BRANCH-DIRECTED DATA PREFETCHING

An Undergraduate Research Scholars Thesis

by

SCOTT SHEPHERD

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

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Paul V. Gratz

May 2023

Major: Electrical Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Scott Shepherd, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Faculty Research Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Branch-Directed Data Prefetching

Scott Shepherd
Department of Electrical and Computer Engineering
Texas A&M University

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

Memory prefetching in computer processors is the practice of predicting memory addresses that will need to be accessed and issuing requests to pull data from those addresses ahead of time. These circuits are crucial to combatting the "memory wall", a bottleneck in processor speed caused by the relatively slower progression of memory access speeds compared to progress in instruction execution speed.

This project builds upon the Signature Path Prefetcher (SPP), a prefetcher for the L2C cache developed in Professor Gratz's CAMSIN research group. The SPP decides prefetch addresses based on a delta access history signature. This project explores the possibility of enhancing the SPP by incorporating branch history data (branch decisions & target addresses) into the existing prefetcher structure.

The Branch-Directed SPP aims to improve overall performance as measured by IPC speedup. Results show that the design performs similarly to baseline SPP across these metrics, outperforming slightly on some trace sets and underperforming slightly on others.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. Paul V. Gratz, and my doctoral candidate advisor, Nathan Gober, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

The original SPP code built upon throughout this project was provided by Nathan Gober. The remaining work conducted for the thesis was primarily completed by the student independently, with assistance from Nathan Gober throughout the development.

Funding Sources

No funding was received for this project.

1. INTRODUCTION

1.1 Background

1.1.1 Prefetching Background

Many developments in computer processors focus on increasing instruction execution rate through shrinking transistor gate size, improving pipelining techniques, and other methods. However, developments in memory have historically focused primarily on size rather than speed. These contrasting goals have led to a bottleneck in processor speed known as the memory wall, since accessing memory takes significantly longer than executing instructions [1].

A contemporary computer memory system is organized into tiered levels, with small capacity, low access latency memory (caches) at the high levels and high capacity, high access latency memory at the lowest. When memory instructions are executed, the processor first looks for the specified address in the highest level cache, called the L1 cache, which stores a small amount of information that is very likely to be used. If the address is not found in the L1 cache, the processor then progresses down a level to check the second level, or L2 cache, then the Last Level Cache (LLC), and finally the DRAM or main memory. These levels of the memory hierarchy increase in size and decrease in access speed going down the hierarchy, so it is beneficial for memory access times to store the most proximately used information in the higher levels. To decide which memory is most likely to be accessed, processor designers use the principles of temporal locality and spatial locality. These principles refer to the observed increased likelihood that a future memory access will either be the same address as a recent access (temporal locality) or a nearby address to a recent access (spatial locality) [1].

Data prefetching takes the idea of deciding which information should be stored in caches one step further. Rather than placing recently used information or information from nearby memory addresses in a cache, prefetcher circuits attempt to predict memory addresses that will need to be accessed based on several factors. Then, a prefetcher issues requests to bring that information into caches ahead of time to reduce memory access latency [1].

1.1.2 Prior Work in Prefetching

The simplest and earliest form of prefetcher is the Next Line prefetcher. On each cache access (read or write), the Next Line prefetcher simply pulls in the next line of physical memory into the cache [1]. This approach takes advantage of spatial locality with a fixed behavior of fetching one line ahead of the current memory access.

The Stride prefetcher works similarly to the Next Line, but it uses a "stride" or delta value to fetch N lines ahead of the current cache access rather than just one line [1]. The stride value updates whenever two access deltas in a row are equivalent. For example, if sequential accesses were to address 0x4000 followed by 0x4002 and 0x4004, the difference between memory lines is +2 then +2, so the stride will be set at +2. This approach still takes advantage of spatial locality while accommodating array-type data structures since with certain data types, consecutive elements could be more than one cache line apart. With the Stride prefetcher, there is still a fixed behavior of fetching N lines ahead, but N is variable based on previous strides seen.

The Signature Path Prefetcher (SPP) is a more complex prefetcher that is still based on the line deltas between cache access addresses. SPP generates a signature based on the four most recent line deltas. Then, the most likely next delta is predicted, the resulting address is prefetched, and a speculative signature is developed based on the predicted delta [2]. This

process continues recursively until the confidence of the prefetches reaches a lower bound, at which further prefetches are presumed to pollute the cache with useless data.

