CPEN 411 Assignment 4

Written by: Pooya Daravi
Student Number: 34584145

Part (A)

We can see that for part (a) the instruction miss rate for the 4-way cache (11.1%) is bigger than the direct-map cache (5.9%). This could be due to the smaller block size of the 4-way cache. It appears that for the fpppp benchmark the instructions might be generally consecutively stored in instruction memory (with little jumps), it would therefore be very helpful for the miss rate to have a block size twice as big. Therefore, the 4-way cache has roughly twice the miss rate of the direct map cache. (it seems that for this fpppp test for example, conflict misses are not substantial) This hypothesis can be verified by increasing the block size of the 4-way cache to 64B while maintaining its capacity to get a miss rate of 5.6%. This is a slightly better result than the direct map version due a decrease in conflict misses. It is expected that this improvement would be relatively small since the instructions are generally consecutive with different addresses and therefore conflict misses are not as substantial for instruction caches as they are for data caches.

Part (B)

The load and store miss rates are very low. This shows that programs have a high degree of locality that would decrease the miss rates to under 1%. We also observe that the store miss rate is about 1/3 of load miss rates. This could be due to the temporal precedence these two instructions. Naturally it would be common practice for programs to load data, apply computation to them and save them back. This means that if the load and save are close enough the load would be a cache miss while the store would result in a cache hit. The relatively high rate of write-back rate could be due to the fact that a lot of the stored values are being evicted due to misses. Increasing the capacity and associativity of the cache would be one simple but costly solution to improve this rate:

	fpppp	gcc	go	vpr
LD	0.16	0.99	0.92	1.8
ST	0.038	0.38	0.52	0.59
WB	0.44	2.2	3.2	5.0

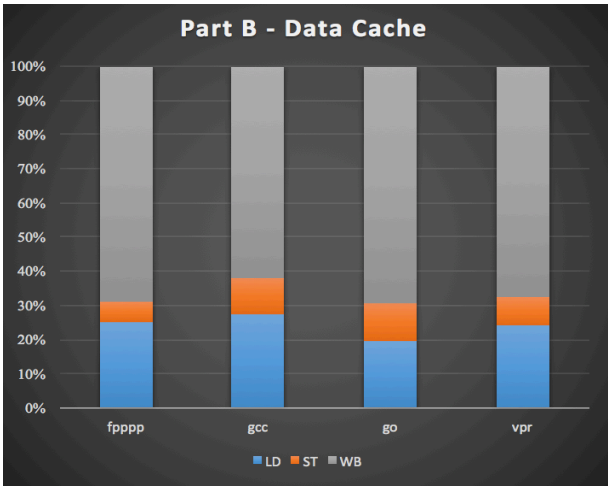

Table 1 and Figure 1 – Load/store miss and write-back rates for part B

Part (C)

For this part I have designed a chain structure that would allow for simulation of an arbitrary number of levels of cache:

