Exploiting Historical Context in Irregular Data Prefetching

July 12, 2016

Houston P. Putman IV
University of Texas at Austin

Abstract

In this paper we propose a new irregular data prefetcher, Temporal Sparse Distributed Memory, which builds on the address correlation and PC localization. Address correlation, learning correlation between pairs of addresses, is the inspiration behind the current leader in irregular data prefetching. TSDM distinguishes itself from current address correlation techniques by including the correlation information from a stream of addresses in the prediction step. This combination of correlation information replicates the historical context of the stream.

We show that the use of historical context gives TSDM a significant advantage over conventional address correlation on micro-benchmarks that exhibit high levels of aliasing. TSDM fares slightly better on irregular benchmarks than the current irregular prefetching leader, and it performs similarly when run on a restricted hardware budget.

1 Introduction

Many programs spend considerable time waiting for data to arrive from DRAM. Therefore a way to make programs faster would be to predict the addresses in the memory that will be needed by the processor and to fetch the data ahead of time, storing it in the processor’s cache. We call this process data prefetching.

Consider a program that uses an array for its data structure, iterating through it repeatedly. Since arrays store data linearly in the memory, the sequence of memory locations that will be accessed will be regular. Regular data streams contain addresses that increase or decrease at regular intervals. In Figure 1 the red arrows represent a regular data stream in which each address increases by 1 from the previous address. Many programs use linear data structures; however many others rely on pointer-based data structures, such as linked lists, which do not store data linearly. Iterating through a linked list repeatedly will give an irregular data stream, where there seems to be no pattern between the accessed memory addresses. Programs using arrays can also have irregular memory patterns. For example using binary search on an array would create an irregular memory stream. The green arrows in Figure 1 represent an irregular data stream, in which no regular pattern can be found.

Many irregular prefetchers have chosen to rely on address correlation (AC) to detect patterns in irregular streams [1-4]. Address correlation allows irregular prefetchers to learn correlations between pairs
of addresses. For example, in Figure 1 a prefetcher that uses AC would correlate A with G, G with J, J with E and so on. Correlated addresses streams may become interleaved, so addresses often don’t correlate with their successors in the overall data stream, which makes it harder to learn correlations. PC localization was introduced to solve this issue by separating the data stream by the PC load instruction of each data access, thereby allowing prefetchers to correlate addresses accessed in similar situations [5–8]. Prefetchers with PC localization train on each stream individually, since each localized stream is treated as separate.

Markov Prefetching [?], the Global History Buffer [?] and the Irregular Stream Buffer [1] are previous techniques designed to use address correlation to solve irregular prefetching. One major issue with address correlation is the amount of data needed to make the predictions [4]. Since some programs can use millions of unique addresses, storing millions of correlations can become intractable. With ISB, Jain and Lin have found a way to implement address correlation and PC localization in a space-efficient way that is feasible to use in hardware. Our goal is to maintain the space efficiency, address correlation and PC localization of ISB while increasing performance.

De-aliasing, which no current irregular prefetcher implementing address correlation with a feasible hardware budget supports, is the area we have chosen to solve to increase prefetching performance. Aliasing occurs when one address has multiple successors that occur in different situations. The example in Figure 3 shows that address Y can be followed by C–K, D–J or D–I. When preceded by X, Y is always followed by C–K. Current irregular prefetchers that use AC wouldn’t be able to make the correct prefetch for X–Y and W–Y, because the prediction would only be based off of Y and not the address that precedes it. Therefore a way to retain stream context is necessary for de-aliasing. If TSDM were to store the context of each stream, the amount of space required would grow exponentially. To solve de-aliasing while only using a linear amount of space, we will need to remove historical context while training and reconstruct it during the prediction step.

We have developed Historical Context (HC) to act as this context replacement during the prediction step. Instead of using one address to determine a successor address, historical context uses a group of addresses to vote on a successor address. In the example above, Y is followed by both C and D, while W is only followed after 2 accesses by D and E. Historical context combines the correlation information of W and Y, so that D has a context of 2, while C and E each have a context of 1; therefore D would be prefetched after W–Y. By having multiple addresses contribute in the prediction step, we feel that historical context successfully recreates the context lost through training.