SPP operates using a signature table, pattern table, prefetch filter, and global history register. When an L2C access is seen by the prefetcher, the page bits of the address are used to index the signature table, which holds 12-bit signatures made of delta history. The 12-bit signature is then used to index the pattern table, which holds the four most frequently seen page-offset delta values for a given delta history signature. The prefetcher then prepares to issue a request for the memory address found at the current L2C access address plus the delta value. The prefetch filter keeps a record of recent prefetches issued to avoid duplicate prefetches. Lastly, the global history register works to maintain delta history across physical page boundaries when a new delta takes the requested address to a new physical page. SPP adds an extra source of variability when compared to the Stride prefetcher since it can predict variable access patterns rather than a simple string of N-line deltas.

1.2 Motivation

It is reasonable to expect that different parts of a program have different memory access patterns. Depending on the location within a program (in instruction address space) and the operations and progression of the program (looping, function calls, recursion, etc.), memory could be accessed in many ways. In other words, program control flow should affect memory access patterns, meaning that a prefetcher could benefit from utilizing program control flow data in its memory prefetching decisions.

SPP takes a step in this direction by looking at varying access patterns throughout a program and correlating them with future access patterns. However, SPP does not take into account a global view of how the program is behaving. SPP does not consider location within

instruction address space, branch decision patterns, or anything of the sort. We hypothesize that incorporating branch data could improve the prefetching capabilities of SPP.

This paper will explore the possibility of improving the performance of the Signature Path Prefetcher by basing prefetching decisions on branch history data, including branch decisions and instruction targets, in addition to the line delta history built into the original SPP design.

2. DESIGN

2.1 Design Overview

The Branch History Based Data Prefetcher (BHBDP) aims to improve upon the Signature Path Prefetcher (SPP) by incorporating branch history as a factor in its prefetching predictions. At the system level, the BHBDP aims to improve the instruction execution speed of a processor through a reduction in memory access latency by requesting data from memory before it is needed. There are two main subsystems within the overall BHBDP system: the Branch Spy L1I/D prefetcher and the Branch-Directed SPP L2C prefetcher. Figure 1 illustrates the subcomponents of the Branch-Directed SPP and the connections between the two subsystems.

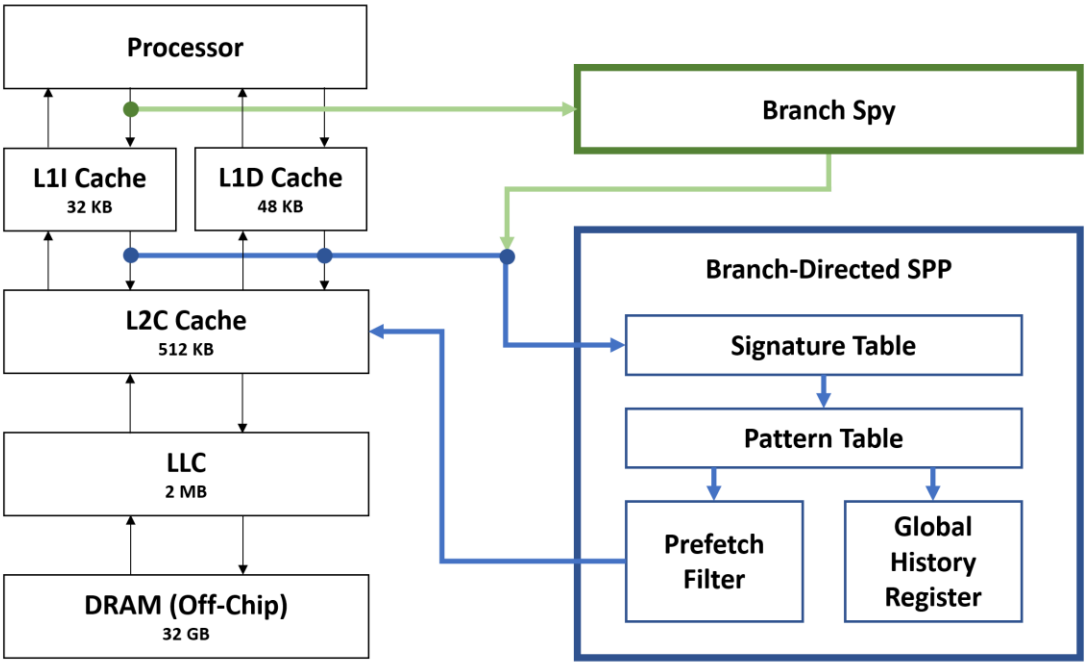

Figure 1: Block Diagram of Branch History Based Data Prefetcher System

The BHBDP uses L2 accesses (checking L2 cache after an L1 cache miss) and metadata outputs from the L1 cache level (branch history data passed through along with L2 access). When L1 cache accesses are made, the Branch Spy subsystem records branch decisions (branch taken or not taken) and sends that data through to the L2 prefetcher (Branch-Directed SPP subsystem) upon an L1 cache miss.