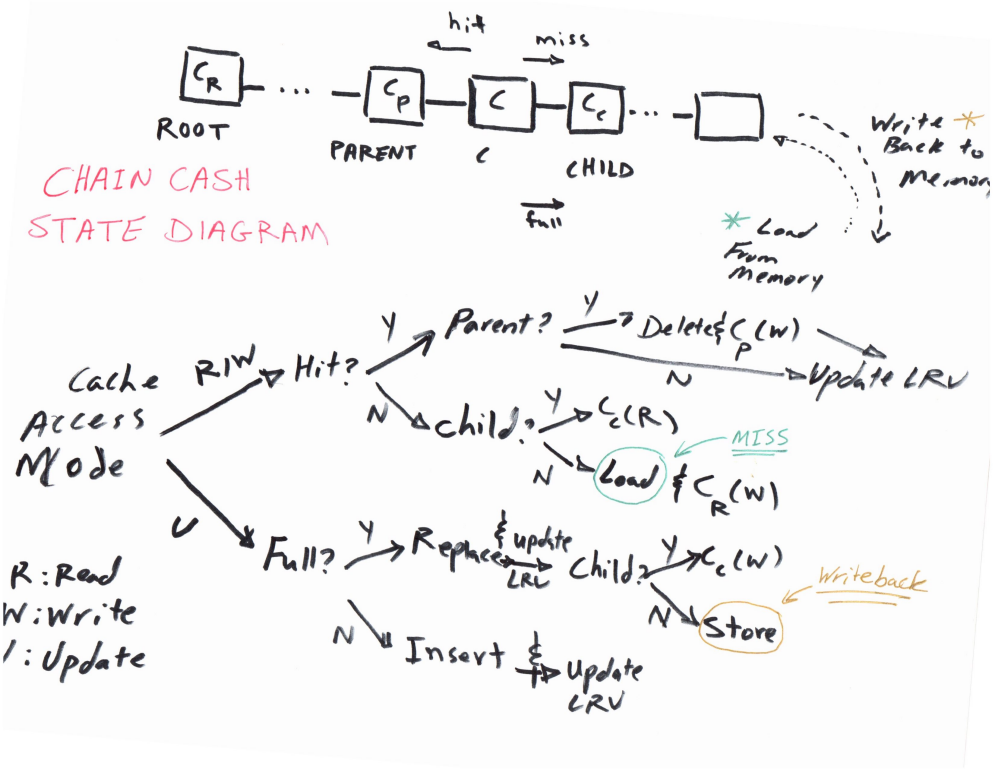

Figure 2 – Chain cash state diagram designed for part C

In this recursive definition, the missed reads result in a read from the child node and hit reads result in a write to the parent node. The values loaded to memory are sent directly to the top most level (the root cache). Other loading policies would also be possible. (for example, loading to the last layer which is closest to the memory) However, I have chosen this policy as that is the mechanism that could be used to simulate a victim cache behavior. We see from the results in the table below that as we increase the associativity, the miss rate decreases. (to different degrees depending on the benchmark) This shows that unlike the instructions, with data there is a high degree of conflict misses that could be avoided by increasing associativity. Like before, we generally see higher load miss rates than store miss rates:

	fpppp	gcc	go	vpr
LD-D	5.96%	2.74%	5.19%	3.52%
ST-D	7.28%	1.19%	1.65%	0.94%
LD-2W	0.86%	1.41%	1.98%	2.24%
ST-2W	0.50%	0.51%	0.83%	0.66%
LD-4W	0.19%	1.09%	1.41%	1.91%
ST-4W	0.05%	0.41%	0.68%	0.62%

Table 2 – Part C results

Comparing these values with the configuration with no victim cache, for the fpppp benchmark, we see that the victim cache results in a very slight improvement for load instruction miss rates, while not helping the store instruction miss rates:

	LD	LD+V	ST	ST+V
Direct-map	6.0132	5.9648	7.2842	7.2842
2-way	0.86078	0.8606	0.49717	0.49717
4-way	0.18562	0.18556	0.053987	0.053987

Table 3 – Miss rates with and without a victim cache for the fpppp benchmark

This could be due to an error in the implementation of the victim cache code or due to the nature of the capacities and access rates of the main and victim caches. Further investigation is required.

Part (D)

It appears that the prefetcher, due to similar reasons as mentioned in part (A), decreases the miss rate by increasing the capacity to take advantage of spatial locality, which would be a very likely characteristic of consecutive instructions unless there is a high number of jumps per instruction:

	fpppp	gcc	go	vpr
LD	11.060%	3.410%	2.988%	0.025%
LD+PF	5.550%	2.263%	1.642%	0.014%
SR	99.572%	77.623%	91.131%	90.745%

Table 4 – Miss rates w/ and w/o a prefetcher and the success rate of prefetcher

We can see that as the miss rate improves (LD/LD+PF) so does the success rate. This could be due to the same factor as explained above, namely the fact that instructions are usually encountered consecutively and thus by prefetching the next instruction we are likely to have prefetched an instruction that would have been missed otherwise

Post Script

At the time of writing the report I have realized that the SimpleScalar instructions are only 8-byte addressable. This means that in my implementation of the instruction cache I should have shifted the PC by 3 bits before taking the offset and index bits. Having done the masking correctly I would have likely had a higher hit rate for sections (A) and (D).