Final Project Report

By Surya Suresh and Trevor Weger

This project focused on implementing the spatial memory streaming technique from the paper "Spatial Memory Streaming" by Somogyi et. al. The main idea of this paper is that programs exhibit spatial variation in memory access patterns that traditional caches are not able to take advantage of. This is because caches with small fixed size blocks are unable to fit the data spanning the spatial region. Increasing the block size will not solve the problem either since only a handful of memory addresses are accessed from the large spatial region. The authors therefore present the idea of spatial memory streaming to exploit spatial variation in programs.

The idea is to first split the total memory address space into distinct spatial regions. Once this is done, two structures called the Active Generation Table (AGT) and Pattern History Table (PHT) are needed to generate and store spatial patterns occurring throughout the region. The AGT stores the trigger access's spatial tag (higher order bits of a load instruction's memory address) and maps this tag to a bitmap, which is an array of addresses that hit the cache and belong to the respective spatial region. The PHT stores the trigger access's program counter (PC) and offset (lower order bits of a load instruction's memory address) and maps this to its bitmap.

Each load instruction in the program is analyzed to see whether or not its corresponding memory address hits or misses the cache. In the event that the address hit the cache, this indicates that a spatial region entry has been recorded in the AGT and thus that entry's bitmap gets updated accordingly. In the event of a miss, there are two cases that could happen depending on whether the miss was a compulsory versus conflict. In a compulsory miss, a new AGT entry is created with the missed address as its trigger access; the bitmap is also updated to set the missed address. In addition, a lookup is done in the PHT to determine if a pattern has been previously recorded using the missed address's PC and offset. If such an entry exists, then that entry's bitmap is extracted and iterated upon to prefetch the addresses into the data cache. Additionally, every prefetch done into the dcache has the potential to evict lines as well. These evicted lines will cause the pattern corresponding to its spatial region to end, removing the entry from the AGT. If the pattern is large enough with more than 1 entry in the bitmap, the pattern gets stored into the PHT. In the event of a conflict miss, the process is similar to what was done for a compulsory miss with the addition of a removal of the victim's AGT entry, which gets added into the PHT if the pattern in its bitmap is greater than 1. With the technique described, the next paragraphs will discuss the implementation into a simulation program called Scarab.

In scarab, all the relevant code was done in the dcache_stage.c file. Global variables such as the AGT, PHT, spatial region size, cache entry number, and minimum pattern size were created. The AGT was implemented as an array of AGT structs where each struct contained the trigger access's PC, offset, spatial tag, and bitmap. Similarly the PHT was implemented as an array of AGT structs for all patterns recorded. The other parameters were created as variables for configurability. Initially the AGT and PHT were planned to be represented with hash tables in order to access entries in constant time; however, the library that scarab used, hashlib, did not properly implement functionality for storing complex data.

As each address is processed in scarab, a cache access is performed to check whether the memory address hit or missed in the cache. On a hit, the AGT array is looped through and once the entry is found, the bitmap, which is itself an array, gets updated. On a miss, an AGT entry is potentially removed to be placed in the PHT and a lookup into the PHT is done to extract the bitmap corresponding to the missed address. To perform the prefetching, a new memory request is issued with a callback function to fetch the data from the bitmap into the data cache as well as writing back any data that was evicted to memory if it was dirty. The end result was functioning runs that tracked spatial patterns and improved miss ratios through prefetching addresses from the patterns.

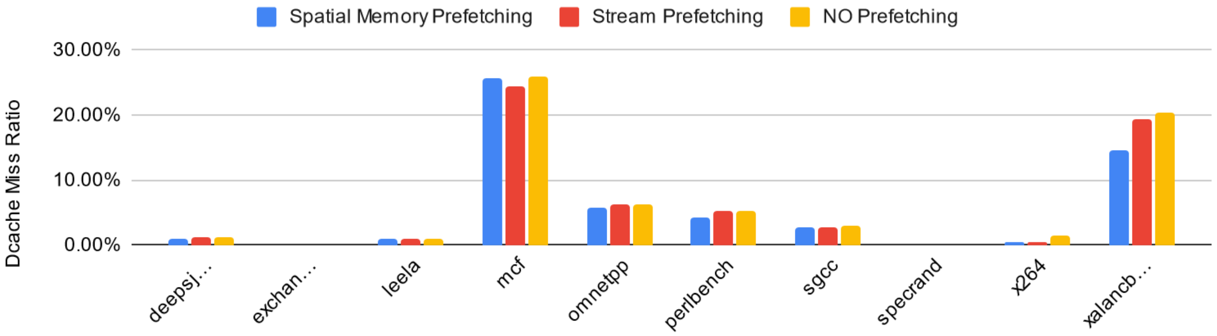


Table 1: Prefetch Comparison

To test our prefetcher, we analyzed the results of our prefetch implementation on three different configurations with Dcache miss ratio as the metric for success. The first configuration is used to compare our spatial memory prefetcher to the built in Stream prefetcher in scarab using no prefetcher as a baseline. The results given by Table 1 show that all benchmarks except MCF showed the spatial prefetcher decreased the miss ratio for the dcache compared to the Stream Prefetcher. Based on the paper we hypothesize that the MCF benchmark must have low spatial locality. Similarly benchmark xalancbmk seems to exhibit high rates of spatial locality as prefetching patterns greatly reduced the overall miss rate.

