**STUDY OF CURRENT AND PROPOSED PREFETCHING ALGORITHMS**.

**Prepared by:**

Sourav Ranjan

Email Address: [sourav.ranjan@ucdenver.edu](mailto:sourav.ranjan@ucdenver.edu)

Sai Vyshnave Koduru Srinivasa

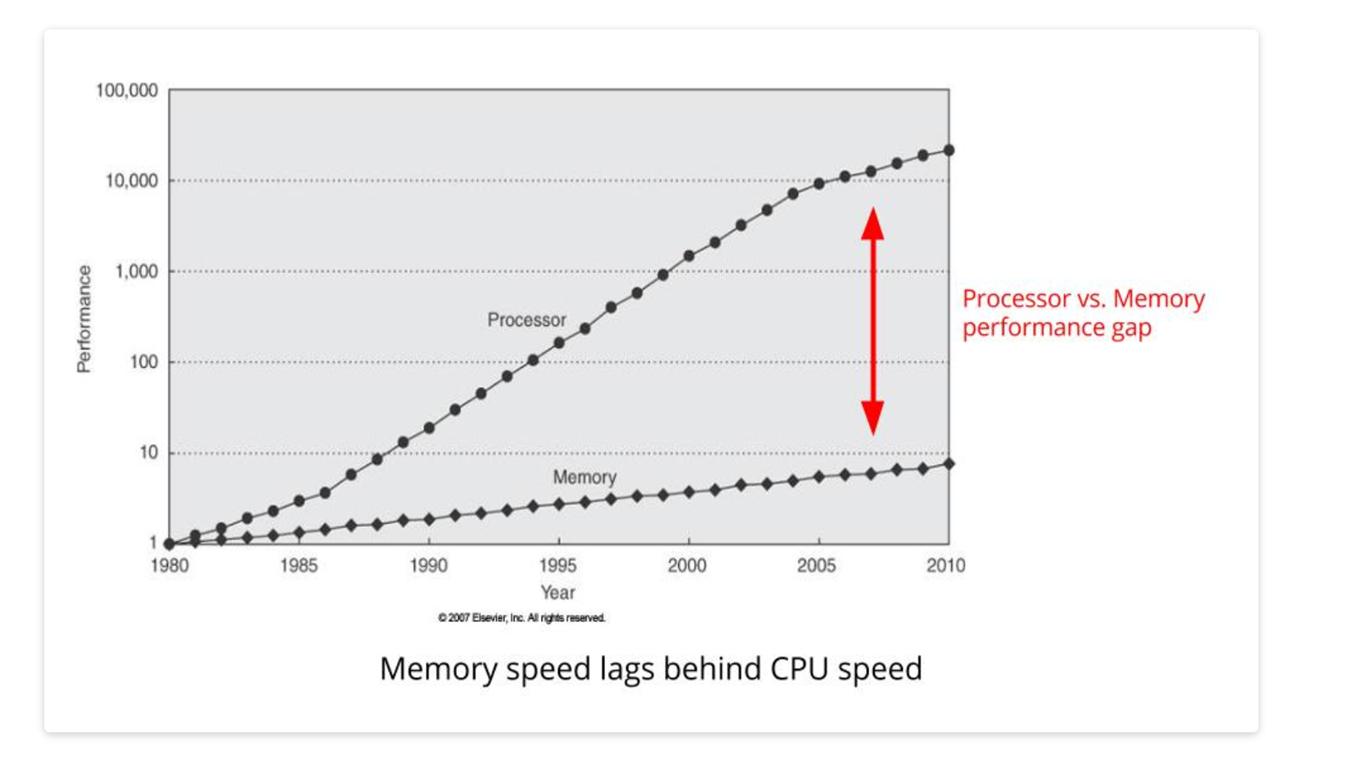
Email Address: [SAIVYSHNAVE.KODURUSRINIVASA@UCDENVER.EDU](mailto:SAIVYSHNAVE.KODURUSRINIVASA@UCDENVER.EDU)

Sapthagiri Venkat Raghuraman

Email Address: [SAPTHAGIRI.VENKATRAGHURAMAN@UCDENVER.EDU](mailto:SAPTHAGIRI.VENKATRAGHURAMAN@UCDENVER.EDU)

