The FNL+MMA Instruction Cache Prefetcher

André Seznec

The FNL+MMA Instruction Cache Prefetcher*

André Seznec
Univ Rennes, Inria, CNRS, IRISA

Abstract

When designing a prefetcher, the computer architect has to define which event should trigger a prefetch action and which blocks should be prefetched. We propose to trigger prefetch requests on I-Shadow cache misses. The I-Shadow cache is a small tag-only cache that monitors only demand misses. FNL+MMA combines two prefetchers that exploit two characteristics of the I-cache usage. In many cases, the next line is used by the application in the near future. But systematic next-line prefetching leads to overfetching and cache pollution. The Footprint Next Line prefetcher, FNL, overcomes this difficulty through predicting if the next line will be used in the "not so long" future. Prefetching up to 5 next lines, FNL achieves a 16.5% speed-up on the championship public traces. If no prefetching is used, the sequence of I-cache misses is partially predictable and in advance. That is, when block B is missing, the nth next miss after the miss on block B is often on the same block $B_{(n)}$. This property holds for relatively large n up to 30. The Multiple Miss Ahead prefetcher, MMA, leverages the property. We predict the nth next miss on the I-Shadow cache and predict if it might miss the overall I-cache. A 96KB FNL+MMA achieves a 28.7% speed-up and decreases the I-cache miss rate by 91.8%.

1 The I-Shadow cache

In order to limit the number of prefetches, we propose to rely on an I-Shadow cache. The I-Shadow cache features only tags. It monitors only the demand accesses to the cache. We use an I-Shadow cache with less entries than the effective I-cache. In the submitted predictor, the I-Shadow is 3-way set-associative cache with a LRU replacement policy. It uses 15-bit partial tags. Each entry is 17-bit wide, i.e., 15 bits + 2 replacement bits.

The sequence of misses on the I-Shadow cache is strongly correlated with the hypothetical sequence of misses on a full I-cache without prefetching. Therefore, in our proposition, prefetches are triggered by I-Shadow misses.

*This work was partially supported by an Intel research grant

2 The FNL prefetcher

Next line prefetching on I-Shadow misses achieves a 9.3% speed-up on the championship public traces. However, in many cases, the next line is not used. In these cases, prefetching is a waste of resources, the average number of L2 accesses (demand misses + prefetch accesses) is increased in average by approximately 45%. Systematically prefetching several cache lines at the same time does not help for performance. In practice, the average I-cache miss rate is not improved significantly, e.g. prefetching the next 5 cache lines induces cache pollution and more accesses on the L2 cache (multiplied by 3!).

The Footprint Next Line prefetcher, FNL, addresses the wasted bandwidth and the cache pollution induced by systematic next-line prefetching.

2.1 Predicting next line use

FNL predicts if the next line will be used in a reasonable future. This prediction is used to prefetch the next line, but also the next line after the next line, up to K lines.

For FNL, we define block B+1 as being accessed in the "near future" after block B if this access is performed during the interval of L I-Shadow misses after the I-Shadow miss on block B. The FNL prefetcher monitors the "near future" use of next line using two direct-mapped tables of M entries. The *Touched* table is a 1-bit entry table that monitors if a cache block has been touched "recently" by a demand access. The *WorthPF* is a 2-bit entry table used to predict that the next line should be prefetched.

On a I-Shadow miss on block B, *Touched*[B] is set to 1 and if *Touched*[B-1] =1 then *WorthPF*[B-1] is set to 3. On each interval of L misses on the I-Shadow cache, if *Touched*[B]==1 then *WorthPF* [B] is decremented and *Touched*[B] is reset. That is *WorthPF* [B] is non-null if B was touched during one of the 3 L I-Shadow miss intervals after block B-1 was missing on the I-Shadow cache. We approximate this partial resetting on every entry with a period L of I-Shadow misses (see the prefetcher code).

On a I-Shadow miss on block B, if *WorthPF* [B] is non-null prefetch of block B+1 to the I-cache is triggered. Furthermore, if *WorthPF* [B+1] is non-null, prefetch of block

B+2 is triggered, etc. Prefetching up to 5 next blocks was found to result in the best performance trade-off for a resetting interval of L=8192 I-Shadow misses.

2.2 Filtering redundant next-line prefetches

When contiguous blocks are missing the I-Shadow cache in a limited interval, FNL partially prefetches the same blocks. One can avoid these redundant prefetches with a simple FNL filter. The FNL filter records the last few blocks that were missing on the I-Shadow cache. When block B is missing the I-Shadow cache, block B-1 is searched in the FNL filter. If block B-1 is present in the filter then only block B+5 is candidate for prefetch. In the submitted predictor, we use a 128-entry 4-way associative FNL filter with FIFO replacement.