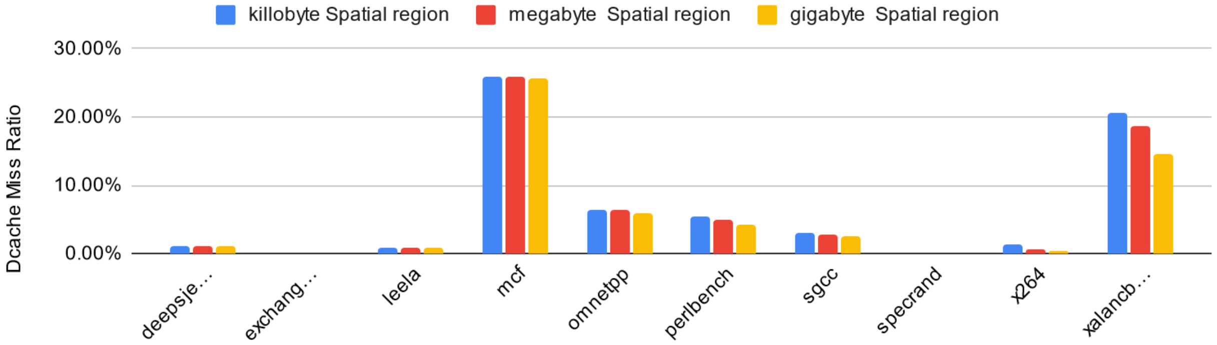
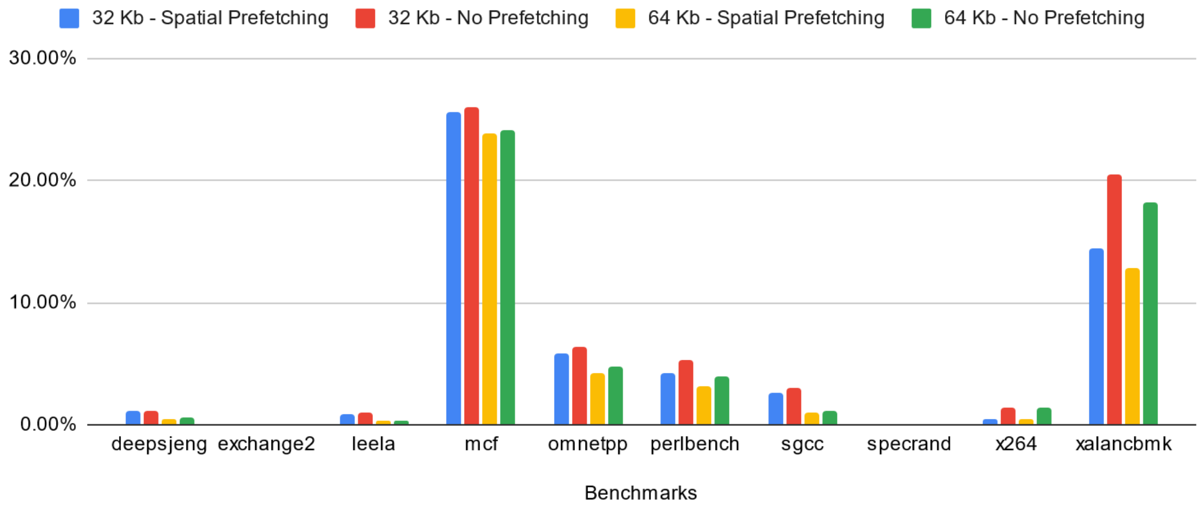


Table 2: Spatial Region Size

The next test we ran evaluated our prefetcher on different spatial region sizes. We tested the ten benchmarks against spatial region sizes of 1 KB, 1 MB, and 1 GB. From the results given by Table 2, we observe that as the spatial region sizes increased in size, the miss ratios of all the programs decreased. We believe this is due to the programs already having high spatial locality. To take advantage of the access patterns, the spatial region size needed to be set to a bigger size. We also observe that some of the programs' miss ratios were improved much more than other programs. For instance xalancbmk’s miss ratios improved a higher percentage compared to the mcf.

Table 3: Dcache Size Variation



The final test we ran was based on varying the Dcache size and comparing results with the baseline being a no prefetch run. Table 3 demonstrates our results and shows for both sized Dcaches the spatial prefetch provides consistent improvements over the baseline. An interesting pattern we noticed was that the spatial region prefetcher reduced the miss rate by the same ratio compared to the baseline between the two Dcache sizes. This suggests to us that the misses covered by spatial prefetching are mostly independent of the dcache size and more dependent on what type of misses are occuring. In conclusion spatial prefetching is beneficial but greatly dependent on an application's spatial locality.