DAT105 Computer Architecture: Lab 1: Exploring the Impact of Cache Hierarchy on Processor Performance

Max Villing, Vinaykumar M K, Lab Group 4

September 2021

0.1 Project Goal

The goal of this lab was to gain a deeper understanding of the way in which cache design impacts processor performance. In addition the optimal configuration of a processor for performing two memory intense benchmarks was also sought. Afterwards in order to learn about the different aspects of cache designs attempts were made to optimize the design of a processor by looking at two benchmarks which were particularly affected by the memory subsystem.

0.2 Methodology

The lab was performed with the use of the simulator simhome which was used to run several benchmark programs on a simulated processor. First the simulator was used to perform 5 different benchmark programs, these being quicksort, dijkastra, gsm-untoast, jpeg-cjpeg and stringsearch-cabce. By looking at the performance results for each of these it was possible to observe the impact the memory subsystem had on each. Furthermore it was possible to identify which programs were most affected by the memory.
In order to find a optimal cache design for the two most affected programs a simple approach was employed. There are three aspects of the cache that can impact performance, associativity, block size and overall cache size. In order to determine which value for each was ideal each was tested in isolation. Values for each were tested, the CPI (cycles per instruction) for each of the two benchmarks was noted down, and then the simulation was retried with the value of the parameter doubled. This was until values that either resulted in no performance change or worsened performance were found. Once ideal values for all three were found they were then used to simulate an ideal execution of the benchmarks with the best values for all three cache aspects.

1 Important observations and Trade-offs

After running the simulator we looked at key performance indicators relating to the memory subsystem for each of the five benchmark programs. These values can be seen in the table 1. We note that all programs had the potential to be speed up and also note that the potential was highest among quicksort and stringsearch and hence they will be used for the cache design section of the project.

Application	dijkstra	qsort	stringsearch	gsm-untoast	jpef-cjpeg
Instruction Count	54881769	4189644	300884	11704482	27259353
Execution Time in cycles	486614356	636137244	5326285	39443875	162107216
CPI base	8.86659	15.18276	17.70212	3.369980	5.94684
MPI I-L1	0.02863	0.059569	0.095626	0.007952	0.0146944
MPI D-L1	0.01085	0.03808	0.003396	0.121506	0.010646
CPI0	1.9848	2.2021	2.3838	1.91	1.948
SPideal	4.46724	6.89467	7.42600	1.76438	3.052796

Table 1: Simulation results for base configuration

2 Conclusion

Based on the results from the five benchmarks we conclude that while all programs had the potential to be speed up their individual potentials varied greatly. We find this to be logical since the nature of their tasks differed greatly. While a calculation intensive program like quicksort had a large potential for improvement something like gsm-untoast had far less potential which is reasonable since calculations are very dependent on memory access while gsm-untoast is far less bottlenecked by memory access.
We also found that for most benchmarks (gsm-untoast is the exception) the instruction cache missed far more than the data cache and thus improvements to its performance would have a greater impact on the performance.
When it comes to processor design we found that while different choices for associativity, block size and overall cache size can have a significant impact on performance (our collected data can be found below) the benefits were not shared equally by both benchmarks. Stringsearch generally saw diminishing returns and often stopped seeing any gains much sooner than quicksort. In one case stringsearch even started losing performance as block size increased. We find this to be logical since stringsearch is doing very different calculations compared to quicksort, since the later is mainly concerned with comparing and moving values it makes sense for it to benefit more. We also found that endlessly increasing block size and associativity would not yield performance improvements since there is a trade off with increased cycle latency.
Below is performance data indicating the impact associativity, block size and cache size had on the CPI for each of the two benchmarks.

Block size	qsort	stringsearch
64	13.6346	17.6132
128	12.7482	17.6199
256	12.7482	17.5627
512	12.2106	17.5594
1024	12.1317	17.5995
2048	12.0917	17.5601

Table 2: Simulation results for Block size

Associativity	qsort	stringsearch
4	14.8432	17.6103
8	14.5693	17.6022
16	14.2914	17.5997
32	13.9915	17.5997
64	13.6822	17.5997
128	13.3735	17.5997
256	13.0647	17.5997
512	12.7562	17.5997
1024	12.3812	17.5997
2048	12.1392	17.5997
4096	12.1392	17.5997
8192	12.1392	17.5997

Table 3: Simulation results for Associativity

Cache size	qsort	stringsearch
8192	14.8252	17.621
16384	14.5028	17.6068
32768	14.1943	17.5997
65536	13.8758	17.5997
131072	13.5619	17.5997
262144	13.2559	17.5997
524288	12.942	17.5997
1048576	12.6112	17.5997
2097152	12.3005	17.5997
4194304	12.1392	17.5997
8388608	12.1392	17.5997
16777216	12.1392	17.59967

Table 4: Simulation results for Cache size

(Note for each table that even when we hit a point where increasing values gave no benefits we opted to try a few more values just to confirm that we had hit a plateau.)