Implementation and comparison of two cache prefetching schemes - 'Prefetch on a Miss' and 'One Block Lookahead'

Akash Thyagaraja
Department of ECE
University of Florida
akashthyagaraja@gmail.edu

Anushka Swarup
Department of ECE
University of Florida
aswarup@ufl.edu

Medini Aradhya
Department of ECE
University of Florida
medini.aradhya@ufl.edu

Varanasi Naga Venkata Sai
Department of ECE
University of Florida
varanasi.n@ufl.edu

***Abstract*—This paper presents the simulation and subsequent analysis of two cache prefetching schemes on various test-benches.It shows the implementation of 'Prefetch on a miss' and 'One Block Lookahead' schemes on Instruction Level 1 and Data Level 1 caches respectively.The source code has been implemented on the Simplescalar LLC simulator. Considerable improvements were observed when either scheme was implemented especially for high sequential instruction benchmarks. Furthermore,performance of 'One Block Lookahead' was better than 'Prefetch on a miss' scheme. It can subsequently be concluded that implementing such schemes can considerably reduce compulsory cache misses in L1 cache.**

***Index Terms*—Prefetch on a miss, One block lookahead, Cache prefetching, Simplescalar.**

I. MOTIVATION

Over the past decade, microprocessor performance has boomed at a dramatic rate. This trend was fueled by the rapid developments in semiconductor and fabrication technologies and has therefore led to decreased clock speeds. In contrast, the development in main memory RAM has been at a very leisurely rate as show in Figure 1. [1]

This expanding gap in disparity between the two opposing trends has necessitated aggressive techniques to improve the performance of the DRAM. These techniques are primarily focused on the structure of the cache hierarchy and reducing number and the latency of memory accesses. Cache memory hierarchies mainly focus on static RAM (SRAM) whose main purpose is to service and keep pace with the processor request rates. This is not an easy problem to solve due to the limitations in cache capacity and in effect the associativity. Increasing the size of SRAM is not only too expensive but not practical for general purpose architectures. One way to solve this problem is by a computer architecture technique called Cache Prefetching.

II. INTRODUCTION

Cache prefetching is a technique used by computer processors to boost execution performance by fetching instructions or data from their original storage in slower memory to a faster local memory before it is actually needed or requested for(hence the term 'prefetch') [2].

Cache prefetching works on the concept of calling forth blocks that would most likely be used by the CPU for subsequent processing.It is able to do this by various methods of speculation, but works most effectively and with greatest performance for sequential instruction programs like loops that are recursively executed and sequential data access operations eg: operations on Vectors. But currently the poor cache memory utilization is one of the main reasons we have seen a high demand in development and use of memory fetch policies. This policy is responsible for speculating both data and instructions and placing them into a cache extension known as the stream buffer that can be then called forth during instruction execution. Data accesses are harder compared to instruction accesses as recursion or repetitive instructions usually require different data whereas the trend instructions, to a certain degree, can be easily predicted by the prefetching scheme. Failure to develop a sophisticated prefetching scheme might result in capacity and conflict issues in the cache.Where capacity corresponds to unavailability of any free blocks in the cache and conflict implies that the multiple data elements map to the same block that would lead to an erroneous result. Cache prefectching schemes seem to provide a huge reduction in compulsory misses, however this must be implemented with utmost attention to detail as any misstep would lead to wastage of an already scarce resource of cache. This would in-turn lead to excessive conflict and capacity misses thereby degrading the overall performance. [3]

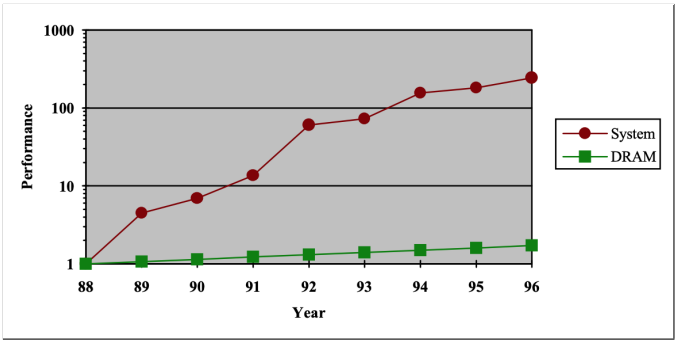

Fig. 1: System/Memory Performance over the past decade

III. METHODOLOGY

Implementing the concept of prefetching involves generation of a 'Stream buffer'.This is one of the most important

