Implementation of an aggressive offset prefetcher using a sandbox

Kaan ICER

***Abstract*— In this project, an aggressive offset prefetcher was implemented by using a Sandbox as a Bloom Filter. Algorithm was applied with different parameters such as varying limited prefetch count during each access and different renewing intervals of the algorithm.**

I. Introduction

Prefetching brings data closer to the processor before it is requested.

Cache prefetching is a technique used by processors to improve execution performance by fetching instructions or data from their original storage in slower memory to a faster local memory before it is needed.[1] Cache prefetching guesses which blocks will be accessed soon and brings them into cache ahead of time. The important thing at this point is the prefetching distance. If the prefetching distance is much larger than the optimal value, it means that the prefetching was done too early. And if the prefetching distance is much smaller than the optimal value, it means that the prefetching was done too late. The main purpose of this project is to set this prefetch distance and its aggressiveness.

It is decided whether the algorithm (an aggressive offset prefetcher by using a Sandbox) works well or not by measuring and averaging the Instruction per Cycle values. The average number of instructions executed for each clock cycle is called instructions per cycle (IPC).[2] 46 different benchmarks were used to measure it. Then the best parameters for the highest IPC was found through several experiments.

II. Procedure for Implementing Algorithm

A. Definition of Sandbox Prefetching

There are two general types of prefetchers: Confirmation-based Prefetchers and Immediate Prefetchers.

F. A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305 USA (corresponding author to provide phone: 303-555-5555; fax: 303-555-5555; e-mail: author@ boulder.nist.gov).

Confirmation-based prefetchers use confidence mechanism. For instance, when address A is seen in cache line, confirmation-based prefetcher waits for the future access and no prefetches are performed. When A+1 is seen, still no prefetches are performed. When A+2 is seen after A+1 was seen, then mechanism will confirm that the prefetching distance 1 is appropriate for the stream. So, A+3 will be prefetched when there is an access to A+2.[3]

Immediate Prefetchers always issue prefetches. Next line prefetching is a popular example of this type prefetchers. When the A is seen, A+1 is prefetched. And when A+1 is seen, A+2 is prefetched, and so on. There is no confidence mechanism in Immediate Prefetchers.[4]

Sandbox Prefetcher combines the ideas of Confirmation-based Prefetchers and Immediate Prefetchers. This mechanism starts to prefetch after a sandbox evaluates prefetcher's prediction success. If the mechanism is successful (based on a threshold), prefetching is made according to the success scores.[5]

B. Algorithm

In each L1 access, we generate a prefetch address. When A is current access, then A+distance is the generated prefetch address. Then we use bloom filter (Sandbox) to map the generated prefetch address. At this step, we don't prefetch any data. Then we check whether A, (A-distance), (A-2*distance) and (A-3*distance) exists in the sandbox or not. Maximum score for each access is 4, and minimum score is 0.

Sandbox Prefetching consists of 16 candidate prefetchers. Initially this set of prefetchers is for offsets (by offset, prefetching distance is referred) -8 to -1, and +1 to +8. At the beginning of each evaluation period, the sandbox, the L1 access counter, and the prefetch accuracy score are all reset.

"After this, the candidate prefetcher generates a prefetch address, based on the reference cache line address and its own prefetch offset, and adds this address to the sandbox. Finally, the counter that tracks the number of L1 accesses this period is incremented. Once this number reaches 256, the evaluation period is over and the sandbox and other counters are reset, and the evaluation of the next candidate prefetcher begins. After a complete round of evaluating every candidate prefetcher is over, the bottom 4 prefetchers with the lowest prefetch accuracy score are cycled out, and 4 more offset prefetchers that have not been recently evaluated from the

range -16 to +16 are cycled in."[6] Figure 1 is visual representation of this process. This process continues until 15, 16, -15, -16 are cycled in.

-8	265
-7	103
-6	296
-5	312
-4	408
-3	507
-2	616
-1	794
1	742
2	556
3	580
4	209
5	284
6	246
7	92
8	193

-8	265
-9	176
-6	296
-5	312
-4	408
-3	507
-2	616
-1	794
1	742
2	556
3	580
-10	87
5	284
6	246
9	284
10	142

Fig. 1.

When it is time to issue real prefetches; if the score is greater than 768, then 3 prefetches will be done. If the score is greater than 512, then two prefetches will be done and if the score is greater than 256, then only a single prefetch will be done.[7]

C. *Sandbox*

A Bloom filter is a probabilistic data structure, generated by Burton Howard Bloom in 1970, which is used to test whether an element is a member of a set or not.[8] We chose the size of the Bloom filter (Sandbox) to 2048 bits in our algorithm. I used 3 different hash functions to map elements into the filter. In Fig. 2., it is explained how the position indexes of generated prefetch addresses are created. Last 11 bits of generated prefetch address is directly used. Last 11 bits and previous 11 bits are XORed. And last 11 bits and the 5 bits that are shown in the Fig. 2. are XORed as well.