As described in Section 5.3, this filter does not bring any performance when using the IPC simulation framework, but was introduced to avoid the need for a highly multiported prefetch queue.

2.3 Brief Evaluation of FNL

With a 192-entry I-Shadow cache, 64K entries *WorthPF* and *Touched* tables, achieves respectively 12.3% and 16.5% speed-up over no-prefetching when prefetching up to one next line and up to five next lines respectively. However prefetching up to 5 next cache lines only reduces the average miss penalty from 21.9 cycles to 20.5 cycles while only prefetching the next cache line reduces this average miss penalty to 16.0 cycles.

3 The Multiple Miss Ahead prefetcher

On a I-Shadow cache miss, FNL prefetches cache lines that are contiguous with the current cache line. FNL can not prefetch blocks are not contiguous. The MMA, Multiple Miss Ahead, prefetcher component aims at targeting this limitation of FNL.

3.1 A first step: the next predicted miss prefetcher

When no prefetch is implemented on the I-cache, the next miss is often very predictable. That is, in many cases when block B is missing, block B' is systematically the next missing block. This property holds for the I-Shadow cache that only monitors the demand accesses on the I-cache. It can be exploited in a prefetcher for the I-cache as follows. When B and B' are missing consecutively in the I-Shadow cache **and** B' is missing in the I-cache, we associate B' with Bb the fetch address of block B (i.e. the block index plus the first instruction address offset b) in a next miss prediction table. When the same association B' with Bb occurs 2 times consecutively, the entry is considered high confidence. Then on the next occurence of address Bb missing on the I-Shadow cache, B' is prefetched in the I-cache.

We implemented this Next Predicted Miss prefetcher, NPM, using a 8K-entry skewed-associative next miss prediction table. An average 12.2 % speed-up is obtained over no-prefetching on the IPC traces. Only 23.2% of the misses are eliminated, but the extra demand on the L2 cache is limited to a mere 1.0%. In practice the performance benefit is mainly brought by decreasing of the average miss penalty from 21.9 cycles to only 10.5 cycles.

3.2 Multiple miss ahead prefetcher

With the NPM prefetcher, many prefetches are triggered too late. In order to limit/avoid the occurrences of these late prefetches, a prefetcher should trigger the prefetch in advance in order that the prefetched block is present in time in the cache.

Instead of predicting the next block after an I-Shadow miss on block B as in NPM, the Multiple Miss Ahead prefetcher, MMA predicts the *n*th block $B_{(n)}$ that will be missing in the I-Shadow cache **and** misses the I-cache. We refer to *n* as the ahead distance of the MMA prefetcher.

This Multiple Miss Ahead prefetcher, MMA, allows to trigger the prefetch of blocks in advance of their effective demand use. In practice, the ahead distance should be chosen to allow to cover the access latency of the second level cache. On the championship framework, ahead distances of 6 to 15 lead to the best performances. However ahead distances in the 30-40 range do not lead to significant performance difference.

3.3 Combining MMA and FNL

MMA (or NMP) and FNL can be easily combined since the misses they target are slightly different. In that case, the next lines following the block prefetched by MMA are also prefetch candidates by FNL.

3.4 MMA Prefetch filtering

In order to limit the number of issued prefetch requests, requests can be filtered. In our implementation, MMA does not issue a prefetch on block B and its successors if a prefetch for the same block has been issued recently. The MMA filter records the last 16 MMA triggered prefetch addresses.

4 Performance evaluation

The submitted predictor will be referred to as FNL5+MMA9, i.e., maximum 5 next lines and ahead distance 9. The submitted configuration features the following major components, a 192 entry I-Shadow cache, 64K-entry **Touched** and **WorthPF** tables, a 8K-entry miss ahead prediction table, and the two prefetch filters, the 16-entry MMA filter and a 128-entry FNL filter.