Within the Branch-Directed SPP subsystem, the signature table uses delta patterns from recent L2 accesses and branch history information to calculate a 12-bit signature for each physical memory page. The PT subsystem stores predicted delta patterns along with a confidence estimation (based on prefetch success) evaluating whether the resulting prefetched address will be useful to the processor.

If the predicted address (current demand address + predicted delta) remains in the current physical page, the prediction moves on to the filter subsystem. The filter subsystem runs a final check to ensure that the predicted address is not a redundant prefetch before sending it off to the cache hierarchy. This step reduces unnecessary memory bus traffic. The filtered prefetch addresses are sent to the L2 cache so that it will be placed in a more readily accessible location.

If the predicted address from the PT subsystem (current offset + predicted delta) leaves the current physical page, the prediction moves on to the GHR subsystem. The GHR stores prefetch requests that cross a page boundary so that the new page can retain the learning stored in the 12-bit signature from accesses in the previous page.

2.2 Branch Spy Design

The Branch Spy subsystem is module that records branch history data based on instruction demand fetch to the L1I cache. When a branch target is fetched, the branch is known to be taken. If a branch decision is made, but no target is fetched, the branch is known to be not

taken. The Branch Spy maintains a history of recent branch decisions. Each time a new branch decision is made, the branch history value is shifted left by one bit, then the new decision is placed in the least significant bit as a zero or one. In this manner, the most recent 63 branch decisions can be held in the integer (leaving the most significant bit at zero to avoid issues with signage and integer overflow). The Branch Spy is placed between the L1 and L2 cache levels to observe branch history data from L1I cache accesses and pass that data along to the L2C cache (where SPP is located) on L1I cache misses.

Branch history data is sent from the Branch Spy subsystem to the Branch-Directed SPP subsystem via the metadata feature of the ChampSim processor simulator [3]. When a cache operation occurs (memory read, for example), a 32-bit integer is passed in as a metadata input, and a 32-bit metadata output can be passed on to the next cache level. Since the Branch Spy subsystem operates as an L1 prefetcher and the SPP subsystem operates as an L2C prefetcher, the branch history data stored in the Branch Spy can be sent through its metadata output into the metadata input of the SPP subsystem.

Once each branch decision is known, and the target of each decision is known, there are many ways to record branch history data in a way that differentiates the value by branch decision patterns. Five categories of recording branch history data were tested: bimodal, bins, majority, target, and overlay. These categories are listed in Table 1 with a brief description for reference. The remainder of Section 2.2 will describe them in more detail.

Table 1: Branch History Data Indicator Categories

Name	**Bits**	**Description**
Bimodal	2-3	3-, 4-, or 5-bit counter. +1 for branch taken, -1 for branch not taken. Differentiates between long stretches of a particular decision (ex. looping) and transitions between the two. The 2 to 3 most significant bits of the counter are used for the branch history indicator.
Bins	3	3-bit representation of N branch decisions. For a 3-bit bin indicator using 23 branch decisions, Bin 0 means 0-2 of the decisions were taken. Bin 1 means 3-5, …, and Bin 7 means 21-23 decisions were taken. Tested with N = 7, 23, 63.
Majority	1	Majority decision of last N branch decisions (1 for majority taken, 0 for majority not taken). Tested with N = 7, 15, 31, 63.
Target	3-6	3 or 6 bits of the most recent branch target address (64-bit address) that was taken. Tested with target address bits [23:21], [23:18], [17:15], and [17:12].
Overlay	12	12-bit value of 12 most recent branch decisions XOR with 12-bit signature (for indexing pattern table) or 12-bit L2C access page (for indexing signature table)

Many of the branch data indicators used are based on those used in branch predictors since they have a similar goal of using past branch data for future predictions. The bimodal indicator is based on the bimodal counter developed for an early branch predictor [4]. This indicator is a saturating counter that increments by one whenever a branch is taken, and decrements by one whenever a branch is not taken. When the maximum value is reached ($2^N - 1$, where N is the number of bits in the counter), a subsequent taken branch does not increment the

counter, and similarly when the minimum value of zero is reached, a subsequent “not taken” branch does not decrement the counter. The bimodal indicator was tested for 3-bit or 4-bit counter in which the two most significant bits are used as the indicator and for a 4-bit or 5-bit counter in which the three most significant bits are used as the indicator. To refer to these indicators, they are named as “bimodal_XofN” in which X is the number of bits in the indicator (2 or 3), and N is the number of bits in the counter (3, 4, or 5).

The overlay indicator is based on the XOR computation used in the gshare branch predictor, which showed a positive effect when combining global branch decision history and branch PC page with an XOR operation [5]. Its XOR operation is also similar to that of a branch table buffer (BTB). This indicator is calculated as the XOR of a 12-bit value holding the 12 most recent branch decisions (least significant bit is most recent decision, 0 for not taken, 1 for taken) with the 12-bit index that the original SPP design uses, which varies depending on which table (signature table or pattern table) the overlay indicator is indexing. There are no additional variations of this indicator, so it is simply named as “overlay” for future reference.

