# Lab 1 – Report

## Lab goals

This lab has the objective to analysis the impact of cache sizes on performance for different benchmark applications. We modify cache parameters like size, block size and associativity to study the effect of each one regarding performance and try to find the optimum configuration for the given benchmarks.

## Methodology

We use the SimpleScalar tool set that simulates a processor (described in a configuration file) that executes given programs (cf. table 1). We run the programs for each configuration and measure the individual CPI and MPI for the target cache. To find the optimum configuration, we considered the instruction and data cache separately and optimized in the order cache size, associativity and block size while picking the best configuration before moving to the next step. This gives clean speed up values per cache while we still can combine the optimal configuration afterwards. For adjusting the cache size, we increased the number of sets. As the cache size is the product of nsets, block size and associativity, we may have to adjust nsets when changing the other parameteres to keep the cache at the same size.

As two configurations only differ in one cache parameter the CPI is comparable. We use the default configuration as a reference for calculating speed ups and will refer to the calculated speed up for convenience. The default configuration has a cache size of 4kB, block size of 32 bytes and associativity of 2. By setting memory latency to one, we can figure out the optimal CPI value per program. This behaves like an optimal cache since all data is available within one clock cycle.

## Simulation results and observations

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | *dijkstra* | *qsort* | *stringsearch* | *gsm-untoast* | *jpeg-cjpeg* | *geom. mean* |
| *SPOPT1* | 4.38 (95,3%) | 5.43 (75,1%) | 6.33 (78,1%) | 1.76 (99,4%) | 2.90 (93,9%) | 3.78 (87,8%) |
| *SPOPT2* | 3.72 (80,9%) | 5.34 (74,0%) | 5.96 (73,5%) | 1.75 (98,5%) | 2.66 (86,2%) | 3.53 (82,1%) |
| *SPIDEAL* | 4.60 | 7.23 | 8.10 | 1.77 | 3.09 | 4.30 |

Table 1: final speed ups for given benchmark programs. Speed up refers to base config without modification

Table 1 gives SPIDEAL per benchmark program, which is the upper bound for speed up. This is a measure for memory usage of each application. A lower speed-up implies that the application does less suffer from memory latency, hence “stringsearch” and “qsort” are the applications where the memory subsystem is most responsible for execution time. We pick them as target programs for the following optimizations of data and instruction cache.

|  |  |  |
| --- | --- | --- |
|  | *SPD* | *SPI* |
| stringsearch | 1.0076 | 7.0386 |
| qsort | 1.2035 | 2.5738 |
| geom. mean | 1.1012 | 4.2563 |

Table 2: Final speed ups per cache type

As we optimized the data (cf. figure 1) and instruction cache (cf. figure 2) separately, we can give separate speedups for either of them. Whereas optimizing the data cache lead to a speedup of 1.1, the optimized I-cache has a speedup of 4.3. Hence improving the instruction cache has a higher effect on the overall performance. Even with the optimized configuration we do not reach the ideal speed up (cf table 1: SPOPT1). This has two reasons: The cache might still be too small to fit all the needed memory blocks. Hence blocks are requested multiple times from memory (causing miss penalty) and second: The first time a memory block is accessed it is not yet cached, so even if the cache is big enough to store all accessed blocks there is some miss penalty for the first fetch.

Comparing the different parameters in figure 1/2 the cache size has the highest impact on speed up for stringsearch and qsort (only I-cache). Performance of qsort depends highly on the block size of the D-cache. Hence fine-granted optimization still depends on the program code.

Figure 1: Performance development for data cache

Figure 2: Performance development for instruction cache

SPOPT1 (cf. table 1) combines both optimizations, but ignores increasing cache latency for higher cache sizes. Therefore, SPOPT2 considers the higher latency for bigger caches. This leads to decreased performance but is still much better than the base configuration (82.1% of SPIDEAL). For this setup the latency “destroys” the improvement that has been made so far as soon as it reaches a value of about 30. This causes the speed up for the GSM-application to drop below one, hence this is worse than the base configuration.

## Conclusion and learning outcome

We can conclude that the cache configuration influences the performance of the system, especially the cache size has a big impact on performance – even if this causes higher latency accessing the cache. Associativity and block size have less influence, but still cause some improvement. For these applications the changes that have been made to the cache configuration are still worth if we consider increased cache latency.

We’ve seen that performance of these benchmark applications is more dependent on the instruction cache than the data cache. When we modify the instruction cache configuration this leads to better performance than applying the same changes to the data cache.