ECE 511 Assignment 3.1

Andrew Smith

February 28^{th}, 2019

1 Introduction

The goal of Assignment 3 Checkpoint 1 was to model a Markov Prefetcher and analyze its performance impact on the test system. Prefetching is an intuitive way to reduce the number of stalls due to cache misses at the cost of increasing the total systems bandwidth requirement.

2 Implementation

The Markov Prefetcher is a derivative of a queued prefetcher. It has a Markov Predictor that feeds predictions into a prefetch queue, and these predictions get get loaded into the L2 cache. The Markov Prefetcher becomes more accurate over time as it begins to learn the memory access pattern of the program. The prefetcher stores the previous request's address and trains based on the previous missed address. It records the number of times sequential pairs of misses occur to automatically prefetch a cache line if an address causes a cache miss. The miss address is used to look up an entry in the table, and then the top n most popular next miss addresses are selected from the predictors and submitted to the prediction queue. If the address is not found in the table it is added based on an LRU policy.

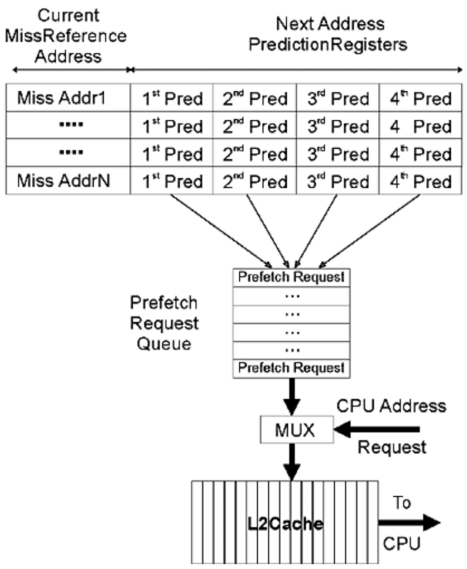

Figure 1: Markov Prefetcher[1]

3 Results

To evaluate the prefetchers I simulated the prefetchers in the default GEM5 L2 Cache. There are a few key metrics that are important to consider when evaluating the effectiveness of a prefetcher. The instructions per

cycle of the system, the L2 miss rate, and the memory bandwidth of the system. To highlight the differences in the prefetchers, four benchmarks were performed: Blackscholes, Bodytrack, Fluidanimate, and x264.