The bins and majority indicators were developed for this paper as methods of summarizing recent branch decisions to understand the type of code that the program might be in. The bins indicator takes the count of N branch decisions modulo the number of values in each bin. For example, the indicator bins_63to3 in Table 2 takes the count of the past 63 branch decisions that were “taken” divided by 8: 64 count options (0-63 branches taken) divided by 8 value options (0-7 since it is a 3-bit indicator). Three bins indicators were tested, in which 7, 23, or 63 branch decisions are translated to 1 of 8 bins in the 3-bit indicator. The bins indicators are named as “bins_Nto3” in which 3 is the number of bits in the indicator and N is the number of decisions compressed into the bin indicator (7, 23, or 63). The majority indicator is similar, but

only sets one bit based on whether the majority of N decisions were taken (1) or not taken (0). The majority indicator was tested for N = 7, 15, 31, and 63, and the indicators are named as "majority_N" for future reference. The bins and majority indicators aim to show approximate trends in program behavior since a large amount of "taken" branches could indicate loop-like behavior, which would be caught by the majority flag or by a high bin value.

Finally, the target indicators record a portion of the instruction address (bits [X:X-2] for 3 bits of target or [X:X-5] for 6 bits of target) for the most recent branch target address taken. The goal of the target indicator is to separate signature development and/or delta prefetches based on the approximate location of the code that the program is in. The address segments tested were [17:12], [17:15], [23:18], and [23:21] to test variation in the number of bits and to test the higher and lower bits of the address. The lowest 12 bits, which comprise the physical page offset, were avoided to avoid too much noise and variation. The goal is to distinguish the general area of the program, not the specific part of the code. The indicators are named as "target_X_X-2" for the 3-bit target indicators and "target_X_X-5" for the 6-bit indicators, in which X is the most significant bit grabbed from the recent branch target address.

With all the variations to the branch data indicators discussed in this section, there are 16 total indicators tested in this paper. The indicators are listed in Table 2 with their category, name, number of bits, and a brief description.

Next, the indicator value must be sent through to the SPP subsystem. The bimodal and target indicators are updated on each branch decision and stored until the next L1 miss to be sent through to SPP. The bins and majority indicators are calculated from the 64-bit branch decision history integer immediately before sending metadata through to SPP. The overlay indicator is decided within the Branch-Directed SPP subsystem since other information is needed for the

Table 2: Branch Data Indicators Tested

Category	Name	Bits	Description
Bimodal	bimodal_2ofN	2	Bits [N-1:N-2] of N-bit bimodal counter (two most significant bits)
	bimodal_3ofN	3	Bits [N-1:N-3] of N-bit bimodal counter (three most significant bits)
Bins	bins_Nto3	3	Count of previous N branch decisions that were "taken" reduced to 8 bins (number of distinct 3-bit values)
Majority	majority_N	1	Majority decision of last N branch decisions
Target	target_X_X-2	3	Bits [X:X-2] of the most recent branch target address that was taken
	target_X_X-5	6	Bits [X:X-5] of the most recent branch target address that was taken
Overlay	overlay	12	12-bit value of 12 most recent branch decisions XOR with existing 12-bit index

XOR operation to compute the final indicator, so the raw branch decision history is sent through to SPP rather than the final overlay indicator.

Once branch data is sent through to SPP, it can be incorporated in two different places: the signature table, and the pattern table. Different branch data indicators can be tested for indexing each of these tables, so the metadata output value must be structured to fit any two of the branch data indicators shown above. Therefore, there must be 6 bits of space for each. Bits [11:6] will store the indicator used to index the signature table of the SPP, and bits [5:0] will

store the indicator used to index the pattern table of the SPP. The upper 4 bits of the metadata output store a 4-bit value, 4'b0101, which is checked to confirm that any metadata entering SPP is only coming from the Branch Spy, as expected. This leaves bits [27:12] unused for the current design.

2.3 Branch-Directed Signature Path Prefetcher Design

The original Signature Path Prefetcher, which this research expands upon to include branch history data, operates using a signature table, pattern table, prefetch filter, and global history register. When an L2C access is seen by the prefetcher, the page bits of the address are used to index the signature table, which holds 12-bit signatures made of page-offset delta history. The 12-bit signature is then used to index the pattern table, which holds the four most frequently seen page-offset delta values for a given 12-bit delta history signature.

