Cache Optimazation

Chih-Hsuan Chen, 110550029

***Keywords*—Cache; Processor; CPU; Aquila; DRAM ; Memory; CoreMark**

I. Introduction

Since the cost of TCM is too high and the speed of accessing the memory directly is too slow, many device use cache to get a relatively well performance and lower cost. The performance of a cache can be influenced by its size, associativity, replacement policy and so on, and also depends on the behavior of the process. Both the software and hardware factors should be considered.

This assignment is aim to analyze to behavior of CoreMark and try to optimize the cache. The performance of the cache will be evaluated by "Iteration/Sec" from the CoreMark test result and its hit ratio.

II. Implementation

A. Hit Ratio and the Behavior of Process

There are eight counters read/write_hit/miss_count/latency to and two flags counting_count/latency. The latency counters counts the total cycles of cache hits/misses when reading from or writing to the cache, and the count counters the total freqency of the above mentioned events. The average latency can be computed by dividing the total cycles with the total freqency that the event occurs. The flags indicate whether the counters should count or not.

When the cache receives p_strobe_i, since the count counters should only count exactly once when the event starts, so counting_count will be 1 only if counting_latency is 0 and will return to 0 at the next cycle, counting_latency will be 1 directly and return to 0 when the p_ready_o becomes 1.

The signal cache_hit is used to specify hit/miss. Since the signal p_rw_i only holds one cycle and the signal rw which registers p_rw_i delays one cycle than p_rw_i, if one of p_rw_i and rw is 1, the request will be specified as writing to cache.

After getting all the information above, the performance of the cache can be analyzed with hit ratio, average latency and CoreMark test result, and the behavior of CoreMark can be analyzed with the ratio of read and write.

B. Replacement Policy - Least Recently Used (LRU)

An array RECORD is used to keep track of the recently used order. If RECORD[i] is 0, it means that the i-th element in this set is the least recently used one; if RECORD[i] is n-1, where n is the associativity of the cache, it means that the i-th element in this set is the most recently used one. The element whose RECORD value is 0 will be the victim that will be replaced when a cache miss occurs.

When a cache hit occurs, the RECORD value bigger than the hit one in the c set will -1, and the RECORD value of the hit one will be n-1. When a cache miss occurs, all the RECORD value of in the accessed set will -1 (the RECORD value of the victim will become n-1 from 0, and 0-1 will equal to n-1 due to overflow).

C. Replacement Policy - Least Frequently Used (LFU) (1)

I implemented two kinds of LFU in this assignment, and this version will consider the total used freqency. There is an array RECORD counts the total used freqency, and an array RANK to record the ranking of the RECORD value in a set (i.e. the value of RANK[0] is the index that has the least RECORD value and it is the least frequently used one). If two elements have the same freqency, the one whose has larger index will has bigger RANK index. RANK[0]will be the victim that will be replaced when a cache miss occurs.

When a cache hit occurs,

- The RECORD value of the hit element (suppose it is the x-th element is that set) will +1.
- For every index i in RANK, if RECORD[RANK[i]]=RECORD[x] and RANK[i]>x, or RECORD[RANK[i]]=RECORD[x]+1 and RANK[i]<x, the ranking has to change (RANK[i-1]=RANK[i]). (Note that the value of RANK is index.)
- If RANK[i] has to change, but RANK[i+1] doesn't change or i=n-1, the new rank of x is i.

When a cache miss occurs,

- The RECORD value of the victim element will 1.
- The RANK doesn't change.

D. Replacement Policy - Least Frequently Used (LFU) (2)

This LFU version will consider the recent h times that the set is accessed. Every element has a shift register to record if it is the hit one when the set is accessed. The shift register will shift right and the new record will be put at the most left bit. And there is an array RECORD counts the total 1s in the corresponding shift register. This LFU version has the same RANK and RECORD machenism as the previous version.

III. Result

A. Behavior of CoreMark

1) Read and Write: Table I is the statistics with the original cache (4-ways, 2k, FIFO). It shows that CoreMark requested more than three times cache read than cache write. The read/write ratio is all the same regardless of the cache size, associativity and replacement policies for it is only depends on the behavior of CoreMark.