**ABSTRACT**: *As processors have greatly increased in speed over time, improvements to memory access speeds have been less significant. This creates a situation in which the processor is frequently waiting long periods of time for the memory system to provide it some data. Caches are a powerful and commonplace strategy for mitigating this inefficiency, but cache misses still constitute one of the major hindrances on processor performance. One technique for further reducing memory stalls is to prefetch data that will be needed soon into the cache. While prefetching is potential powerful, it must be employed carefully because prefetching data which already resides in the cache or which is not actually needed will degrade performance. With systems using cache we see if there is a cache miss it imposes a severe penalty on the performance of processor as there are not enough instructions for the processor to execute. To avoid this situation, processors use prefetching to reduce cache misses which is of two types: instruction prefetch and data prefetch. We will also dive into hardware vs. software cache prefetch and compare their metrics. The prefetching schemes have been designed to avoid cache pollution and bandwidth waste. In this paper, we have compared the effectiveness of the prefetching algorithms. They are reference prediction table, delta correlating prediction table and differential finite context methods prefetcher. Based on the results DCPT has 51% mode speed up when compared to the other prefetchers.*

**INTRODUCTION**

Processor performance seems to follow Moore’s Law, increasing by 60% a year, while memory performance increases by just 7% each year. .This was our first motivation for looking for new and effective strategies to reduce the large gap between processor speed and memory speed. 

The problem of memory latency is observed .In order to reduce the memory latency problem the concept of prefetching is used .The data’s and the instructions are requested speculatively before the processor needs it .We call the prefetcher as a good one when there are reduced number of cache missed and an increased hit ratio. If a prefetch is made too early before even it is required the process of memory latency is increased. The report briefly describes three prefetching schemes they are reference prediction table, delta correlation prediction table and differential finite context method prefetcher.

**PREFETCHING SCHEMES**

1. **Reference prediction table:**

It is a strided prefetcher proposed by Tien-Fu Chen and Jean-Loup Baer .It is a large table where every entry is indexed by the program counter.

|  |  |  |
| --- | --- | --- |
| **Program counter** | **Last address** | **delta** |

fig: RPT table format

Firstly ,an empty table is created .When there is a cache miss the table is accessed to find the entry indexing the current program counter. When there is no entry found a new entry is created ,If an entry is found the last effective address which is stored will be used to calculate the new delta . When the delta of the current address is equal to the delta of the last address then the new prefetch is issued .The Reference prediction prefetching algorithm is used when a single prefetch instructions is issued at the same time. The cache block prediction can be done one iteration in advance .It is a single table entry .The total size of the RPT table is 144 bits.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **No. of bits** |
| PC | Uint | 64 |
| Last address | Uint | 64 |
| Delta | Int | 16 |

1. **Delta correlating prediction table:**

The delta correlating table is inspired by the RPT and the Pc/Dc prefetching schemes. The DCPT is same as the RPT but it has an additional table which stores the last prefetch values. The DCPT table consists of a circular buffer which stores the n delta values for the correlation.

Initially the program counter checks for the corresponding entry in the table. If the entry is found a new entry indexing to the current PC is created .After the creation the Last address is set to the current cache line miss address .Then the delta value is calculated by considering the current address and the last address .If the delta is equal to zero then it is ignored.If the new delta is not equal to zero then the updation of the circular buffer takes place .

|  |  |  |  |
| --- | --- | --- | --- |
| **Program counter** | **Last address** | **Last prefetch** | **Delta1….n** |

fig: RPT table format

The delta correlation algorithm begins as soon as the entry gets updated .The process of backward matching is done to find the striding pattern .In order to find the match the most recent deltas are compared ,If there is a match then they are added to the temporary prefetching candidate buffer .After the delta values are added to the candiadate buffer they are compared with the set of instructions like content of the cache,Miss status holding register and the pending buffers which holds the prefetching requests which are still not finished.This follows FIFO buffer.The total size of the DCPT is 576 bits . DCPT needs 4 times more storage than RPT ,DCPT holds a sequence of deltas for each entry , and RPT stores only the last calculate deltas.

|  |  |  |
| --- | --- | --- |
| **Name** | **type** | **No of bits** |
| **PC** | **uint** | **64** |
| **Last address** | **uint** | **64** |
| **Last prefetch** | **uint** | **64** |
| **Delta buffer** | **unit** | **64\*n** |

1. **DIFFERENTIAL FINITE CONTEXT METHOD PREFETCHER**

The differential finite context method prefetcher is a markovian prefetching technique. The algorithms requires two tables for the implementation they are the history table (HT) and delta table (DT).The Hash table is indexed by a PC tag.