3.1 Prefetcher Comparisons

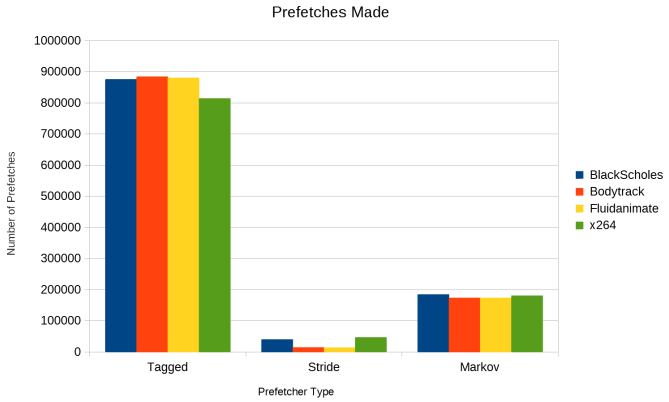

Figure 2: Number of Prefetches Made

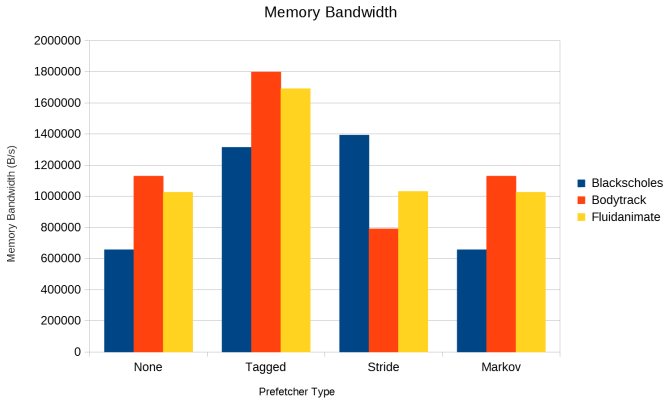

Figure 3: Memory Bandwidth

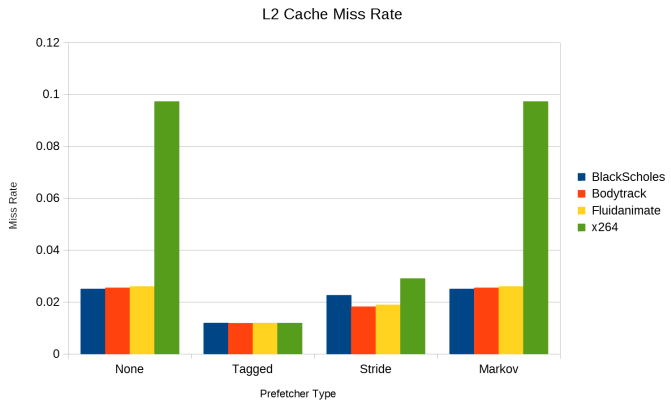

Figure 4: L2 Miss Rate

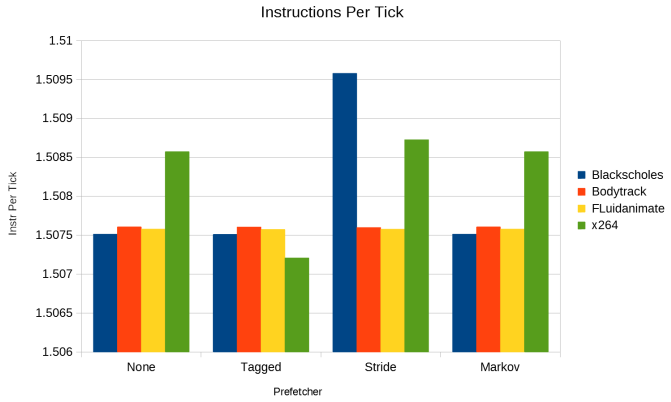

Figure 5: Instructions Per Cycle

Of all the prefetchers tested there is no clear prefetcher that is better than another. The tagged prefetcher has the simplest concept and the simplest hardware, but it also issues 5 to 15 times the prefetches of the other prefetchers (Figure 2). This leads to 1.5x the bandwidth of a system without a prefetcher (Figure 3). The tagged prefetcher had the lowest L2 Cache Miss Rate of all three prefetchers (Figure 4). The Markov Prefetcher issued fewer prefetches when compared to the Tagged prefetcher making it potentially well suited for lower memory bandwidth applications. Stride prefetching had the second lowest miss rate however it lead to a terrible IPC in the Blackscholes benchmark (Figure 5). Somehow the data for the Markov Prefetcher matches the data for the baseline system exactly even though when I was debugging it was working properly. I believe some debugging is required for reporting stats correctly.

3.2 Fine Tuning the Markov Prefetcher

In order to evaluate the Markov Prefetcher, I tuned some of the parameters to see what affect they would have on the system performance. I varied the size of the table, the number of predictors per entry and the training threshold to see where the baseline Markov Prefetcher can be improved.

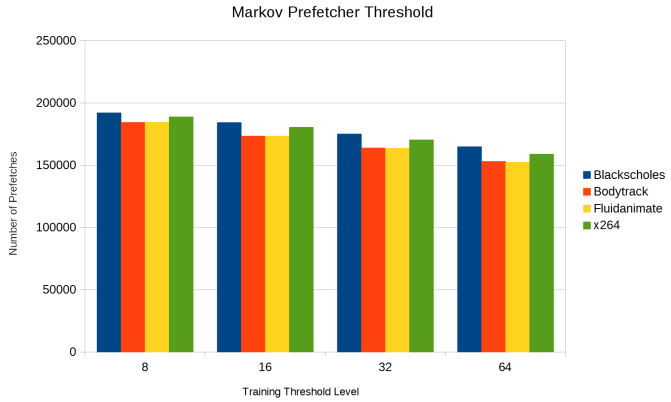

Figure 6: Predictions Issued for different training thresholds

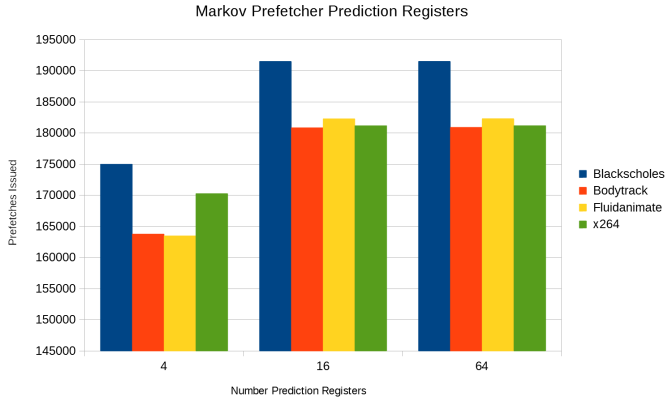

Figure 7: Prefetch Issues for varying number of predictors per table entry

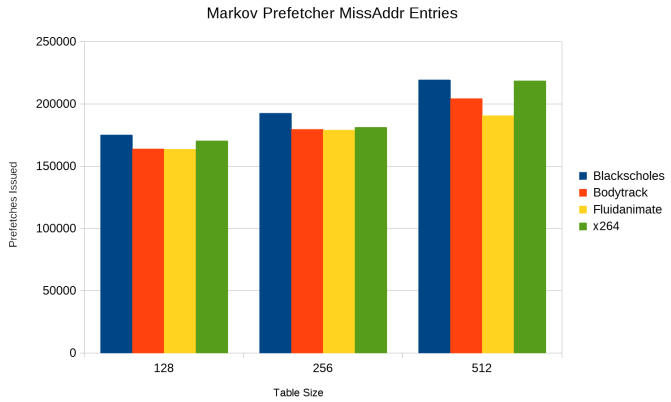

Figure 8: Prefetch Issues for varying Table Size

Increasing the number of predictors and the number of table entries for missed addresses lead top more predictions being issues, most likely due to more reuse from addresses a;ready in the table. If there is a lot of replacement happening in the table then retraining must occur which will prevent predictions being

made for that entry until the training threshold is reached (Figure 7). Increasing the predictors per entry was only improved marginally after 16 Prediction Registers per entry. This is due to only two being selected each prefetch so increasing registers without the number of prefetches issued has diminishing returns (Figure 8). As expected I found that the larger the training threshold the less predictions issued (Figure 6). This is simply because the prefetcher spent more time training before it could issue prefetches.

4 Conclusion

Of all the prefetchers examined, Tagged, Stride, and Markov, each has their own pros and cons in the real world. The tagged prefetchers naive approach would not be useful on low memory bandwidth systems however with the emergence of high bandwidth HBM2 and DDR5 systems in the near future, it might not be a bad strategy. The Markov prefetcher is the most clever of the three prefetchers however it is expensive to implement because it would require a lot of area for the table that holds the prediction information. It could also potentially have a slow critical path for all the compare logic in the fully associative Miss Address registers.

5 Resources

[1] D. Joseph and D. Grunwald, "Prefetching using Markov predictors," IEEE Transactions on Computers, vol. 48, no. 2, pp. 121–133, 1999.