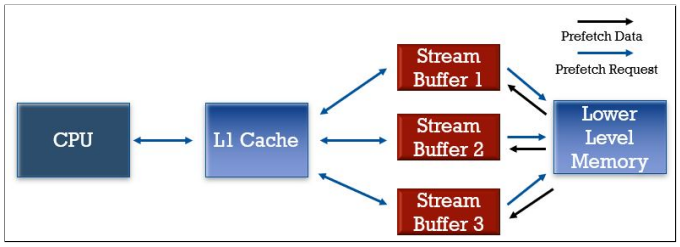

Fig. 2: Stream Buffer Structure

concepts developed by smith et.al.

In essence, the stream buffer is an extension of the cache that is intended to work as follows: Post a cache miss the address of the cache that was missed and the next 'n' addresses are stored in a stream buffer sized n.

The stream buffer can either be part of the cache or designed separate from the cache memory. Some architectures have also been found to contain more than one streaming buffer having its own dedicated memory locations.Implementing such buffers would lead to lesser latency in accessing the required data subsequently improving the hit rate of the architecture. The size, depth and design of the streaming buffer depends largely on the microarchitecture and the type of applications the system executes. [4]

Prefetching can be broadly classified into two categories, hardware prefetching and software prefetching. Hardware prefetching requires no user intervention and is designed before the fabrication whereas software prefetching requires certain degree of user intervention.

In hardware prefetching,instruction to prefetch is based on the current cache block.Prefetches in Hardware prefetching schemes are handled dynamically by the hardware at runtime, this means that there is no compiler action required which is an added advantage although this would mean increased hardware complexity to accommodate this concept.

Majority of the prefetching schemes have targeted sequential data/instruction sets.example: Loops and vector operations where the the subsequent data/instruction is easily predictable owing to its sequence. However,prefetch request is only sent out once the pattern or sequence is detected.Only if the prefetched instructions are right can it be optimized to mask the main memory latency. Some of the earlier systems use a history table to keep a log of what was prefectched previously.These tables are used to improve the accuracy of the prefetching scheme.Access keys are used to access these tables.

Since prefetching has been a pivotal concept in computer architecture, attempting to implement such schemes would provide a deeper understanding on these concepts.This project deals with the implementation of the afore mentioned schemes on the simplescalar simulator.

IV. Implementation

In this research two cache prefetching schemes were implemented, pre-fetch on a miss and one block lookahead. The aim was to analyse how these prefetching schemes would benefit the performance in terms of effect miss-rates. To simulate

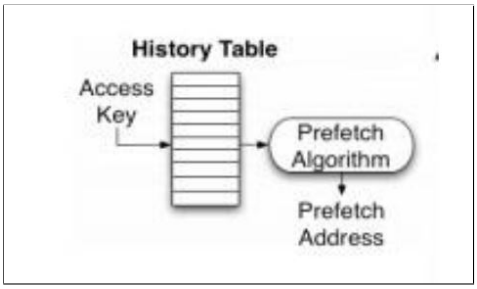

Fig. 3: History Table

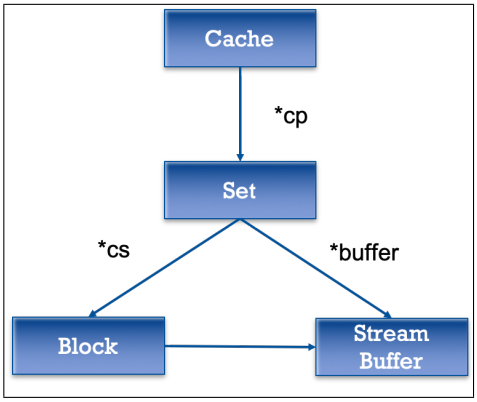

Fig. 4: Pointers Used

the cache behaviour, the Simplescalar computer architecture simulator was used. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction [5]. It includes tools like sim-profile, sim-safe, sim-outoforder, sim-fast. The implementation involves a two level cache simulation using inbuilt C source codes. The designed cache structure is as follows:

