HW 3 – Cache Optimization

沈昱宏, 110705013

***Abstract*—In this homework assignment, I analyze the cache behavior of the pi program using 5000 digits. I fixed the data cache size to 4KB and tried different implementations to improve the cache performance on the pi program. I tried two ways to improve the cache performance. Firstly, I modified the cache ways to try to improve the performance. Secondly, I tried changing the FIFO cache into LRU cache. The result shows that the LRU cache did improve the cache performance.**

Keywords—Aquila, Pi, Cache

I. THE PI PROGRAM

To design a cache that is suitable for the PI program, it is crucial to know how the PI program runs. There are 5 main variables in the program, and all of them are arrays of integers. All the integer arrays are accessed sequentially, some of them will only be accessed one by one, some of them will be used with another integer array.

II. MEASURING D$ PERFORMANCE

This experiment is done on ARTY board using the original implementation and NDIGITS = 5000 in the PI program. I counted the cache hit / miss rate by counting the total number of cache hits for r/w and the total number of p_strobe_i for r/w. Additionally, I computed the total number of cycles between p_strobe_i and p_ready_o to find the cache latencies of r/w. Please note that I do not count the latency of the cache when the request is hit. To separate the hit cycles, I use the following formulat to compute the cache latency (#cycle_for_read – #hit_read) / (#read_requests - #hit_read). The result shows that both reading and writing have a good hit rate and the latency of reading & writing on cache miss is approximately the same.

TABLE I. CACHE STATISTICS (FIFO)

Category	r/w	Percentage / #Cycle
Cache Hit Rate	read	77.5 %
	write	83.3 %
Average Cache Latency	read	55.7 cycles
	write	55.6 cycles

III. N-WAY ASSOCIATIVE CACHES

I changed the number of cache ways and observed the performance of different cache ways under N_DIGITS = 5000. The result shows that the cache performance of different cache ways is quite similar.

TABLE II. MSEC FOR EACH CACHE WAYS (NDIGITS = 5000)

N_WAYS	Execution time (msec)
2	30522
4	30522
8	30516

Additionlly, I tried using different N_WAYS for smaller digits. I tested the execution time for 2500 digits and the result shows that a 2_way set associative cache performs best. This showed that the performance of higher way cache decreased slower as the NDIGITS (memory usage) grows.

TABLE III. MSEC FOR EACH CACHE WAYS (NDIGITS = 2500)

N_WAYS	Execution time (msec)
2	6377
4	7294
8	7474

IV. REPLACEMENT POLICY

A. LRU

The current implementation of the replacement policy is FIFO, which is simple but not effective. I implemented a LRU replacement policy and modify the victim_sel signal to choose the address to replace. I place the hit record in a table, the table size is N_LINES * N_WAYS * WAY_BITS. When an address hits, I will reorder the line in the table so that the curretly used element is placed at the last register in N_WAYS. The first element in the line (least recently used) will be selected as the victim.

B. Result

The evaluation metric used is the same as part II. The result shows that there are some improvements when changing the FIFO replacement policy into LRU cache, especially on the cache hit rate on read requests.

TABLE IV. CACHE STATISTICS (LRU)

Category	r/w	Percentage / #Cycle	Improvement
Cache Hit Rate	read	80.4 %	3.8 %
	write	84.6 %	1.6 %
Average Cache Latency	read	55.5 cycles	0 %
	write	55.5 cycles	0 %

TABLE V. IMPROVEMENT OF LRU CACHE (N_WAYS = 4, NDIGITS = 5000)

Method	Time (msec)	Time Improved %	Total Hit Rate
FIFO	30522	-	79.9 %
LRU	28987	5.3 %	82.2%

V. ANAYSIS & DISCUSSION

As you can see from part III and IV, changing the replacement policy will improve the performance more than simply changing the set size. Another way to improve the cache performance is to change the cache line size. I did not have time to implement this idea because it will require lots of modifications, but it is a possible way of improvement. Although the cache line size of the ARTY board is fixed, we can create two sets of tables to simulate the larger cache line size. The memory access of the PI program is nicely ordered (the memory address increase or decrease by 1 element), loading 8 values once might be faster than loading 4 values two times (though the cache latency might increase). This approach is suitable only for the PI program, using higher cache line size for other programs is likely to decrease performance.

VI. CONCLUSION

In this homework assignment, the cache behavior of the Pi program was analyzed under varying configurations, including the number of cache ways and replacement policies. The results show that while the cache performance is relatively stable across different associativities for large number of digits. Furthermore, implementing an LRU replacement policy improves cache hit rates and execution times compared to the default FIFO policy.