In this thesis we present Temporal Sparse
Distributed Memory (TSDM), an irregular data prefetcher that implements AC/PC/HC in a space efficient way. TSDM is built on the idea of Sparse Distributed Memory (SDM) [9], a model of the human memory. SDM is designed to employ redundancy and fuzzy input matching in an environment with limited space. We will show how TSDM has followed the design principles integral to SDM and how they are the building blocks our implementation of historical context.

2 Related Work

Irregular prefetching has been studied for a long time and therefore many ideas have been implemented to solve it. We have included the subset of prefetchers we feel are relevant to our solution. These prefetchers are all based on address correlation and the GHB and ISB can be used with PC localization.

2.1 Markov Prefetching

The Markov Prefetcher [?] was an early data prefetcher that implemented address correlation. The prefetcher stored multiple possible successors for each address, and each successor has a probability of success associated with it. When an address is accessed the Markov predictor prefetches the possible successor with the highest probability of success. Since the Markov Predictor follows the Markov process, each prediction can only be made with information from the current address. Therefore the Markov Prefetcher cannot implement the historical context necessary to solve aliasing, since the successor with the highest probability will always be chosen no matter what context the address is used in.

2.2 Global History Buffer

The Global History Buffer [?] stores a FIFO history of memory accesses. A hash table contains pointers from keys, be it PCs or addresses, to the most recent use of that key in the GHB, designed to make finding the indexes of keys fast. Since many addresses can map to the same key, each element in the GHB contains a pointer to the next element in the GHB mapped to by that key. Because of this, finding prediction information can be time consuming since a collection of off-chip pointers must be fetched one at a time. Since the GHB can store multiple uses of the same address, it can provide context. Due to the space and time constraints this is infeasible in a real situation [10].

2.3 Irregular Stream Buffer

Regular prefetching is a simpler problem than irregular prefetching, and ISB [1] exploits this by mapping physical addresses to structural addresses. Structural addresses represent temporal locality instead of physical locality. Therefore they can be prefetched regularly, one after another. This technique lends itself to suitably sense the structural address space can be stored off chip and recently used segments can be cached on chip.

3 Sparse Distributed Memory

Kanerva’s Sparse Distributed Memory [9] is a model of human memory. SDM generalizes concepts as points in a high-dimensional space, and the distance between two concepts can be represented as distances between these points.

SDM solves a problem very different than irregular data prefetching, but the problem shares a few traits with de-aliasing. Primarily de-aliasing requires that a prefetcher be able to accept fuzzy input, since we can expect noise in the data stream. Secondarily because we can expect noise, data must be stored redundantly so that an address being absent from the stream doesn’t adversely effect prefetching performance.

4 Aliasing

Aliasing occurs when an address is followed by different addresses in different situations. An example irregular data stream is shown in Figure 3, where the same addresses are used in different contexts. Although Y is followed by C and D and K, I and
J afterwards, it is clear that it is followed by C, K when preceded by X. We propose that adding de-aliasing would increase the performance of irregular data prefetchers utilizing address correlation.

Current prefetching techniques based on address correlation have been unable to address aliasing in a space efficient manner. ISB is unable to accomplish this because every address can have one successor in its structural address space. Markov Prefetching cannot distinguish between different situations and therefore will always prefetch the address with the highest confidence. GHBs store context information, since it has a history of the stream; however the storage requirements are too high to be viable in a limited hardware environment.

5 Solution

Using the ideas behind Kanerva’s SDM, specifically fuzzy input matching and data redundancy, we have created TSDM to address aliasing in a space efficient manner while using address correlation and PC localization. This is accomplished by storing historical stream information for a selection of addresses seen in the stream. We introduce historical context by combining the historical stream information from multiple addresses. From this we can distinguish our predictions from situations where these multiple addresses are accesses individually instead of together. Since we believe that an address can be accessed in multiple situations, the ability for an address to store information for multiple situations is very important. By adding historical context, or historical stream information from other addresses in the stream, we hope to be able to choose which situation is correct for that place in the data stream.