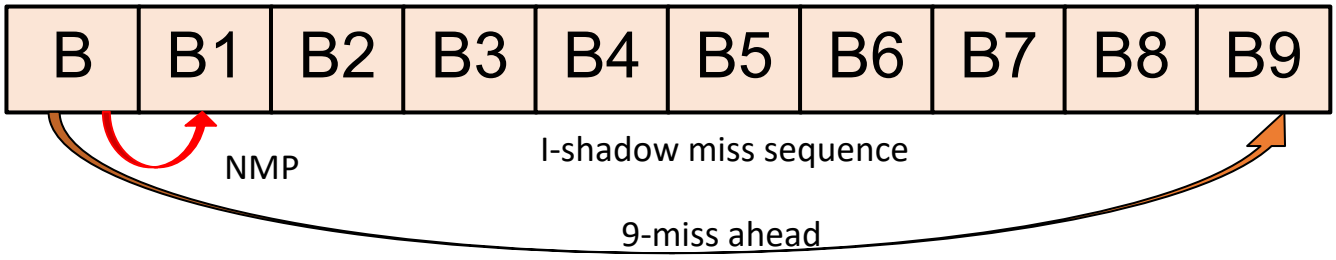

Figure 1. MMA vs NPM prefetcher

The predicted addresses are block addresses, i.e. 58-bits wide. An entry in the miss prediction table consists of a partial tag, 12 bits, the target block 58 bits, and a 1-bit tag to manage both confidence and replacement for a total of 71 bits. The storage budget of the overall submitted predictor is large close to 96 Kbytes, but easily fits in the allowed 128Kbytes budget: 408 bytes for the I-Shadow cache (192 17-bit entries)), 8Kbytes for the *Touched* table (64K 1-bit entries), 16Kbytes for the *WorthPF* table (64K 2-bit entries), 71Kbytes for the miss ahead prediction table (8K 71-bit entries), 116 bytes for the MMA filter (16 58-bit addresses) and 136 bytes for the FNL filter (128 17-bit entries).

We evaluated the different prefetchers described in the previous sections with the IPC framework. Table 1 reports simulation results for the submitted predictor FNL5+MMA9, FNL5, MMA9, NMP, FNL5+NMP, FNL5+MMA30, and FNL3+MMA9 all assuming the same table sizes as in FNL5+MMA9. Reported statistics are the average speed-up (geometric mean), the average reduction of I-cache miss (arithmetic mean), the average I-cache miss latency (arithmetic mean) and the average increase of number of requests on the L2 cache (arithmetic mean), compared with the I-cache without prefetcher.

As already mentioned, NMP is very effective at predicting the next miss when it predicts. The extra accesses on the L2 cache are very limited (1.0%). But when an I-Shadow cache miss is also an I-cache miss, the prefetch of at most one extra cache block is triggerred during the initial I-cache service. Therefore the prefetch is often late, but reduces the average miss penalty.

FNL5 prefetches up to 5 blocks after an I-Shadow miss. It is quite efficient at reducing the I-cache miss rate without prefetching too many useless cache blocks (16.5% speed-up for only 11.8% extra accesses on the L2 cache).

Combining FNL5 and NMP is efficient resulting in 22.6% speed-up while the increase of the demand on the L2 cache is still limited to 23.1%. But still some prefetches issued by NMP are late prefetches.

MMA9 achieves performance in the same range (21.1%) as FNL5+NMP without even leveraging next-line prefetching. MMA9 also generates a limited increase of the demand on the L2 cache (3.0%) indicating that the ahead prediction of misses is very precise.

Finally, our submission, FNL5+MMA9, achieves 28.7% speed-up with a larger, but still reasonable, 38.3% extra demand on the L2 cache. The average misprediction rate is reduced by 91.8%.

FNL5+MMA30 illustrates that prefetch can be triggered very early in advance and still be quite precise since the extra demand on the L2 cache is limited to 48.5%. Furthermore the results for FNL3+MMA9 illustrates that one can limit the maximum number of prefetched next lines to 3 without losing any significant performance.

We also simulated larger and smaller prefetchers to assess the scalability of our proposal. Quadrupling the size of the predictor structures only leads to an extra 0.1% performance improvement. On the other hand, half size, one fourth size and one eight size FNL5+MMA9 result respectively in 28.4%, 27.3% and 24.1% performance speed-up. Moreover, we did not try to limit the storage budget in MMA. A simple optimization to limit the budget would have been to specialize several ways of the MMA table to prefetch targets sharing their most significant bits with their triggering addresses.

5 Effective hardware considerations

5.1 Timing considerations

The computation of prefetch addresses is not a critical path on FNL+MMA. For FNL, the I-Shadow cache has a shorter response time than the I-cache (less associativity, narrow tags). For MMA, the response time of the large 9-miss ahead table might be longer than the one of the I-cache, but MMA prefetches can be delayed by a few cycles without any performance loss.

5.2 Insertion in an out-of-order execution environment

The FNL+MMA has been tested with the IPC framework. FNL+MMA exploits the I-Shadow cache which simulates the contents on the demand misses. The IPC framework is a trace-based simulator that does not simulate wrong path. In a real hardware implementation, wrong path execution is likely to alterate the regularity/predictability of the miss sequences on the I-cache and the I-Shadow cache. Therefore the I-Shadow cache should be updated at commit time and not at fetch time as currently done in the simulator code. FNL+MMA is resilient to large ahead distances in the range of 30 or more. This indicates that one could trigger the MMA prefetcher at commit time. This would avoid polluting the cache with prefetches triggered on the wrong

	FNL5+MMA9	FNL5	MMA9	NMP	FNL5+NMP	FNL5+MMA30	FNL3+MMA9
Speed-up	**1.287**	1.165	1.211	1.122	1.226	1.284	1.285
Miss reduction (%)	**91.8**	46.0	72.5	23.2	55.0	92.2	90.7
Extra L2 accesses (%)	**38.3**	11.8	3.0	1.0	23.1	48.5	32.7
Miss latency (cycles)	**20.9**	20.5	22.2	10.5	15.8	21.7	20.5

Table 1. Performance statistics for various prefetcher configurations

path while prefetches would still be initiated timely on the correct path.