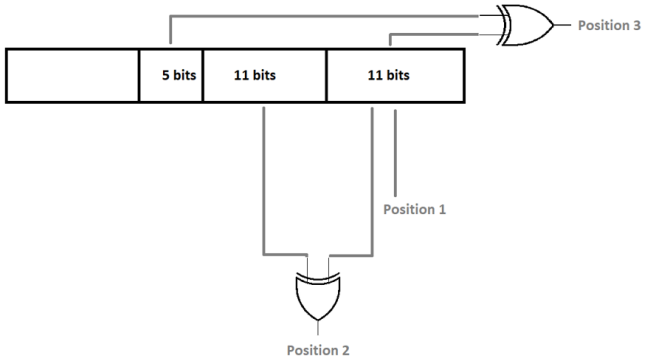

Fig. 2.

There is a small probability of false positives in Sandbox.

D. *Deciding to combine Next Line Prefetching and Sandbox Prefetching, and adjusting other parameters*

Sandbox Prefetching with unlimited prefetch count during each L1 access was implemented and results of some benchmarks can be seen in the third column in Fig. 3. 1st column shows the case without prefetching, 2nd column shows the case with next line prefetching. For some benchmarks, there is no prefetching when Sandbox was implemented. Because the results are sometimes same with the case when there is no prefetch. So, next line prefetching can be implemented when the score is not enough to exceed threshold.

	Standard	Standard	Standard	Standard
1	Benchmark	no-no-no	next_line-no-no	problem3-no-no
2	600.perlbench_s-570B.champsimtrace.xz	0.553508	0.561519	0.561519
3	602.gcc_s-1850B.champsimtrace.xz	0.28345	0.388587	0.423258
4	602.gcc_s-2226B.champsimtrace.xz	0.12241	0.22334	0.408913
5	602.gcc_s-734B.champsimtrace.xz	0.355956	0.420359	0.425926
6	603.bwaves_s-1740B.champsimtrace.xz	0.714864	0.900471	0.714864
7	603.bwaves_s-2609B.champsimtrace.xz	0.716257	0.902883	0.716257
8	603.bwaves_s-2931B.champsimtrace.xz	0.624769	1.01632	1.01632
9	603.bwaves_s-891B.champsimtrace.xz	0.444119	0.863406	1.31859
10	605.mcf_s-1152B.champsimtrace.xz	0.259041	0.275493	0.275493
11	605.mcf_s-1536B.champsimtrace.xz	0.15837	0.221548	0.15837
12	605.mcf_s-1554B.champsimtrace.xz	0.159256	0.147412	0.159256
13	605.mcf_s-1644B.champsimtrace.xz	0.124216	0.139494	0.149635
14	605.mcf_s-472B.champsimtrace.xz	0.281254	0.29335	0.281254
15	605.mcf_s-484B.champsimtrace.xz	0.335199	0.364977	0.335199
16	605.mcf_s-665B.champsimtrace.xz	0.318291	0.33195	0.318291
17	605.mcf_s-782B.champsimtrace.xz	0.151397	0.168241	0.157262
18	605.mcf_s-994B.champsimtrace.xz	0.276399	0.276237	0.276399
19	607.cactuBSSN_s-2421B.champsimtrace.xz	0.740887	0.855438	0.854677
20	607.cactuBSSN_s-3477B.champsimtrace.xz	0.846247	0.897946	0.879723
21	607.cactuBSSN_s-4004B.champsimtrace.xz	0.835937	0.884566	0.858464

Fig. 3.

When the Sandbox Prefetching is combined with Next Line prefetching, the average IPC is higher than both next line prefetching and Sandbox prefetching. It can be seen in Fig. 4.

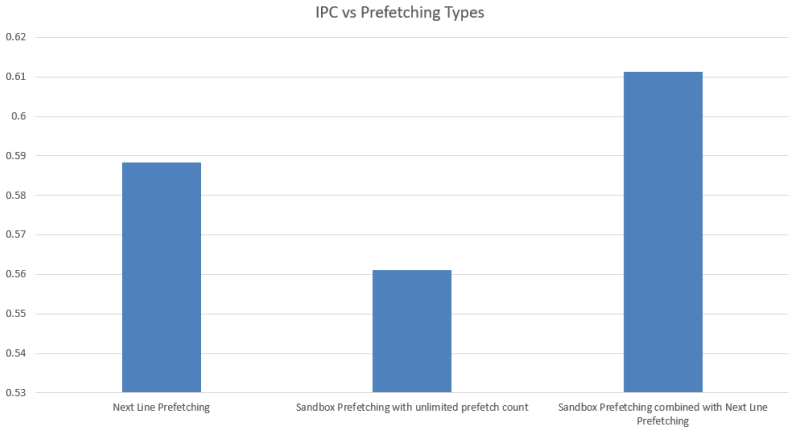

Fig. 4.