The operation of the new Branch-Directed SPP design maintains a similar approach of using page-offset delta values to inform prefetch requests, but the delta values are accessed using a combination of both delta history and branch history data. Instead of indexing the signature table by solely the L2C access page, branch history data is included in the index. Similarly, the 12-bit signature used to index the pattern table is no longer comprised solely of delta history. The top bits of the signature are replaced with branch history data to allow for delta access patterns to be based on branch decisions and/or targets as well.

As seen in Section 2.2, six methods of recording branch history data were researched this semester. Each of these branch history indicator values were incorporated into the signature table index and pattern table index separately to research their impact on memory latency and therefore overall processor performance. The signature table indices tested are shown in Table 3. Additionally, the pattern table indices tested are shown in Table 4.

Table 3: Signature Table Indexing Methods Using Branch History Data

	Bit											
Method	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
majority_N	1-bit BHI	Lowest 11 bits of L2C access page										
bimodal_2ofN	2-bit branch hist indicator		Lowest 10 bits of L2C access page									
bimodal_3ofN, bins_Nto3, target_X_X-2	3-bit branch history indicator			Lowest 9 bits of L2C access page								
target_X_X-5	6-bit branch history indicator						Lowest 6 bits of L2C access page					
overlay	12-bit decision history XOR 12 bits of L2C access page											

Table 4: Pattern Table Indexing Methods Using Branch History Data

	Bit											
Method	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
majority_N	1-bit BHI	11-bit delta history (4 deltas, left-shifted by 3 bits per lookahead)										
bimodal_2ofN	2-bit branch hist indicator		10-bit delta history (4 deltas, left-shifted by 3 bits per lookahead)									
bimodal_3ofN, bins_Nto3, target_X_X-2	3-bit branch history indicator			9-bit delta history (3 deltas, << left-shifted by 3 bits per lookahead)								
target_X_X-5	6-bit branch history indicator						6-bit delta history (2 deltas, left-shifted by 3 bits per lookahead)					
overlay	12-bit decision history XOR 4 deltas, left-shifted by 3 bits per lookahead											

Another part of the design that must be discussed is the table size changes. The original SPP design used 5.31 KB of storage space [2]. In hardware design, greater storage space is equivalent to greater physical space in the processor, and therefore a higher cost of manufacturing. Therefore, it is important to maintain a small size in the prefetcher. The current design for the branch directed SPP uses 45.53 KB of storage. This design choice was made to

allow for the maximum possible improvements in processor performance by creating enough entries in the signature table and pattern table that each possible index would have its own entry. That way, no delta history data is overwritten throughout the progress of the simulation. Future work can be done to reduce the storage space of the prefetcher, but the analysis of this paper is limited to the larger signature and pattern tables to better understand the upper bound of performance for the tested prefetcher designs. Additionally, to better compare the design of this paper to the original SPP design, the baseline SPP used for comparison later in the Results section uses the larger table size. By doing so, the performance of the designs in this paper can be fairly compared to the original SPP design. Table 5 compares the storage space of the original SPP design to the storage space of the current branch directed SPP design.

Table 5: Storage Space of Original SPP Design vs. Branch Directed SPP Design

		Original SPP Design		**Branch Directed SPP Design**	
Structure	**Bits per Entry**	**Entries**	**Bits**	**Entries**	**Bits**
Signature Table	41	256	10496	4096	167936
Pattern Table	48	512	24576	4096	196608
Prefetch Filter	8	1024	8192	1024	8192
Global History Register	33	8	264	8	264
		Total Bits:	43528	**Total Bits:**	373000
		Total KB:	5.31	**Total KB:**	45.53

3. RESULTS

3.1 Performance Metrics

The primary performance indicator of a processor core is instructions per cycle, or IPC. IPC is proportional to performance since the goal of a processor is to execute the maximum number of instructions in the minimum number of clock cycles. It is a useful measure of relative performance between prefetchers because it is not influenced by the clock frequency of the processor or any other physical characteristics. Instead, as long as all other variables are kept consistent across runs (branch predictors, cache sizes, core characteristics, etc.), the only variation in IPC is a result of more or fewer addresses being prefetched into the cache successfully. Cache misses slow the processor down (lower IPC) since the next level down in the cache hierarchy will have a longer access time. If a prefetcher does its job properly and pulls relevant data into the cache ahead of its usage, the cache miss ratio will be reduced, leading to a lower clock cycle count for the program. Therefore, the denominator of the IPC will decrease, increasing the value. The analysis of these prefetcher designs will be limited to performance as a measure of IPC only.

3.2 Simulation Methodology

The design was evaluated using the trace-based ChampSim processor simulator. The simulations were set up to model a single out-of-order core, with its parameters and the parameters of the memory hierarchy shown in Table 6.