5.3 The FNL and MMA filters

The FNL and MMA filters were introduced in this study. In simulations using the IPC framework, they do not bring any extra performance. However, in a realistic hardware implementation they would be needed.

On the FNL5+MMA9 predictor, up to 11 candidates for prefetch could be triggered by a single I-Shadow miss on block B. The 11 candidates are the group of the 5 next lines after block B and the MMA prefetch candidate if any, and its 5 potentially associated next lines. The IPC framework allow all these requests to check the prefetch queue for their presence through an associative search. On the same simulated cycle, redundant prefetches are dropped while new requests are added to the prefetch queue. In an effective design, this can not be realistically implemented, and would necessitate further buffering before the prefetch queue, potentially delaying the prefetches.

The FNL filter and the MMA filter have been introduced to pre-filter the requests before check/insertion in the prefetch queue. The FNL filter is checked at most one time for each of the two possible groups of prefetchs. The MMA filter is checked once for the MMA group. The filters considerably reduce the pressure on the prefetch queue. If one does not use these filters then the average number of prefetch requests treated by the prefetch queue is more than doubled.

5.4 Dealing with physical address space

One limitation of the IPC framework is the use of a virtual address virtual tag I-cache assisted by a "magic" virtual-to-physical address translation. In practice, FNL+MMA can operate on the physical addresses without requiring any virtual-to-physical address translation. However FNL does not work when crossing page boundaries; this limitation decreases the observed speed-up from 28.7 % to 28.4 %. Moreover in many real implementations, physical addresses are narrower than virtual addresses, e.g. 48 bits.

6 Related work

Next line prefetching for I-caches can be traced back to the 70's [5]. To the best of our knowledge, the technique we propose to filter the instruction footprint to avoid overprefetching is original. The predictability of the sequence of misses on instruction caches and its exploitation for prefetching has been has been explored in [2, 1]. For instance in [1], Feldman et al. point out that miss sequences are amenable to be quite predictable if one can filter the noise induced by control flow hazards and all incidents in execution. With the I-Shadow cache, we go a step further by eliminating the perturbations introduced by the prefetcher itself. Using multiple miss ahead prediction was inspired by work on branch prediction and instruction fetch front-end [4, 3]. Multiple miss ahead prediction allows timely prefetching.

7 Conclusion

In many cases, the instruction stream presents spatial locality. The next-line prefetcher exploits this locality. However next-line prefetching and particularly multiple next-line prefetching indiscriminately prefetches many blocks that will not be used in the future. Our Footprint Next Line prefetcher efficiently filters these useless prefetches and therefore leads to higher performance than conventional next-line prefetching.

In many cases the next missing instruction block in the overall I-cache can be predicted from the simulation of a smaller I-cache supporting only the demand accesses. Unfortunately, predicting the very next instruction block results often in a late prefetch, i.e., decreases the miss latency but does not eliminate the cache miss. Inspired by previous work on branch prediction [4, 3], the Multiple Miss Ahead prefetcher predicts the potential misses several blocks in advance, and eliminates many of these late prefetches.

On the IPC traces, the combined FNL+MMA prefetcher reduces the average miss rate by 91.8 %. It achieves a 28.7% average speed-up at a limited 38.3% increase of the instruction induced L2 accesses.

References

[1] Michael Ferdman, Cansu Kaynak, and Babak Falsafi. Proactive instruction fetch. In Carlo Galuzzi, Luigi Carro, Andreas Moshovos, and Milos Prvulovic, editors, *44rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2011, Porto Alegre, Brazil, December 3-7, 2011*, pages 152–162. ACM, 2011.

[2] Michael Ferdman, Thomas F. Wenisch, Anastasia Ailamaki, Babak Falsafi, and Andreas Moshovos. Temporal instruction fetch streaming. In *41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-41 2008), November 8-12, 2008, Lake Como, Italy*, pages 1–10. IEEE Computer Society, 2008.

[3] A. Seznec and A. Fraboulet. Effective ahead pipelining of the instruction addres generator. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[4] André Seznec, Stéphan Jourdan, Pascal Sainrat, and Pierre Michaud. Multiple-block ahead branch predictors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 116–127, 1996.

[5] Alan Jay Smith. Sequential program prefetching in memory hierarchies. *IEEE Computer*, 11(12):7–21, 1978.