When I decided to use limited prefetch count, IPC value increased. It can be seen in Fig. 5.

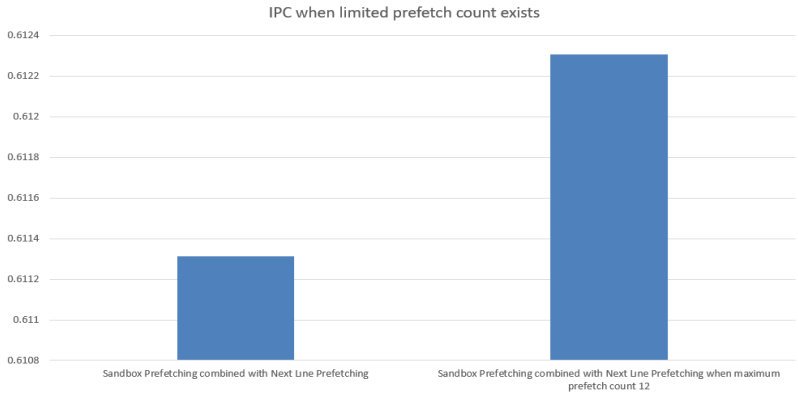

Fig. 5.

When limited prefetch count exists, priority belongs to minimum absolute valued offsets (distances). In Fig. 6., different number of limited prefetch counts are implemented.

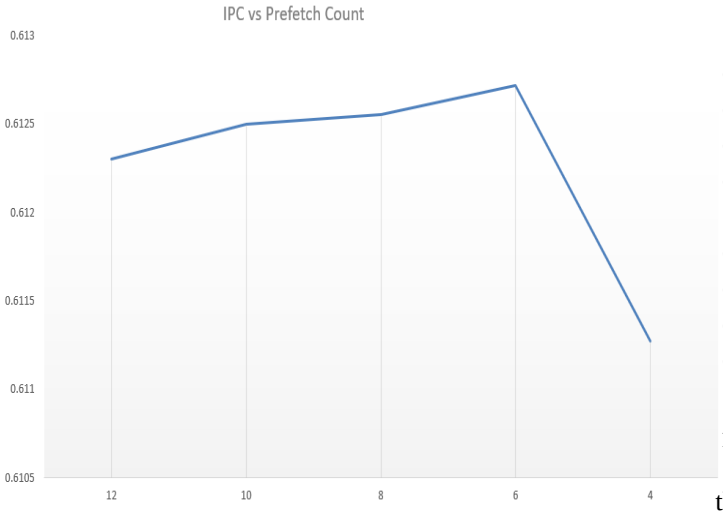

Fig. 6.

When I searched the scores between the offsets -8 and +8, the IPC increases little bit. So, it is good to limit the Number of Evaluated Prefetch Offsets interval. In Fig. 7., the limited prefetch count is 8 and the algorithm is renewed during each 25 million L1 accesses. So, for rest of the implementations, it is decided to use prefetch offsets between -8 and +8. In Fig. 8., it is confirmed again that the prefetch count 6 is better than 8.

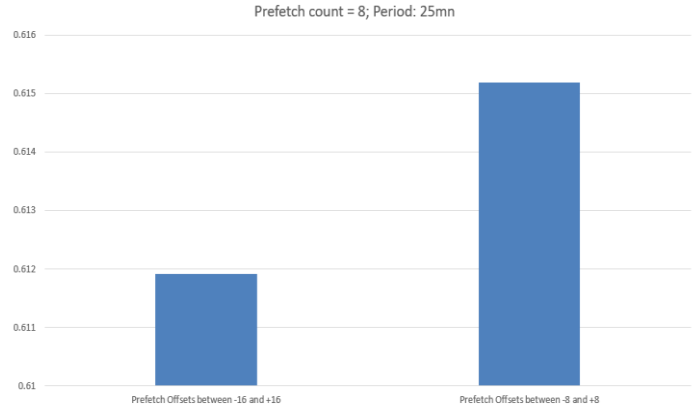

Fig. 7.

Fig. 8.