Table 6: Processor and Memory Hierarchy Parameters for ChampSim Simulation

Processor Element	**Parameters**
Core	1 Core, 3.2 GHz, 256 entry ROB, 4-wide, 64 entry scheduler, 64 entry load buffer
Branch Predictor	16K entry gshare, 20 cycle mispredict penalty
L1D Cache	32KB, 8-way, 4 cycle delay, 8 MSHRs, LRU replacement policy
L1I Cache	32KB, 8-way, 4 cycle delay, 8 MSHRs, LRU replacement policy
L2 Cache	256KB, 8-way, 8 cycle delay, 16 MSHRs, LRU replacement policy, non-inclusive policy
L3 / LLC	2 MB, 16-way, 12 cycle delay, LRU replacement policy, non-inclusive policy

The full set of traces using during the Third Data Prefetching Championship (DPC-3) was used to evaluate the design. In total, there are 48 traces used. Multiple weighted SimPoints were used for each trace. SimPoints [6] are representative samples of a benchmark program used to better understand the performance of a long program without needing to simulate too many instructions, keeping the simulation resource usage down. Each SimPoint ran in ChampSim for 200 million warmup instructions plus 1 billion simulation instructions. ChampSim records key statistics throughout the run, such as instructions per cycle (IPC), cache hits and misses, and prefetcher accuracy and coverage for each cache level.

Since only single core simulations were performed, a single DRAM channel was used. ChampSim uses separate virtual and physical address spaces, with arbitrarily randomized mappings between the two. For these simulations, a 4KB page size is used when mapping virtual to physical addresses, consistent with the original SPP paper [2].

3.3 Control & Baseline Results

Before examining the results for the index variations tested, we must discuss the control and baseline results that will be used for comparison. The control test for this paper uses the same processor and memory hierarchy parameters as defined in Table 6, but no prefetchers are attached to any cache level. By testing the no-prefetcher control across the same trace set, we can ensure that the tested SPP variations are improving performance, and we have a good standard by which to compare the SPP versions. When the IPC speedup of each SPP version is tested with respect to the control, we can see the performance improvement by implementing a given L2C prefetcher. Additionally, the original SPP design is tested across all traces as a baseline for comparison of the SPP variations designed in this paper. When the IPC speedup of each SPP variation is tested with respect to baseline SPP, we can see the performance change when swapping out the original SPP design for the new design. Table 7 shows the geometric mean IPC of the no-prefetcher control design and baseline SPP design, along with their geometric mean speedup when compared to the control design. The geometric mean speedup of baseline SPP compared to control, 1.1360, will then be used in Sections 3.4 and 3.5 when comparing speedup of each index variation design with the baseline SPP design.

Table 7: Geometric Mean IPC for No-Prefetcher Control and Baseline SPP

Name	**Geometric Mean IPC**	**Speedup vs. No-PF Control**
No-Prefetcher Control	0.9601	1.0000
Baseline SPP	1.0907	1.1360

3.4 Signature Table Index Variation Results

The first step in testing the design of the branch-directed SPP was to examine the effects of basing the index to the signature table on branch history data. In the baseline SPP design, the

index to the signature table is simply the lowest 12 bits of the current L2 cache access page (bits [23:12] of the address being accessed). Therefore, the 12-bit signatures held by the signature table are distinguished by the current physical page of memory being accessed. By replacing part of the signature table index with each of the 16 branch history data indicators discussed in Table 2, this paper examines the possibility of better separating the use and development of delta history signatures based on information from recent branch decisions or targets. Table 7 shows the geometric mean IPC speedup of each signature table index variation compared to both a control processor (no prefetcher implemented) and a baseline SPP implementation. IPC speedup is simply a ratio of the IPC for a given design to the IPC for a design used for comparison. For example, if a design has an IPC speedup of 2 vs. the control, the design runs twice as fast as the control.

As seen in Table 8, none of the signature table indexing variations tested outperform the baseline SPP design. Baseline SPP has a geometric mean IPC speedup over the no-prefetcher control of 1.1360. The highest geometric mean IPC speedup of the signature indexing variation methods tested was the target_23_21 design with a speedup over control of 1.1360, and speedup over baseline of 1.0000. Therefore, the best design exactly matches the baseline SPP speedup, but none of the designs improve performance.

There are still some trends we can see from the signature table indexing data, even if there were no performance improvements. The target indicators performed the best overall for indexing the signature table, with the more significant target bits performing better than the less significant bits. The remaining indicators, in descending order of performance, were majority, bimodal, bins, and finally overlay, which performed the worst by far.