5.1 Storage

TSDM uses a mapping of addresses to Fuzzy Historical Streams (FHSs) for storing temporal correlation information. For a given base address, a FHS stores the information for address streams that follow the base address every time it occurs in the data stream. There are possibly exponentially many address streams that follow the base address. Each historical stream is stripped of context before being added to a FHS, allowing each FHS to store its correlation information in a linear amount of space. For example when the stream X, Y, Z is added to a FHS we no longer know that X is followed by Y, we only know that X possibly occurs at 1 access after the base address and Y possibly occurs 2 accesses after the base address. This context will be added back later in the prediction process, shown in the next section.

An example of the FHS mapping is shown in Figure 4. Each address key maps to a FHS that stores the possible addresses that can follow the key in the stream. Each element in the FHS contains an address, the relative position seen after the key in the stream and a confidence of the element, which in its most basic form is a counter that increments every time the address is seen with the same relative position after the key.

There are many ways to choose the address keys that map to the FHSs in TSDM, from having every address be key to only having a small sample.

5.2 Historical Context

We have designed a new type of context, historical context (HC), to combat aliasing. Historical context is the idea of letting a stream of addresses have input in the decision of what address to prefetch. HC is used to replace the context lost when writing the streams into FHSs. This is different from common AC techniques, which decide what to prefetch based
Figure 5: Combining FHSs

<table>
<thead>
<tr>
<th>Key</th>
<th>Relative Current Stream Position</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>X</td>
<td>C:4</td>
</tr>
<tr>
<td>Y</td>
<td>D:2</td>
</tr>
<tr>
<td></td>
<td>K:2</td>
</tr>
<tr>
<td></td>
<td>L:2</td>
</tr>
</tbody>
</table>

Combined: C:6-2 D:2-1 K:4-2 L:2-1 J:1-1

Confidence – Context

Assuming an address stream of X,Y

on the current address only and do not try to recreate stream context.

HC is accomplished in TSDM by combining the FHSs of multiple addresses, as show in Figure 5. This example uses the TSDM in Figure 4 and an address stream that ends in X,Y. We know that Y is followed by C,D and K,I,J afterwards. We also know that C occurs 2 accesses after X and I,J come 3 accesses after. This information comes straight from the FHSs. Y has 2 options to prefetch for the next address and 3 for the address after the first prefetch. When the FHS of X, offset by one position, is added to the FHS of Y things become more clear. The elements of X are offset by 1 to find the current relative stream position. This offset is determined how far back in the history the key address is. The current relative stream position of an address represents how far in the future we expect to find that address in the stream. Both Y and X can agree on prefetching C and then K, since they are the addresses with highest context and confidence for the relative current stream positions of 1 and 2.

5.2.1 Fuzzy Matching

Fuzzy matching is integral to historical context, since the context would be unnecessary if every address was always followed the same address. Fuzzy matching allows the same address to be prefetched for multiple address streams with slight differences. Consider the example TSDM in Figure 4 and the address streams X,Y,C and X,Y,N. Even though N has never been seen before in this context, X and Y give enough context for the TSDM to know to prefetch K. We know that there can be slight differences when we see a familiar address sub-stream, so fuzzy matching is important in allowing us to make predictions when all information does not necessarily agree.

5.2.2 Distributed and Redundant Storage

Distributed and redundant storage requires that information be stored in multiple places so that data loss affects the predictions as little as possible. As shown in Section 5.2.1, TSDM is able to make predictions without FHS data from all members of the history. This shows that if random FHSs were to be lost from the TSDM, the predictions would be able to still be made with significant context. This concept is important in keeping the size of the TSDM manageable enough to fit on chip, since storing all FHSs on chip is unreasonable when there are millions of unique data accesses in a program.