- Data Level L1 Cache - dl1
- Instruction Level L1 Cache - il1
- L2 Cache - ul2

The source code for the default cache built in by simplescaler's sim-cache tool was employed for carrying out the experiments. For implementing the two prefetching schemes, the source code was modified by adding appropriate data structures and pointers to simulate the stream buffer behaviour. The following pointers were added for this purpose:

- cp - cache pointer
- cs - cache set pointer
- buffer - stream buffer

In the next subsections, explanation of the two prefetching schemes has been detailed along with the testbenches used to carry out the experiments.

A. Prefetch on a Miss

In prefetch on a miss scheme, if a block 'n' is missing from the cache, the 'n'th & the 'n+1'th block is fetched from the lower level memory.Now, block 'n' goes to cache and 'n+1' goes to the stream buffer.Considering sequentially

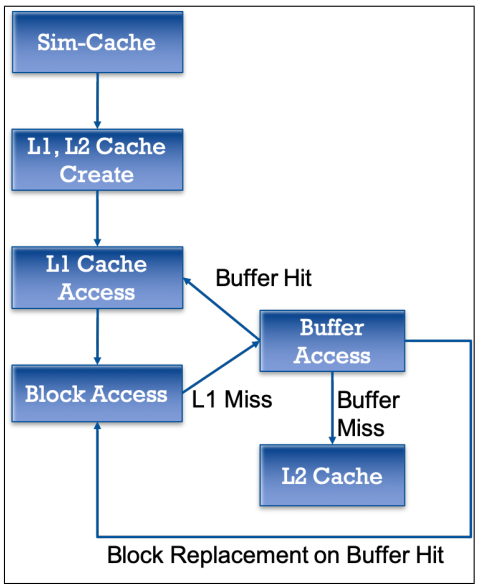

Fig. 5: Flowchart for Block Access

accessed data and according to the principle of locality the likely hood of subsequent access's being present in the stream buffer would be high.In this way the scheme helps in lowering down the compulsory misses as on any consecutive requests the stream buffer registers a hit.

In our implementation we made use of three data pointers to access the cache and the buffer. The code checks for hits in the L1 cache. If it registers a hit, the block is sent to the requested processor. This is termed as slow hits. In case of a miss in the L1 cache the block is requested from the L2 cache along with the next consecutive block which is placed in the buffer.

Prefetch on a miss is considered a desirable scheme for instruction level caches because instructions are more likely to be fetched in blocks and thus getting the n+1 block is beneficial. Also, this schemes only brings data to the cache on a miss. This removes the danger of filling up the cache with unwanted data and thereby reduces the possibility of capacity and conflict misses.

B. One Block Lookahead

One block lookahead is another type of prefetching scheme which is used to reduce the miss penalty and bus latency issues that are present while using the prefetch on a miss scheme. One block lookahead differs from prefetch on a miss in terms that in this scheme the subsequent blocks are brought to the cache irrespective of whether it is a hit or a miss. This scheme is more popular with respect to data access since they are more frequent than instructions.Bringing in a block irrespective of a hot or miss could prove to be detrimental if the data accesses of the prefetched block is incorrect as it would wastefully occupy lot of space in the cache/streambuffer.

C. Testbenches

The simulated cache structure was tested on various test-benches to analyse the cache behaviour on different kinds of datasets and functions. The list of the testbenches used for the current analysis are as follows:

- anagram: finds anagrams in phrases
- factorial: factorial calculation of a number
- test-math: math operations on integers
- test-llong: math operations in long datatype
- test-lswlr: printing strings

V. Results & Conclusion

Experimental analysis of the developed prefetching schemes was tested using caches simulated using Simplescalar. The software simulates 256kb data and L1 caches with 32 sets.