Table 8: IPC Speedup vs. No-Prefetcher Control & Baseline SPP for Signature Table Indexing Variations

Category	**Name**	**Speedup vs. Control**	**Speedup vs. Baseline SPP**
Bimodal	bimodal_2of3	1.1244	0.9897
	bimodal_2of4	1.1286	0.9934
	bimodal_3of4	1.1230	0.9886
	bimodal_3of5	1.1271	0.9921
Bins	bins_7to3	1.1166	0.9849
	bins_23to3	1.1249	0.9902
	bins_63to3	1.1291	0.9939
Majority	majority_7	1.1280	0.9929
	majority_15	1.1296	0.9944
	majority_31	1.1338	0.9980
	majority_63	1.1342	0.9984
Target	target_17_12	1.1321	0.9965
	target_17_15	1.1333	0.9976
	target_23_18	1.1340	0.9982
	target_23_21	1.1360	1.0000
Overlay	overlay	1.0669	0.9391

3.5 Pattern Table Index Variation Results

The next step in testing the branch-directed SPP design was to examine the effects of basing the index to the pattern table on branch history data. In the baseline SPP design, the index to the pattern table is simply the 12-bit signature found in the signature table based on a history of page offset deltas between subsequent L2 cache accesses. Therefore, the most likely next page offset delta (the output of the pattern table when accessed) is based solely on previous delta history. By replacing part of the pattern table index with each of the 16 branch history data indicators discussed in Table 2, this paper examines the possibility of more accurately predicting

future delta accesses using information from recent branch decisions or targets. Table 9 shows the IPC speedup of each pattern table index variation compared to both a control processor (no prefetcher implemented) and a baseline SPP implementation, in the same way as Table 8.

Table 9: IPC Speedup vs. No-Prefetcher Control & Baseline SPP for Pattern Table Indexing Variations

Category	**Name**	**Speedup vs. Control**	**Speedup vs. Baseline SPP**
Bimodal	bimodal_2of3	1.1345	0.9984
	bimodal_2of4	1.1343	0.9982
	bimodal_3of4	1.1338	0.9979
	bimodal_3of5	1.1336	0.9976
Bins	bins_7to3	1.1346	0.9987
	bins_23to3	1.1341	0.9983
	bins_63to3	1.1341	0.9981
Majority	majority_7	1.1353	0.9992
	majority_15	1.1351	0.9990
	majority_31	1.1354	0.9993
	majority_63	1.1354	0.9993
Target	target_17_12	1.1308	0.9950
	target_17_15	1.1345	0.9986
	target_23_18	1.1284	0.9933
	target_23_21	1.1324	0.9965
Overlay	overlay	1.1258	0.9910

As seen in Table 9, none of the pattern table indexing variations tested outperform the baseline SPP design. The highest geometric mean IPC speedup of the pattern indexing variation methods tested was a tie between the majority_63 and majority_31 designs with a speedup over

control of 1.1354, and speedup over baseline of 0.9993. Therefore, the best design has approximately a 0.07% reduction in speed, virtually the same as the baseline SPP design.

There are still some trends we can see from the pattern table indexing data, even if there were no performance improvements. The majority indicators performed the best overall for indexing the signature table, with the higher decision count indicators (majority of 31, 63) performing better than the lower options (majority of 7, 15). The remaining indicators, in descending order of performance, were bins, bimodal, target, and finally overlay, which performed the worst by a small margin.

3.6 Analysis

Across both the signature table and pattern table, none of the 16 indexing variations tested improved the performance of SPP. All variations performed better than the no-prefetcher control, but none outperformed the baseline SPP design.

Based on the speedup results from testing each of the indexing variation options in the signature table and pattern table, branch history data is not beneficial to making prefetching decisions in the SPP framework. The modifications tested in this paper, though detrimental to performance, help to clarify why the baseline SPP design works well.

It is clear from testing the overlay indicator as a pattern table index that it is valuable to maintain the shifted delta structure for the 12-bit signature, even though the 12-bit signature is compressed. Conventional wisdom would suggest that the XOR of distinct table indices can produce a composite index with the information of both, as shown in the gshare branch predictor [5]. However, this appears not to be generalizable to all tables. The structure of the SPP signature is valuable for use as an index into the pattern table, and the XOR of the branch target obscures

the signature data. The overlay indicator, which imitates the gshare XOR operation, also does not perform well as a signature table index.

Based on the branch target indicators tested, it is better to use target data to separate signature developments rather than to alter signatures themselves. At a high level, the signature table correlates memory access patterns distinguished with the memory region the program is using. The pattern table, on the other hand, attempts to discern what the next cache line delta will be based on a string of past deltas. It follows that the instruction target information would be more in line with the purpose of the signature table, since its incorporation into the signature table index would begin to distinguish memory access pattern histories by program region in addition to memory region. The more positive results for incorporating instruction target data in the signature table index as opposed to the pattern table index confirms that the signature table functions according to this intuition, separating out memory access history based on the general state of the program.