5.2.3 Prediction Ordering and Prefetching

As mentioned above, TSDM combines the FHSs of multiple addresses in the stream to make predictions and there are multiple ways to interpret the combined FHS. We currently use two different interpretation methods when looking at combined FHSs. Each element of a combination FHS stores an amount of context and a total confidence. The amount of context represents the number of the FHSs, out of all the FHSs that were combined, that contained the address of the element at that current relative stream position. The total confidence is the sum of all the elements confidence that have the same address and relative current stream position. An example of these computations can be found in Figure 5.

The same address would prefetched for the given example regardless of whether higher context or confidence was preferred. It is easy to imagine examples where the two would not agree however, and both interpretations are legitimate ways to order and make predictions. Usually the top prediction is prefetched. If higher degree is desired and accuracy is not an issue, than the top x predictions for each relative current stream positions can be prefetched which would likely lead to higher coverage.
6 Detraining

Data prefetchers are expected to run constantly while the processor is being used. And over this time memory is created, used, deleted and reused constantly. Therefore our prefetcher should be able to learn new information easily when the current information is old. Detraining is the method for the prefetcher to lose information over time, allowing room for new information to be used. We have developed 3 types of detraining, forgetting, pruning and thinning, that are implemented inside TSDM.

6.1 Forgetting

The easiest way to allow new information to be weighted more than old information is to reduce the confidence of old information. Forgetting allows the TSDM to decrease the confidence of every element in every FHS in the TSDM by a very small amount every time a new address is accessed.

6.2 Pruning

When a single relative position in a FHS has too many addresses, that means that over time we see many unique addresses there and therefore are unlikely to make the correct choice. Pruning allows TSDM to delete all elements with a certain relative position in a FHS when their count goes above a certain threshold. This should give TSDM a higher accuracy and a lower number of total prefetches, since no prediction will be had for that relative position after it is pruned. Another advantage that pruning gives is that it limits the size of a FHS, allowing us to place size constraints on TSDM.

Pruning is important in detraining and constraining space, but it can often remove important information. For example imagine a data stream where A is followed by B 100 times and C,D,E,F,G and H one time each. If the pruning threshold is less than 8 than the entire FHS will be deleted while it is clear that B should be prefetched after A.

6.3 Thinning

Removing inconsequential elements of a FHS, or thinning, is a detraining tool with more finer touch than pruning. Thinning only occurs when a FHS reaches the pruning threshold and inconsequential elements are found, therefore information is lost through thinning when there is a threat that even more information would be lost through pruning. This is accomplished by removing individual elements that have a confidence lower than a determined threshold. This threshold is a percentage of the maximum confidence of all the elements in the FHS.

Problems can arise with thinning when it is used without forgetting. When TSDM is used in this way, a FHS can see one pattern so many times that thinning will automatically remove any new patterns. This happens because the new pattern will have a confidence of 1 while the learned pattern has a confidence so high that the thinning threshold is greater than 1. Forgetting helps alleviate this issue because it decreases all confidences over time, allowing for new information to be learned before it is thinned.

7 Implementation

The two components of the TSDM are the sample address vector (SA V) and the content matrix (CM). The content matrix stores a fuzzy historical stream in each row, and the address key associated with that FHS in the corresponding place in the sample address vector. The combination of the SAC and the CM make up the mapping described in Section 5. It is not necessary to store a FHS for every address accessed, since TSDM has data redundancy as shown in Section 5.2.2. Therefore the SAV represents a subset of all memory addresses accessed.

The main operations that TSDM allows are training, predicting and meta-data prefetching.

7.1 Training

TSDM stores correlation information in the form of fuzzy historical streams. These fuzzy historical
Figure 6: TSDM Design with a writing length of 3

<table>
<thead>
<tr>
<th>SAV</th>
<th>Content Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B:1</td>
</tr>
<tr>
<td>C</td>
<td>D:1</td>
</tr>
<tr>
<td>G</td>
<td>H:1</td>
</tr>
<tr>
<td>H</td>
<td>I:1</td>
</tr>
<tr>
<td>M</td>
<td>N:1 O:1 R:1</td>
</tr>
</tbody>
</table>

Access Stream: 

Streams are trained by combining new stream information with the historical stream information.