In Sandbox Prefetching, the score searching is done during each 256 accesses. Varying this search interval helps to increase IPC. When the score is evaluated for each 384 L1 accesses, the IPC value increases as seen in the Fig. 9.

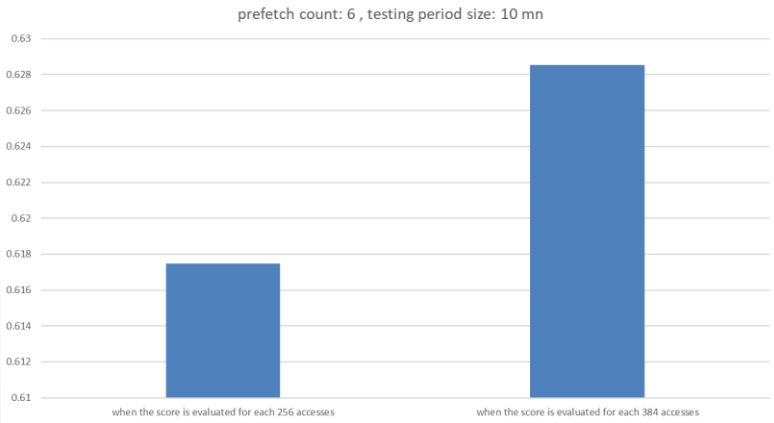

Fig. 9.

Varying the renewal interval of the algorithm also effects the IPC as seen in the Fig. 10. Renewing the algorithm for each 15 million L1 accesses gives the best IPC.

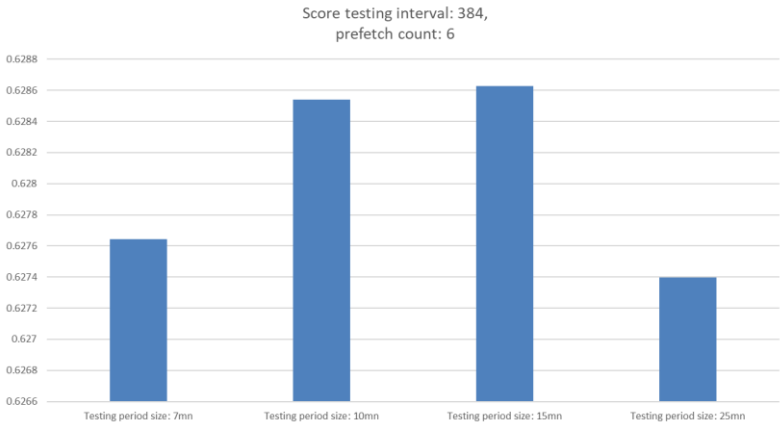

Fig. 10.

In Fig. 11., when the score is evaluated for each 512 L1 accesses, the IPC value is the highest.

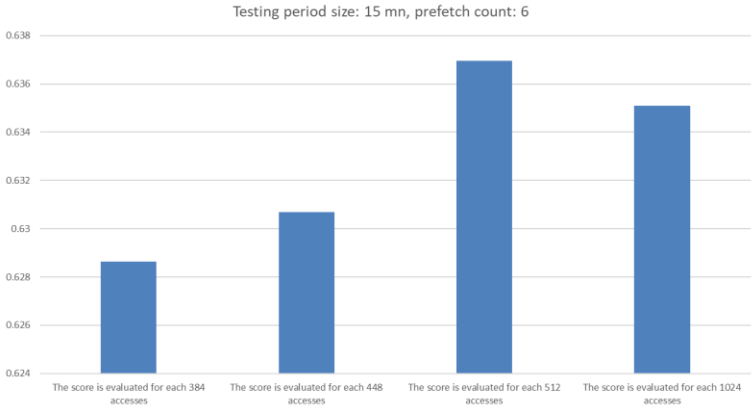

Fig. 11.

III. Conclusion

The advantage of this algorithm is that algorithm was run during only small portion of L1 access. Also 2048 bits were used to store the addresses. So, I could get 0,637 IPC.

References

[1] A. J. Smith, “Cache Memories,” ACM Computing Surveys, vol. 14, no. 3, pp. 473–530, 1982.

[2] J. L. HENNESSY and D. A. PATTERSON, Computer Architecture: A quantitative approach. Amsterdam, Boston, Heidelberg etc.: Elsevier, 2007.

[3] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-fei Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, “Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers,” 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 2–3, 2014.

[4] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-fei Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, “Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers,” 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 2–3, 2014.

[5] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-fei Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, “Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers,” 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 2–3, 2014.

[6] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-fei Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, “Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers,” 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 5-6, 2014.

[7] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-fei Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, “Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers,” 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 5-6, 2014.

[8] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.