The developed schemes were tested on testbenches provided by simple scalar and a self generated testbench for factorial calculation.The "Miss Rate" of the L1 cache was used as the performance matrix for this study.

It was found that the miss rates reduced considerably for all caches when prefetching techniques were used. This is evident from Fig. 6 & 7. Although, percentage reduction in miss rate was more with instruction cache IL1 as compared to data cache ID1. Also, it was observed that there was great reduction in the number of miss rates for the factorial testbench. One possible reason for this could be the recursive nature of the factorial program with need for adjacent blocks sequentially.Thus the principle of locality aids in the process of prefetching.

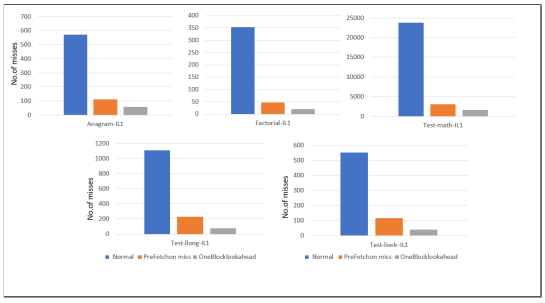

Fig. 6: Results: No. of misses in IL1 V/s prefetching schemes.

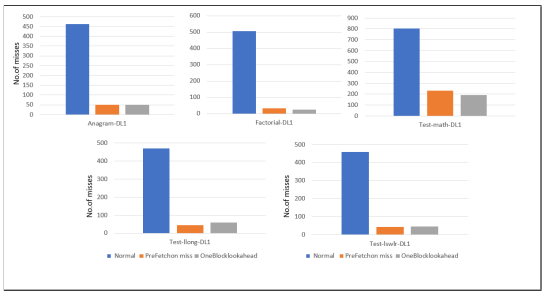

Fig. 7: Results: No. of misses in ID1 V/s prefetching schemes.

	IL1 With Prefetching		IL1 Without Prefetching		DL1 With Prefetching		DL1 Without Prefetching	
Test Bench	Accesses	Misses	Accesses	Misses	Accesses	Misses	Accesses	Misses
Anagram	7740	112	7740	573	4232	50	4232	462
Factorial	7771	47	7771	353	4207	32	4207	506
Test-math	213691	3065	213691	23763	57479	233	57479	805
Test-llong	29623	227	29623	1109	10481	45	10481	471
Test-lswlr	8858	115	8858	550	4738	44	4738	459

TABLE I: Prefetch on a Miss

	IL1 With Prefetching		IL1 Without Prefetching		DL1 With Prefetching		DL1 Without Prefetching	
Test Bench	Accesses	Misses	Accesses	Misses	Accesses	Misses	Accesses	Misses
Anagram	7740	59	7740	573	4232	50	4232	462
Factorial	7771	19	7771	353	4207	25	4207	506
Test-math	213691	1549	213691	23763	57479	191	57479	805
Test-llong	29623	77	29623	1109	10481	59	10481	471
Test-lswlr	8858	39	8858	550	4738	46	4738	459

TABLE II: One Block Lookahead

REFERENCES

[1] "Cache prefetching," Dec. 2019, page Version ID: 929623429. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cache$_prefetchingoldid = 929623429

[2] A. J. Smith, "Cache Memories," *ACM Comput. Surv.*, vol. 14, no. 3, pp. 473–530, Sep. 1982. [Online]. Available: http://doi.acm.org/10.1145/356887.356892

[3] "Figure 1 from Data Cache Prefetching Using a Global History Buffer | Semantic Scholar." [Online]. Available: https://www.semanticscholar.org/paper/Data-Cache-Prefetching-Using-a-Global-History-Nesbit-Smith/6c06f0023fce703f948ae41668875d2f7ad35ea6/figure/0

[4] S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Computing Surveys*, vol. 32, no. 2, pp. 174–199, Jun. 2000. [Online]. Available: http://portal.acm.org/citation.cfm?doid=358923.358939

[5] "SimpleScalar LLC." [Online]. Available: http://www.simplescalar.com/