Training occurs when \( x \) addresses, where \( x \) is the history length, have been accessed since an address in the SAV was. This stream of \( x \) addresses is added to the FHS mapped to by the key address. FHS combination is explained in /srefcontext, but in this instance the context is information is stripped and the resulting FHS only stores a confidence for each address-position pair. The history length determines the maximum amount of context the TSDM can have and how far apart correlations can be made.

An example is given in Figure 6, in which the last 8 addresses accessed are M,N,O,P,M,O,R,P. Since M is in the SAV and the history length is 3, M’s FHS is trained with the stream N,O,P first and the stream O,R,P afterwards. Although O is present in both streams, the information between the two occurrences of O cannot be combined in the FHS since it occurs at relative position 2 in the first stream and relative position 1 in the second. The information for P can be combined since it occurs at relative position 3 in both streams, therefore it ends with a confidence of 2 after the second training call.

7.1.1 Pruning

We have observed that while running benchmarks there are usually a few addresses that accumulate thousands of addresses in their FHSs. This caused TSDM to become unusually slow, therefore we have made pruning mandatory. The example in Figure 6 has a pruning max of 4. Additionally if TSDM were implemented in hardware, the size of each

Figure 7: Updating the Running Prediction Stream

<table>
<thead>
<tr>
<th>Data Access</th>
<th>Prediction Stream Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B:1 C:1 D:1</td>
</tr>
<tr>
<td>Old Pred Stream</td>
<td>B:1 C:1 D:1</td>
</tr>
<tr>
<td>New Pred Stream</td>
<td>B:1-1 C:1-1 D:1-1</td>
</tr>
<tr>
<td>B</td>
<td>C:1-1 D:1-1</td>
</tr>
<tr>
<td>Old Pred Stream</td>
<td>C:1-1 D:1-1</td>
</tr>
<tr>
<td>New Pred Stream</td>
<td>C:1-1 D:1-1</td>
</tr>
<tr>
<td>C</td>
<td>D:1-1 E:1 F:1</td>
</tr>
<tr>
<td>Old Pred Stream</td>
<td>D:1-1 E:1 F:1</td>
</tr>
<tr>
<td>New Pred Stream</td>
<td>D:2-2 E:1-1 F:1-1</td>
</tr>
</tbody>
</table>

Confidence – Context
Assuming an address stream of A,B,C and a TSDM shown in Figure 6

FHS would have to be fixed and that naturally places a limit on the number of addresses that could be stored for each relative position. The simple way to enforce this limit is to employ pruning.

7.1.2 Sample Address Vector Sparsity

The Sample Address Vector is a subset of all unique addresses accessed. The sparsity of this subset is determined by how many non-SAV addresses TSDM can see in a row before adding an address to the SAV. When \( x \) addresses that aren’t in the SAV have been accessed in a row, the last one is added to the SAV. When the sparsity is equal to the history length, this method ensures that the TSDM is trained on all addresses accessed. If the sparsity is greater than the history length, there will be addresses that aren’t trained to the TSDM. A sparsity less than the history length will lead to a more populated SAV and therefore more historical context will be available for predictions. We have found that the sparsity can be increased until it is equal to the history length before performance starts to suffer.
7.2 Prediction

TSDM creates predictions for which addresses should follow in the stream by combining the FHSs of recent key addresses in the stream, as described in Section 5.2. These predictions can be batched or made on an individual basis.

The predictor keeps a running prediction stream, for each PC since streams are PC localized, which is updated whenever an address in the SAV is accessed. Since all data is stored in FHSs mapped to by addresses in the SAV, there is no information to add when an address not in the SAV is accessed. The accessed address’ FHS is added to the running prediction stream, which is updated to reflect the new relative stream positions. All relative stream positions are decreased by 1 for every address accessed, so that the positions represent where we expect to see the predictions. Elements with relative positions less than 1 are removed since there is no need to prefetch the current address or addresses that have already been accessed. An example of this is shown in Figure 7. When the running prediction stream is always updated, addresses can be chosen for prefetching anytime.

