Feedback Directed Prefetching

Aashi Manglik (13006)
Pramod Chunduri (13221)
Kriti Joshi (13358)

***Abstract*— Data prefetching is one of the crucial methods of hiding load latency in modern processors. However, prefetching useless data wastes resources such as memory bandwidth, cache or prefetch buffer space and leads to excess energy consumption. If prefetched data is not accurate, it also leads to pollution of cache, thereby having a negative impact on performance. Hence, accurate prediction of addresses to prefetch is important. Even if the prefetch requests are accurate, they will not be able to increase performance if they are prefetched late, i.e., not prefetched by the time a load or store instruction demands the data. In this project, we simulated a stream-based data prefetcher and dynamically modified its agressiveness using three attributes - prefetcher accuracy, timeliness, and cache pollution.**

I. INTRODUCTION

Predicting the address that will be needed in future and bringing it early into the cache is referred to as prefetching. This can eliminate compulsory misses and reduce memory latency. Prefetching can be done by recognising the pattern in memory requests. Though prefetching mispredicted addresses does not affect correctness, it can deteriorate the performance.

Prefetch distance and prefetch degree determine the aggressiveness of the prefetcher. Prefetch degree is the number of cache blocks requested (as prefetch requests) per demand access. Prefetch distance indicates how far ahead of the demand access stream the prefetcher can send requests. In conventional prefetcher, these two parameters are fixed throughout whereas in the feedback directed prefetching these are changed dynamically for better peformance. In our implementation, the data is prefetched into L2 cache - 128KB 8-way set-associative with LRU replacement.

II. FEEDBACK DIRECTED PREFETCHING

A. Metrics of Prefetcher Performance

1) Prefetch Accuracy: The prefetched cache blocks that are accessed by the instructions later are referred to as used prefetches. When a cache hit occurs, we check if the cache block has come due to prefetching and increment the number of used prefetches.

$$\text{Prefetch Accuracy} = \frac{\text{Used prefetches}}{\text{Prefetches sent to memory}}$$

2) Prefetch Lateness: A dummy Miss Status Holding Register (MSHR) is implemented by capturing the cache transactions. This is done mainly because of the restricted environment provided by the dpc2sim module. The simulated MSHR stores in-flight memory requests to find if the accurate prefetch request arrived in time. An entry in this register is allocated when a prefetch request is sent to memory - the valid bit is set and address gets stored. The valid bit is reset when the block is inserted into L2 cache. If a cache miss occurs, the valid entries in register are compared with the requested address. If the entry is found, number of late prefetches is incremented.

$$\text{Prefetch Lateness} = \frac{\text{Number of late prefetches}}{\text{Used prefetches}}$$

3) Cache Pollution: To track cache pollution, a bit vector of size 4096x1 is initialised with zeros and referred to as pollution filter. This vector is indexed by taking a xor operation of lower 12 bits [11:0] and next 12 bits [23:12] of the cache block address. If a cache block is evicted to accomodate prefetched data and this evicted block was brought into cache through a demand miss, the bit corresponding to this evicted block address is set to 1 in the pollution filter. When a L2 cache miss occurs, the pollution filter bit accessed from the demand address is checked. If it is set, it means that the block would have been there if no prefetcher was used. This bit is then reset. In such case, the counter indicating demand misses due to prefetcher is incremented. Another counter tracks the total number of L2 cache misses. Cache pollution metric is the ratio of misses due to prefetcher to total number of cache misses.

$$\text{Cache Pollution} = \frac{\text{Demand misses due to prefetcher}}{\text{Demand misses}}$$

B. Sampling-based Metrics Updation

We defined an interval of the program execution time at the end of which the above metrics will be recomputed to decide the aggressiveness. The length of interval is based on the number of cache blocks evicted from L2 cache. The maximum number of cache blocks is 4096. The evicted count is initialised to 0 at the beginning of each interval and when it exceeds a certain threshold, we decide whether to modify the aggressiveness (increase or decrease) as illustrated in section II-C. Table III gives the instructions per cycle on each trace for different interval thresholds. This was done to empirically decide the appropriate interval

length for feedback collection.

