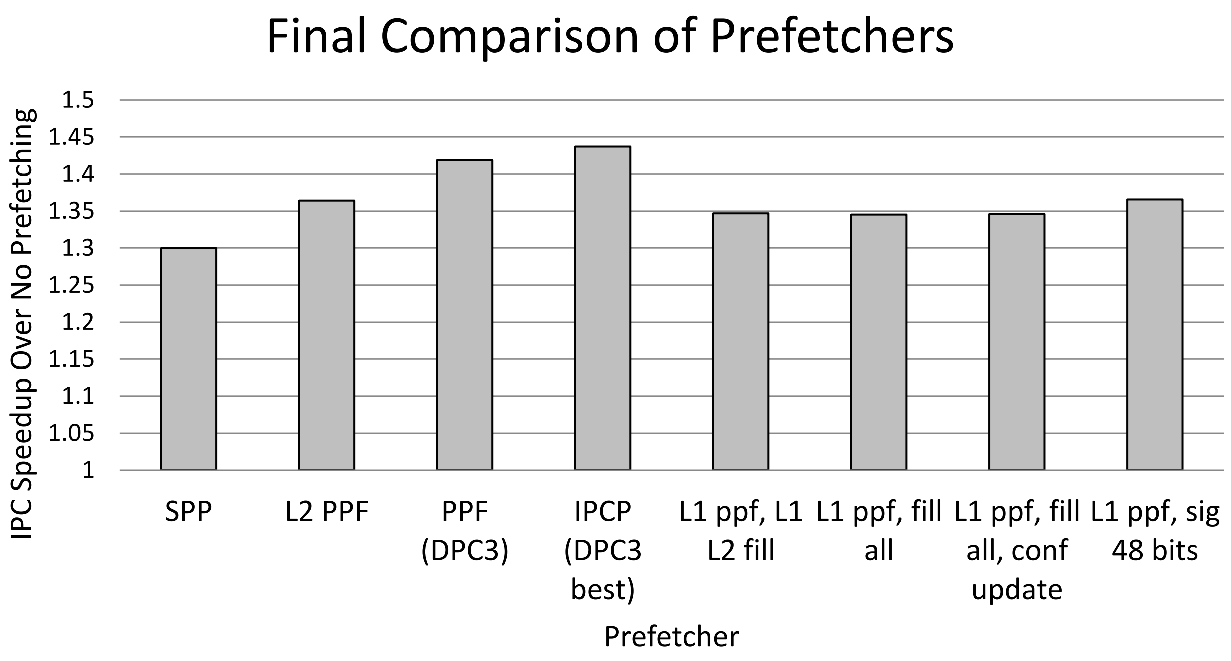

**Figure 14:** Final Comparison of Prefetchers

The results of all tuning runs, displayed in **Figure 14**, show that except for tuning of the signature, all other attempts at modifying a L1 PPF fail to beat the standalone L2 PPF, let alone the version submitted to DPC3. The signature updated PPF outperforms the L2 PPF but does not outperform it. Overall, this shows that it is highly likely that the extra effort needed to implement a L1 resident PPF are not worth undergoing. There is some possibility in gaining increased performance in the L1 through the changing of the features used by the perceptron or perhaps building a PPF at each level, but these have not been tested yet.

The absence of any meaningful performance gain by tuning the PPF perceptron sum thresholds indicates that PPF is fairly insensitive to confidence threshold values so long as the values are not too close to the extremes for perceptron weight sums.

In summary, this testing has indicated that PPF is likely not going to benefit much from being moved from the L2 to the L1. The initial goal of trying to outperform the DPC3 submission with an L1 resident PPF was not met despite tuning revisions. Some lessons learned from this can be applied to the L2 prefetcher though. Shortening the perceptron tables produced little impact on the performance in the L1 and a duplicated test in the L2 showed a similar result. This could possibly allow for some changes to the L2 PPF in which the perceptron tables are reduced to make room for other structures or tables that could have a better performance impact. Additionally, the perceptron sum thresholds produce small performance impacts so long as the values chosen are not toward the extremes. This likely applies to the L2 PPF but has not yet been tested.

## REFERENCES

1. G. E. Moore. 1965. Cramming more components onto integrated circuits. Electronics 38, 8, (April 1965), 114–117

2. The Third Data Prefetching Championship (DPC3). https://dpc3.compas.cs.stonybrook.edu/

3. Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V. Gratz, and Daniel A. Jiménez. 2019. Perceptron-based prefetch filtering. In Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19). Association for Computing Machinery, New York, NY, USA, 1–13.

4. Jinchun Kim, Seth H. Pugsley, Paul V. Gratz, A. L. Narasimha Reddy, Chris Wilkerson, and Zeshan Chishti. 2016. Path confidence based lookahead prefetching. In The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-49). IEEE Press, Article 60, 1–12.

5. Wm. A. Wulf and Sally A. McKee. 1995. Hitting the memory wall: implications of the obvious. SIGARCH Comput. Archit. News 23, 1 (March 1995), 20-24

6. The Second Data Prefetching Championship (DPC2). http://comparch-conf.gatech.edu/dpc2/

7. Mainak Chaudhuri, Nayan Deshmukh. 2019. Sangam: A Multi-component Core Cache Prefetcher. DPC3.

8. Mohammad Bakhshalipour, Mehran Shakerinava, Pejman Lotfi-Kamran, Hamid Sarbazi-Azad. 2019. Accurately and Maximally Prefetching Spatial Data Access Patterns with Bingo. DPC3.

9. Tomoki Nakamura, Toru Koizumi, Yuya Degawa, Hidetsugu Irie, Shuichi Sakai, Ryota Shioya. 2019. T-SKID: Timing Skid Prefetcher. DPC3

10. Samuel Pakalapati, Biswabandan Panda. 2019. Bouquet of Instruction Pointers: Instruction Pointer Classifier based Hardware Prefetching. DPC3.