TSDM makes predictions for the in batches of \( k \) positions, where \( k \) is the Batch Size. On every \( k \)th access in the PC localized stream predictions for the next \( k \) addresses are made and prefetched. Information for relative positions \( 1, \ldots, k \) are pooled together and the top \( k \times l \) predictions are prefetched, where \( l \) is the Degree. Pooling enables the the next \( k \) positions to be prefetched while ignoring position information, therefore the same address can only be prefetched once per batch. Since there can be multiple occurrences of the same address in a pool, they must be combined by taking the sum of the individual confidences and the maximum context. After combination the pool is ordered by confidence and context and the top \( l \times k \) addresses are prefetched. TSDM supports both ordering by confidence then context and ordering by context then confidence, although the results differ minimally between the two. In the above example, which uses a batch size of 2 and a degree of 1, O and either N or R would be prefetched since O has the greatest confidence and N and R have identical confidence and context.

7.3 Metadata Caching

TSDM stores a linear amount of data relative to the amount of unique addresses accessed. This is too large to store in hardware on chip and therefore must be stored in memory, but TSDM requires faster access to correlation information than memory provides. A small on-chip metadata cache allows TSDM to store relevant FHSs that have been prefetched from memory. A FHS is used whenever it’s key address is accessed, therefore we can expect to use the FHSs mapped to by the addresses that are being prefetched. As long as TSDM has a high accuracy, the performance decrease of metadata caching should be minimal since the accuracy of the metadata prefetching would also be high.

8 Results

The goals of this thesis were to solve the problem of aliasing in irregular data prefetching and create a better irregular prefetcher than is currently available. To test how TSDM addresses both of these goals we have chosen to run it on two types of benchmarks. The first is a micro-benchmark designed to only test de-aliasing and the second are irregular benchmarks from SPEC 2006 and test irregular prefetching overall. ISB was chosen as the prefetcher to compare against since we feel it is currently the best solution.
Table 1: Aliased Micro-Benchmark

<table>
<thead>
<tr>
<th>PC</th>
<th>Address Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X+1</td>
</tr>
<tr>
<td>X</td>
<td>X+2</td>
</tr>
<tr>
<td>X</td>
<td>X+3</td>
</tr>
<tr>
<td>X</td>
<td>X+4</td>
</tr>
<tr>
<td>X</td>
<td>X+5</td>
</tr>
<tr>
<td>X</td>
<td>X+6</td>
</tr>
<tr>
<td>X</td>
<td>X+7</td>
</tr>
<tr>
<td>X</td>
<td>X+8</td>
</tr>
<tr>
<td>X</td>
<td>X+9</td>
</tr>
<tr>
<td>X</td>
<td>X+10</td>
</tr>
<tr>
<td>X</td>
<td>X+4</td>
</tr>
<tr>
<td>X</td>
<td>X+11</td>
</tr>
<tr>
<td>X</td>
<td>X+12</td>
</tr>
</tbody>
</table>

This stream was repeated for 1000 PCs, separated so that no two PCs loaded the same address. This stream of 20000 data accesses, 20 per PC, was repeated 10 times.

Table 2: Aliased Micro-Benchmark Results

<table>
<thead>
<tr>
<th>Prefetcher</th>
<th>History Length</th>
<th>Coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISB</td>
<td>57</td>
<td>54</td>
</tr>
<tr>
<td>TSDM</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>TSDM</td>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td>TSDM</td>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>TSDM</td>
<td>...</td>
<td>80</td>
</tr>
<tr>
<td>TSDM</td>
<td>12</td>
<td>80</td>
</tr>
</tbody>
</table>

8.1 Aliasing Testing

In order to isolate aliasing we have developed a micro-benchmark which contains address streams that are almost entirely aliased. This micro-benchmark is shown in Table 1. It is clear that the current standard of irregular prefetching, ISB, would be unable to prefetch the second or fourth column in the stream. This is because X+1 can only be followed by one address in the structural address space, and X+4 can only be preceded by one address. Therefore X+1 cannot prefetch 4 different addresses in a row and X+4 cannot be prefetched by 4 different addresses in a row.