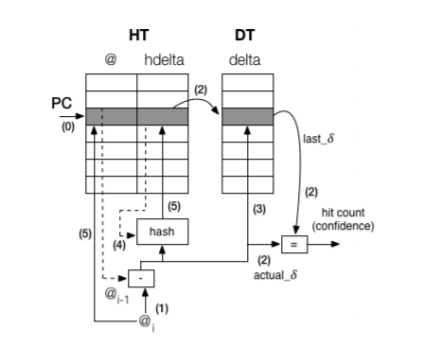
|  |  |  |
| --- | --- | --- |
| **Name** | **type** | **No of bits** |
| **PC** | **uint** | **32** |
| **Last address** | **uint** | **32** |
| **Hashed delta sequence** | **uint** | **32** |

**Fig: entry data format for HT**

|  |  |  |
| --- | --- | --- |
| **Name** | **type** | **No of bits** |
| **Delta buffer** | **unit** | **64\*n** |

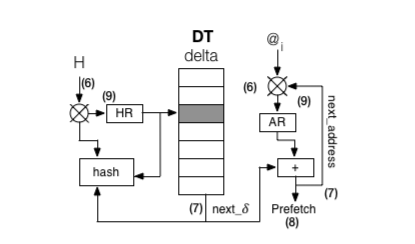
**Fig: entry data format for DT**

The delta values are hashed because it will save lot of space .The PDFCM has two stages they are the update stage and the prefetch stage .An entry is added in the Hash table in the update stage .The PDFCM aims to predict the next occurrence in a pattern .



**Fig: PDFCM update stage**

It checks if the memory instruction is a cache miss or not .If it is not a cache miss then the actual delta value is computed by reading the last delta value through indexing of DT .The delta table is updated for every entry.Then it looks for a match if there is no match same process conitues then the history register is configures with the new computed values they are H and the AR register.After updating the prefetching process takes place .The prefetching is explained with the help of the diagram. For every epoch nearly 2^14 iterations where used .



**Fig: PDFCM prefetch stage**

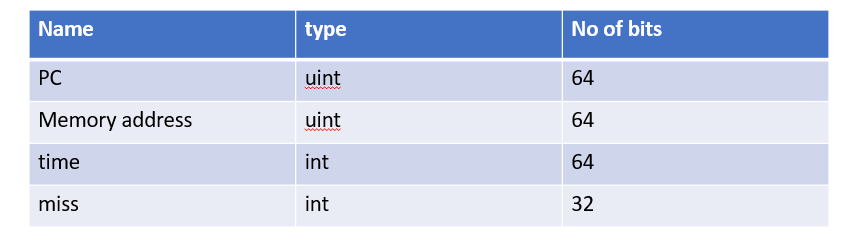
**IMPLEMENTATION AND RESULTS**

We implemented handlers for two events in the M5 frame-work.

* prefetch complete() - triggered when prefetched memory arrives in the L2 cache.
* prefetch access() , triggered at the beginning of each CPU cycle.

With each processed event we receive a snapshot of the system state: program counter, cycle counter and current memory address being requested. We also use the GEM5 API to check the MSHR and cache for addresses.

The spec benchmarks where used which focuses mainly on the image ,compression ,regression etc. The spec was installed in the ubuntu latest version and all the outputs where obtaines .All the outputs are tabulated and compared .The comparison is represented in the form of the graph for better understanding.

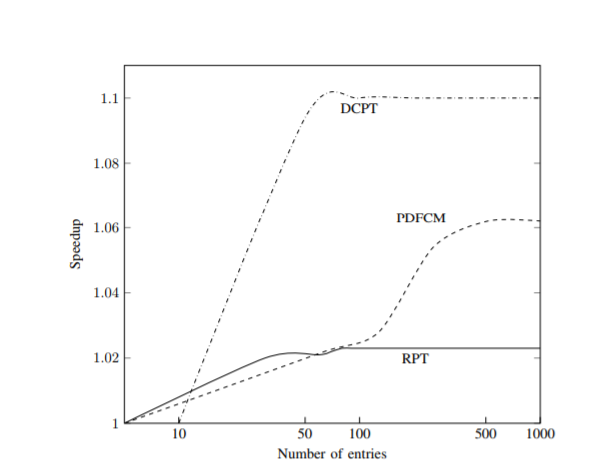
****

We have three important attributes they are speed up,accuracy and coverage.The attributes are used and the prefetchers are compared for different benchmarks.