4. FUTURE WORK

While this project may not have shown positive performance results in terms of processor IPC, there are plenty of valuable learnings. The negative results show which directions not to pursue moving forward, which in turn narrows down the recommended paths forward. The following subjects are areas which this project has not considered, so there is potential for SPP performance improvements down these paths.

4.1 Loop Handling

The main idea that can be pursued in future work in this research topic of delta- and branch history-based data prefetching is to better handle looping. Looping structures will have access patterns distinct from non-looping patterns, so it may be beneficial to separate learning based on whether the program is in a loop, as well as the expected length of the current loop.

One possible solution is to partition the signature table into two signatures for each entry, accessed by L2 cache address page (same as original SPP design): a loop mode signature and a linear mode signature. Access patterns can differ greatly when in a loop or not in a loop. For example, when looping through an array of items, it is likely to see constant delta accesses when moving from one element of the array to the next. On the other hand, when not in a loop, it may be more common to access several pieces of data from a single instantiation of a class or struct, in which the access pattern could differ based on the size of each piece of data.

Another consideration for handling loops is to predict when loops will terminate and use the remaining length of the loop to influence the depth of future prefetches. For example, if a loop is entered with an expected 100 iterations remaining, it would be beneficial to prefetch

deeper than the average depth of 2-3 seen throughout the simulations examined in this paper. Similarly, if a loop is likely to end within a few iterations, prefetching too deep could fill the cache with unnecessary information, reducing its effectiveness. Prior research has been done in the area of loop termination prediction for the application of branch predictors [7], but a similar approach could be used to predict the remaining length of loops and override the prefetch depth of SPP to better accommodate the loop.

4.2 Signature Size

In the original SPP design, the signature used to index the pattern table performed best with a size of 12 bits. To maintain a level playing field to compare the indexing variations tested in this paper, the signature size was fixed at 12 bits. However, now that program control data has been added into the SPP structure, the 12-bit signature is not necessarily the best option to handle the data. Future work could look into expanding the signature size to accommodate for the new information being included in the signature. For example, rather than replacing 3 bits of the delta-only signature with a bimodal indicator, the bimodal indicator could be appended to the front of the signature, resulting in a 15-bit signature that includes the full 12-bit delta history and a 3-bit bimodal indicator. An increased signature length may help provide better delta predictions since the full 12-bit delta history could still be included, and the branch history data would simply be used to differentiate between the same delta pattern occurring in different areas of the program.

4.3 Storage Considerations

For the simulations conducted in this paper, the storage capacity of the design was not considered as a limitation. We assumed perfect preservation of pattern table and signature table contents, increasing the size to accommodate all data with no aliasing. Once the loop handling

approaches and signature length variations are tested with the large signature and pattern tables (to eliminate overwrites of signatures and predicted deltas), the next step will be to reduce the table sizes to a physically implementable size. The goal will be to reduce the storage space to a similar level to the original SPP design, maintaining its classification as a lightweight prefetcher.

5. CONCLUSION

Through this project, we sought to design a new version of the Signature Path Prefetcher that handles program control flow data, such as branch decision history and branch target addresses. After incorporating control flow data into SPP prefetching decisions through varied implementations of the signature table and pattern table indices, the original SPP design remained the top performer in terms of IPC.

The results discussed in this paper provide a better understanding of which elements in SPP are most essential. It is clear from the underperformance of the overlay indicator in the pattern table index that the structure of shifted cache line deltas is essential to SPP's performance. From a broader perspective, we can also see that the XOR approach adapted from the gshare branch predictor and BTB structure does not apply as generally as we had hoped, considering the decrease in performance when using the overlay indicator in the signature table index. Finally, branch target addresses prove more useful in separating signature development than in predicting next deltas, so future developments in SPP should utilize target address data in the signature table rather than the pattern table.

Moving forward, the results of this paper demonstrate that indexing changes alone are unlikely to improve SPP performance. Some potential directions for future work include loop handling, experimenting with signature length, and working toward a physically implementable storage size.

REFERENCES

[1] W. A. Wulf and S. A. McKee, “Hitting the memory wall,” ACM SIGARCH Computer Architecture News, vol. 23, no. 1, pp. 20–24, 1995.

[2] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, “Path confidence based lookahead prefetching,” 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016.

[3] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The Championship Simulator: Architectural Simulation for Education and Competition," 2022.

[4] J. E Smith. A study of branch prediction strategies. In Proc. 8th Int. Sym. on Computer Architecture, pages 135–148, May 1981.

[5] S. McFarling. Combining Branch Predictors. WRL Technical Note TN-36, June 1993. 3.2, 3.4, 5.2.2, 6

[6] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and More Flexible Program Analysis," 2005.

[7] Sherwood, T., & Calder, B. (2000). Loop termination prediction. *Lecture Notes in Computer Science*, 73–87. https://doi.org/10.1007/3-540-39999-2_8