TABLE I
THE STATISTICS WITH THE ORIGINAL CACHE

Hit rate	*Miss rate*	*Read rate*	*Write rate*	*Iterations/Sec*
99.78%	0.22%	79.45%	20.55%	24.906596

TABLE II
THE STATISTICS WITH DIFFERENT REPLACEMENT POLICY

Policy	*Hit rate*	*Miss rate*	*Iterations/Sec*
FIFO	99.78%	0.22%	24.906596
LRU	99.81%	0.19%	24.949031
LFU(1)	87.11%	12.89%	13.624627
LFU(2) (history 16)	87.11%	12.89%	13.629757
LFU(2) (history 32)	87.11%	12.89%	13.629762
LFU(2) (history 64)	87.11%	12.89%	13.629756

2) Replacement Policy: The performance of each replacement policy is also closely related to the behavior of processes. Table II shows the statistics of running CoreMark with a 4-ways, 2k cache with different replacement policy.

We can find that LRU has slightly better performance than FIFO, it has higher hit rate and runs more iterations per second. On the other hand, LFU has extremely poor performance regardless which version and how many history recorded.

A feature of LFU is that the newly loaded data in the cache may be replaced with another data soon. Since when a data block is loaded into the cache, it has the least used freqency of all the data block in that set, and it will be the victim to be replaced when there is a cache miss. The version 2 LFU can alleviate this problem theorically for the too old history records will be discarded, but actually it has the same hit rate as the version 1, only runs slightly more average iterations per second and the size of history doesn't make any improvement.

We may suppose that CoreMark doesn't use some specific cache data in high frequency, so no matter what kinds of LFU have poor performance. But it may still have a little tendency to use the data block that it just used, hence it has better performance with LRU than FIFO.

TABLE III
THE STATISTICS WITH DIFFERENT CACHE SIZE AND ASSOCIATIVITY

Associativity	**Cache size**	**Policy**	*Avg hit latency*	*Avg read miss latency*	*Avg write miss latency*	*Hit rate*	*Miss rate*	*Iterations/Sec*
2	2k	FIFO	1	50.363	50.182	99.69%	0.31%	24.758975
2	4k	FIFO	1	50.393	49.044	99.97%	0.03%	25.212755
2	8k	FIFO	1	42.736	47.053	100.00%	0.00%	25.264906
4	1k	FIFO	1	50.309	50.257	98.55%	1.45%	23.046004
4	2k	FIFO	1	50.394	50.228	99.78%	0.22%	24.906596
4	4k	FIFO	1	50.364	47.897	99.98%	0.02%	25.227126
4	8k	FIFO	1	34.567	44.654	100.00%	0.00%	25.26491
4	16k	FIFO	1	31.000	44.836	100.00%	0.00%	25.264918
8	2k	FIFO	1	50.362	50.231	99.78%	0.22%	24.905506
8	4k	FIFO	1	45.135	46.765	100.00%	0.00%	25.264845
8	8k	FIFO	1	32.075	44.647	100.00%	0.00%	25.264917
4	2k	LRU	1	50.386	50.220	99.81%	0.19%	24.949031
4	8k	LRU	1	32.585	45.763	100.00%	0.00%	25.264914

B. Latency

Table III is the statistics of the cache with different size and associativity. The hit latency of the cache is always 1 cycle, since the cache can prepare the requested data within a cycle if that data is already in the cache. As for the miss latency, the read and write latency are roughly equivelent as the cache size is less than 4k, and the write miss latency is higher as the cache size is larger the 4k.

C. Effect of Cache Size and Associativity

From Table III, we can observe that the smaller the cache is, the less hit rate it has and less iteration per second the processor runs. This is intuitive because a larger cache can cache more data, and then reduce the direct access to the memory, which costs lots of time.

When the cache size is fixed, the more associativity will give better performance (it's obvious when the cache size is 2k), but the difference becomes smaller as the cache size is larger. This phenomena is also reasonable the more associativity means that each set can store more data block and the blocks will be replaced less frequently, which can further reduce the cache miss. But when the cache size grows, the cache can store more data blocks originally, so the effect of more associativity decreases.

Table III also shows that the larger the cache is, the less average miss latency it has, and the more associativity the cache is, the faster the average miss latency declines. While a larger cache will increase the latency theorically.

IV. DISCUSSION

Althought the larger cache can provide better performance, but considering its cost, we probably can try increasing the associativity to improve the cache when the resource is limited.

As for the replacement policy, LFU requires the most space but has the worst performance, LRU has a little better performance than FIFO but also requires more space so that it can keep track of the recently used order. We need to test FIFO and LRU with other programs to determine which replacement policy is better.

REFERENCES

[1] https://myapollo.com.tw/blog/interview-question-cache-replacement-policies/#google_vignette