**Speed up:**

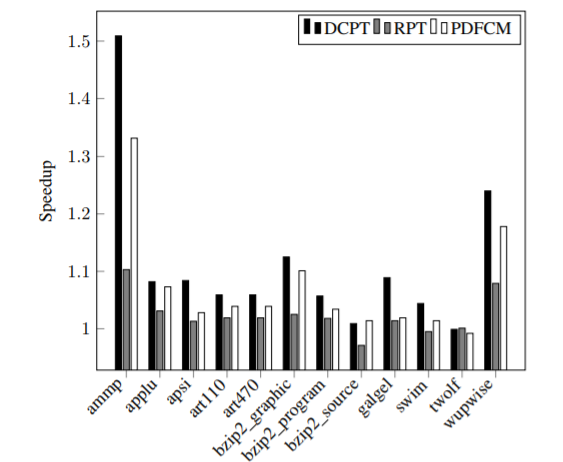
**Speedup = IPCwith prefetch / IPCwithout prefetch**

If the speed up is larger than 1 , Program issued more instructions without prefetchers, i.e more computation is done. The speed up is deceptive.

****

**Fig: difference in speedup depending on the number of entries.**

If we observe the output ,There was a steady increase in the DCPT for the first 50 entries ,after that it remains constant for the next 1024 entries. For PDFCM, the amount of entries in the DT and HT should be close in powers of two to avoid memory aliasing. More entries in the table should increase accuracy. To maximize the amount of entries possible for 16-, 32- and 64-bit variables for an L2 cache with 8KB of memory, we can have 1024 entries in each table for 16 bits, 512 with 32 bits, and 256 with 64 bits. We find that the best score is when using a 32 bit configuration. This indicates that using 64 bits in each entry introduces memory aliasing due to loss of information when hashing deltas.

****

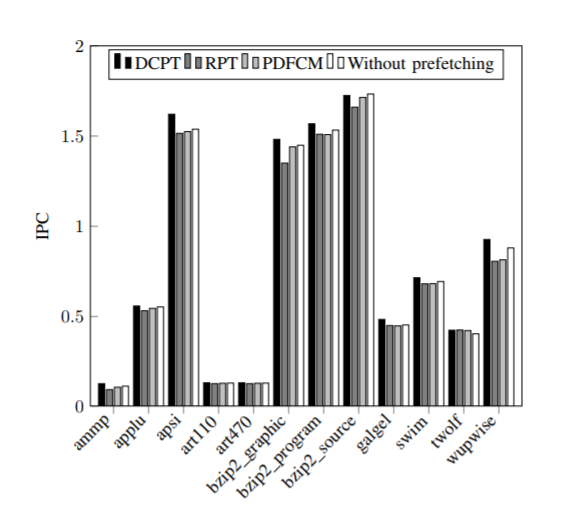
**Fig: Speedup for each benchmark test relative to no prefetching**

DCPT clearly has better speedup when compared to the other schemes, with up to 51% speedup. DCPT outperforms RPT mainly due to: RPT not being able to detect the same access patterns as DCPT, because DCPT holds a history of several deltas while RPT only holds the last calculated delta. Also RPT is more conservative with respect to issuing prefetches compared to DCPT. PDFCM also has less coverage and accuracy than DCPT, assumably because of small table sizes and loss of accuracy in the hashing function.

**COVERAGE :**

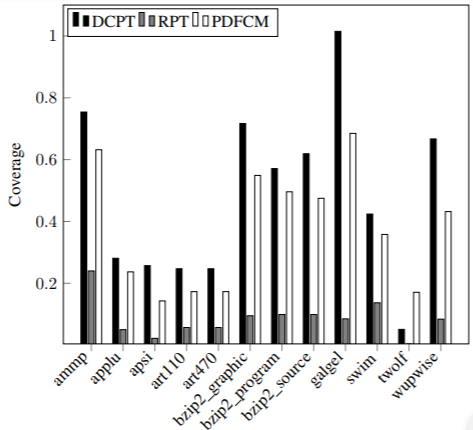
Program avoids accessing memory, IPC count will begin increasing which leads to spinlock .In order to accommodate this the concept of the coverage is used. High coverage then the prefetcher predicts most of the memory access issued. Low coverage occurs if no prefetcher was made. A high speed up with low coverage then good results are achieved by prefetcher .