C. Dynamically Adjusting Prefetcher Aggressiveness

We used the metrics described in section II to adjust the aggressiveness of prefetcher. There are 5 possible configurations declared in Table I and we choose one of these configurations dynamically as proposed in Srinath et al [1]. For instance, the prefetching starts with third configuration (prefetch distance = 16, prefetch degree = 2). Then after some interval, if the prefetcher accuracy is high ($\geq$ 75%), cache is not-polluting (cache pollution $\leq$ 0.25) but prefetch lateness is high, we should increase the aggressiveness to increase timeliness.

TABLE I
PREFETCHER CONFIGURATIONS

Configuration	Prefectch Distance	Prefetch Degree
1	4	1
2	8	1
3	16	2
4	32	4
5	64	4

If the cache pollution is high, the prefetcher is made less aggressive (decreasing prefetch distance, prefetch degree or both) to reduce pollution and to save memory bandwidth.

III. EXPERIMENTAL SETUP

The prefetcher simulation has been done on the infrastructure provided in the Data Prefetching Championship 2015. This setup provides traces of 8 famous benchmarks of the SPEC CPU2006 suite, namely gcc, GemsFDTD, lbm, leslie3d, libquantum, mcf, milc and omnetpp. We simulated 100000000 instructions for the given benchmarks after a warmup of 10000000 instructions.

IV. EXPERIMENTAL RESULTS

The table below shows the Instructions Per Cycle (IPC) values for three scenarios, one where no prefetching happens, second with a stream prefetcher and third with the stream-based feedback directed prefetcher (FDP).

TABLE II
PERFORMANCE COMPARISON

Benchmark	No prefetch	Stream prefetch	FDP
gcc	0.291329	0.275646	0.291321
GemsFDTD	3.444995	3.430123	3.430123
lbm	1.057604	1.259321	1.079090
leslie3d	0.985227	1.031766	1.086480
libquantum	3.148157	3.169687	3.169687
mcf	0.344332	0.341985	0.341907
milc	0.980482	1.053728	0.986086
omnetpp	2.191883	2.210184	2.203755

Based on the results, the following inferences can be made:

- The feedback directed prefetcher we have used is not generally outperforming a stream based prefetcher. However, it is still a good improvement over a no prefetch scenario.
- One reason for this poor performance can be attributed to the consistently low prefetch accuracy that we have noticed. That is, the number of used prefetch blocks are far fewer compared to the prefetch requests sent to memory. A change in aggresiveness is not affecting the accuracy much, due to which, significant improvement is not achieved over a stream prefetcher.
- Another problem might be that, as proposed in the paper[1] that in case of heavy cache pollution, the prefetched blocks should not be alloted the MRU position but rather be inserted close to the LRU position. However, due to the restricted environment provided by the championship infrastructure, we did not have the facility to choose the position in cache to insert the block, and so could not implement this feature. This inference is seconded by the fact that the cache pollution metric is observed to be on a rising trend as more instructions are simulated.
- Also important to note is the fact that *dpc2sim* has three levels of cache. All interactions are done with the mid-level cache. The last level cache (LLC) is not at all touched. However, if the MSHR occupancy of L2 cache is more than half, the prefetch requests are made to LLC, which we completely ignore in our computations. Again however, the infrastructure limits us to L2 cache interactions only.
- As shown in Table III, length of the interval (number of evictions after which sampling is done) doesn't affect performance. Same has been observed when the threshold values for accuracy, lateness and pollution were changed. This might be due to the low number of instructions being simulated. Due to incompatibility between the provided shared object and PIN, we were unable to generate a bigger trace.

TABLE III
INTERVAL THRESHOLD VARIATION

Benchmark	T=1024	T=2048	T=3072	T=4096
gcc	0.291011	0.291321	0.290855	0.291287
GemsFDTD	3.430123	3.430123	3.430123	3.430123
lbm	1.075331	1.079090	1.075331	1.079090
leslie3d	1.084779	1.086480	1.085603	1.086999
libquantum	3.169687	3.169687	3.169687	3.169687
mcf	0.341864	0.341907	0.342125	0.341650
milc	0.984585	0.986086	0.984585	0.986086
omnetpp	2.201233	2.203755	2.204267	2.207636

REFERENCES

[1] Santhosh Srinath, Onur Mutlu, Hyesoon Kim, and Yale N Patt. Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers. In High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on, pages 6374. IEEE, 2007.