The historical context in TSDM allows the prefetcher to correctly prefetch every address in the stream, as long as the history length is long enough. With a history length less than 3, X+1 would only have the context of itself and X+4 to make a prefetch decision; however with a history length of 3 or greater, X+1 knows to prefetch X+5 when it is preceded by X+3 and X+4 and X+7 when it is followed by X+6 and X+4.

The results of this benchmark are shown in Table 2. These show what we would expect, that ISB performs reasonably and TSDM performs similarly with a history length less than 3. When the history length is increased to 3 and above the coverage jumps to 80%, for the reasons described above. This shows that TSDM has the ability to solve aliasing, unlike any previous irregular prefetcher.

8.2 SPECint2006 Results

We have tested TSDM and ISB on the irregular benchmarks from SPECint2006 which rely on pointer-based data structures. These benchmarks were tested in the SNIPER simulator.

The results shown in Figure 9 show basic versions of TSDM compared to ISB on individual benchmarks. Accuracy decreases as history length, batch size and sparsity increase, mostly likely since we are batching prefetches that won’t be used for a long time. Since more prefetches are made as batch size increases, the coverage is relatively the same for all prefetchers. The speedup increases dramatically as history length and batch size increases, but this increase is caused almost entirely by batching prefetches and not be providing more coverage. This is proved in Section 8.2.1.

8.2.1 Isolating Context

The variable context results are shown in Figure 10. These tests have no sparsity and a batch size of 1. The results show that additional context of SDM compared to ISB helps speedup and coverage a small amount, around 2%, while slightly decreasing accuracy. Although additional context helps, the rewards start to diminish after a history length of 4. Speedup does not show the same trend as seen in the base results above, therefore we can assume that batching provides the bulk of the speedup increases in Figure 9.

The results in Section 8.1 show that TSDM with a long history length is able to de-alias significantly better than ISB, so we would expect an increase in...
coverage larger than 2% with additional context. A simple explanation for the lackluster benchmark results would be that the SPEC 2006 irregular benchmarks don’t have a significant aliasing, although some analysis of the traces would be required to confirm this.

8.2.2 Metadata Caching

The results in Figure 11 follow those in Section 8.2.1 because the tests are identical except that these are run with a limited hardware budget. The best performing hardware-limited TSDM had a history length of 4, and was able to match the performance of ISB. This shows that metadata caching and prefetching allowed hardware restrictions to be placed on TSDM with only a small performance decrease. The hardware budget doesn’t necessarily reflect all data stored for TSDM, although it does represent a bulk of it. Additional checks need to be added to implement an accurate hardware budget.

8.2.3 Pruning

TSDM results with a variable pruning threshold are shown in Figure 12 and follow closely what we would expect to see. Small pruning sizes, such as 1 and 2, remove most information since many addresses need to be able to train multiple sets of successors. Therefore coverage plummets on this end of the results. On the other end large pruning sizes, such as 7 or 6, have results very similar to the results with no pruning, i.e. pruning size of 0. This is because very few addresses have 7 or 8 unique sets of ancestors, therefore most addresses are able to train without being subjected to pruning. We have found a pruning size of 4 or 5 to be a sweet spot for reducing the size of TSDM while not losing a significant amount of performance.

8.2.4 Forgetting

The variable forgetting results are shown in Figure 13. Since there are often tens of thousands of accesses between when training and predicting occur on the same address, the forgetting constant must be extremely small. Or else the confidence of each prediction would approach zero quickly. These results show a maximum coverage increase at a forgetting constant around $5 \times 10^{-7}$, although the increase is trivial. We believe that forgetting is more important on longer tests since it is more likely that addresses will be re-purposed in a longer trace. In the future we plan on creating longer benchmarks to test this hypothesis.