**Coverage = Number of useful prefetches/ Number of cache misses without prefetch**

****

**Fig: IPC for each benchmark test**

Comparisons of the IPC between the implemented prefetching schemes and the IPC without prefetching. The figure shows that the IPC of all three schemes are improved in almost each benchmark. Since the speedup is calculated using the IPC without prefetch, described using Equation , the improvement of IPC is reflected upon in Fig.

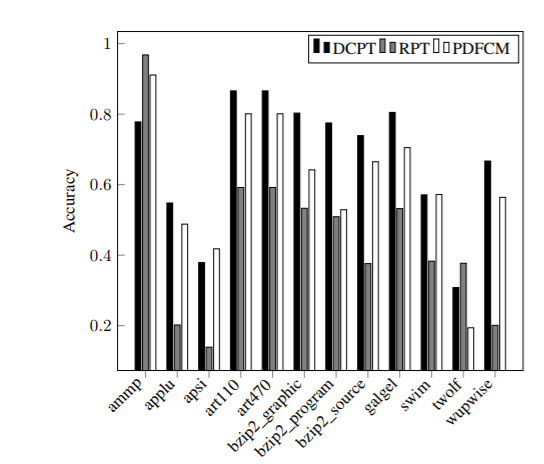
****

**Fig: Coverage for each benchmark test. DCPT outperforms the coverage for both the other methods.**

**ACCURACY**

High accuracy then the prefetcher data are not needed by the application or we can also say that the data is not fetched at right time. If Low accuracy then it is due to cache pollution. The prefetched data will be replaced by more useful data .

**Accuracy = Amount of useful prefetches / Amount of prefetches**

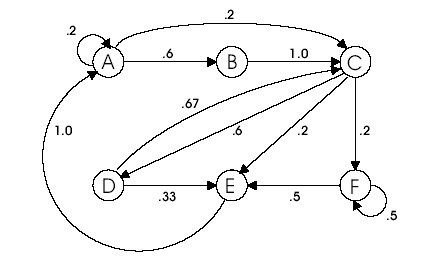
****

**Fig: Accuracy for each benchmark test.**

We can see that test results for RPT are affected by cache pollution and bad timeliness. When RPT does not detect the access pattern, there is a reduction in coverage and accuracy. Cache miss increase which will make the load instruction take longer time to fetch the required data from the main memory. Thus the average amount of instructions executed per cycle will decrease.

# Basic Markov Prefetching

* Markov prefetching forms address correlations
* Uses global memory addresses as states in the Markov graph
* Correlation Table *approximates* Markov graph

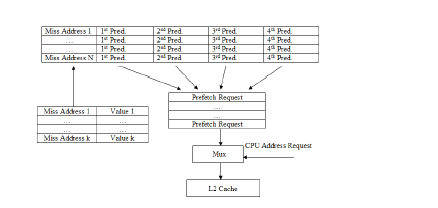


Miss addresses:

A B C D C E A C F F E

A A B C D E A B C D C

Design of Modified Markov Predictor  
The first modified predictor consists of the original design plus an additional table used to detect pointer references. Also, there are more predictions associated with each miss address. The goal of the modified design is to limit bandwidth usage by only adding entries to the table when pointer references are detected. This adds more complexity to the predictor, but it should improve prefetch accuracy and limit cache pollution. The design of the modified predictor is shown in Figure 3. An entry is added to the pointer table when a cache miss occurs. Once the value for the address is known, it is added to the value field in the table. When a cache miss occurs, the predictor checks if the address is the same as any of the value fields. This situation occurs when a pointer is referenced, and the original address is added to the prediction table. The next cache misses that occur become part of this prediction entry. The rest of the predictor works the same way as the original design.



The second modification will be to add a stride prefetcher on top of 1st Modification described above. Stride Prefetchers Chen and Baer investigate a mechanism for prefetching data references characterized by regular strides. Their scheme is based on a reference prediction table (RPT) and look-ahead program counter (LPC). The RPT is a cache, tagged with the instruction address of load instructions. The entries in the RPT hold the previous address referenced by the corresponding load instruction, the offset of that address from the previous data address referenced by that instruction, and some flags. When a load instruction is executed that matches an entry in the RPT, the offset of the data address of that load from the previous data address stored in the RPT is calculated. When this matches the offset stored in the table, a prefetch is launched for the data address one offset ahead of the current data address. The reference address stream was used to index the reference prediction table, In practice, we found little performance difference between using the reference addresses or the miss address stream. Our later simulations of stride prefetchers use the miss address stream.

Currently We have not been able to implement above mentioned Prefetcher based on modified Markov Model.