9 Conclusions

In this paper we have introduced the idea of historical context, which is designed to allow irregular prefetchers to perform better with highly aliased streams. We have implemented TSDM to incorporate historical context, address correlation and PC localization in a practical prefetcher. In order to make TSDM practical we introduced the idea of prefetching metadata so that most of the prefetcher data could be stored in memory.

The idea behind historical context in TSDM is that allowing multiple addresses to have a say in prefetches, we can distinguish similar situations in the address stream. Beyond this historical context is also integral in the implementation of metadata prefetching, since future prediction information and data redundancy allows for metadata cache misses to be less important.

We have shown that TSDM is able to produce a much higher coverage than ISB for a microbenchmark designed to have a significant amount of aliasing. However we have also seen that these results do not transfer to standard irregular benchmarks. We have also shown that metadata prefetching allowed TSDM to be implemented in limited hardware with only a small performance decrease.

10 Future Work

TSDM was designed in a way that it can solve multiple problems besides irregular data prefetching. It is also easy to alter the variables behind TSDM, so changing the metadata prefetching or PC localization is trivial. We have outlined two areas of future
research that we feel TSDM could contribute to and from.

10.1 Prefetching More Information

TSDM provides a new way to look at irregular prefetching, and can be adopted to solve different problems very easily. For example TSDM could be used to prefetch TLB entries or prefetch super-pages. ISB, the only other feasible irregular prefetcher, would be unable to solve either of these problems.

ISB would be unable to prefetch TLB entries since it syncs with the TLB to reduce on-chip hardware. TSDM removes the reliance on TLB syncing by adding metadata prefetching. TLB usage is usually heavily aliased, so TSDM should prefetch its entries well.

When used with super pages, ISB is unable to maintain a reasonable hardware budget. This is because it stores a page of structural addresses, which are TLB resident, into its cache on chip. When a page is a megabyte large, ISB is unable to cache it since it is much larger than ISB’s on-chip cache size. Since TSDM doesn’t load pages of correlation information at a time, the size of the page is irrelevant to how TSDM functions. Therefore TSDM should be able to work with super pages, giving it a leg up on ISB.

10.2 Revisiting PC Localization

Since TSDM is PC localized its batched prefetching is also PC localized, so when batching TSDM could be prefetching addresses that won’t be used for thousands of accesses. By predicting PCs, we could prefetch addresses closer to when they are actually used. This would allow for less data to be stored on chip, by storing PC information in memory and prefetching what is needed as PCs are predicted.

The de-aliasing properties of TSDM allow us to also look at prefetching without PC localization. PC localization can provide too small or too large of a grain for context, reducing the amount of correct correlations TSDM can make. Removing PC localization allows TSDM to dynamically determine the grain size for each address. In order to remove PC localization we would likely have to increase history length significantly in order to catch trends for PCs that occur sparsely.

Acknowledgments

My thesis advisor is Calvin Lin. Akanksha Jain has provided invaluable insight and help throughout this entire process.

References


The prefetchers SDM-1 through SDM-16 represent implementations of TSDM where history length, batch size and sparsity are all equal. So SDM-1 has a history length, batch size and sparsity of 1 while SDM-16 has a history length, batch size and sparsity of 16. SPEC 2006 benchmarks that mostly contain regular data streams have been removed since TSDM and ISB are irregular prefetchers.
This graph shows variable context and pruning for TSDM in comparison to the baseline ISB. In order to isolate context, the history length was set to the desired amount of context and the batch size and sparsity were set to 1. Therefore for every address in the lookback history should have input into each prefetch decision, so the amount of context is equal to the history length.
These tests were run under the same conditions as the tests shown in Figure 10, but the hardware budget was limited for these. The TSDM was limited to 32 KB of on-chip storage for content matrix rows, and the latency for retrieving rows from memory was 0.
The prefachers tested are the same as the ones used in Figure 9. So history length, sparsity and batch size are set to be equal for each prefetcher.
The prefetchers tested are the same as the ones used in Figure 9. So history length, sparsity and batch size are set to be equal for each prefetcher.