CSC367: Assignment 1

Faisal Shaik

August 2024

1 Memory Bandwidth Experiment

2 Cache Hierarchy Experiment

2.1 Introduction

Computers have a hierarchy of memory systems, with the fastest and smallest memory being the CPU registers and the slowest and largest memory being the hard drive. The cache is a small, fast memory that is used to store frequently accessed data. This report details an experiment that measures the performance of a cache hierarchy using a simple matrix. We seek to determine the different cache sizes and latencies for the cache levels in the hierarchy.

2.2 Methods

The experiment was conducted on the DH 2020 lab machines. These featured a **64 byte cache line**. The method used in this experiment can be broken down into two steps,

a. **Fill Up Cache Levels:** We loop through a large matrix that will, by assumption, be large enough to fill up the cache levels and further invade direct memory. At each iteration i, we have a "working set" of data; the first i rows of the matrix. In each iteration, we loop through every row in our working set and access the middle element, to "load" the rows into the cache. This gradually fills up the cache levels as the working set grows.

b. **Measure Access Time:** After touching i rows in the working set at the i^{th} iteration, we then write 64 bytes of data to the first row of the matrix. We measure how long this write takes, which is a measure of the write latency.

The code for the experiment as described above can be found in the `part1/part1b.c` file. The idea is that at each iteration, the working set will fill more and more of the cache levels, and eventually the first row of the matrix will be evicted from the cache levels, causing a cache miss and yielding a higher latency. The matrix had **262144 rows**, each row being 64 bytes long, for a total size of **16 MiB**.

Data Collection:

We can then plot the latency against the size of the working set to determine the cache sizes and latencies for the cache levels in the hierarchy as shown in the results section. The code for generating the graphs can be found in the `part1/gen/generate_graphs.py` file. The plots were generated in the following steps:

a. The data was collected by running the `part1/part1b.c` program using `make run`. The `make run` command invokes a bash script `myscript.sh` that runs the C executable which generates a CSV file containing the latency and working set size data.

b. The CSV file was then read and plotted using the `part1/gen/generate_graphs.py` script, which is invoked by `myscript.sh`. The script first makes sure that all the necessary Python packages are installed. Specifically, `matplotlib` and `pandas` are used to generate the scatter plots and box plots.

2.3 Results

Below are the results of the experiment. All graphs were generated using the `part1/gen/generate_graphs.py` script.

For the scatter plots in *Figure 1* and *Figure 2*, the x-axis represents the size of the working set in bytes, and the y-axis represents the latency in nanoseconds. Both plots shows the latency of writing to the first row of the matrix as the working set grows.

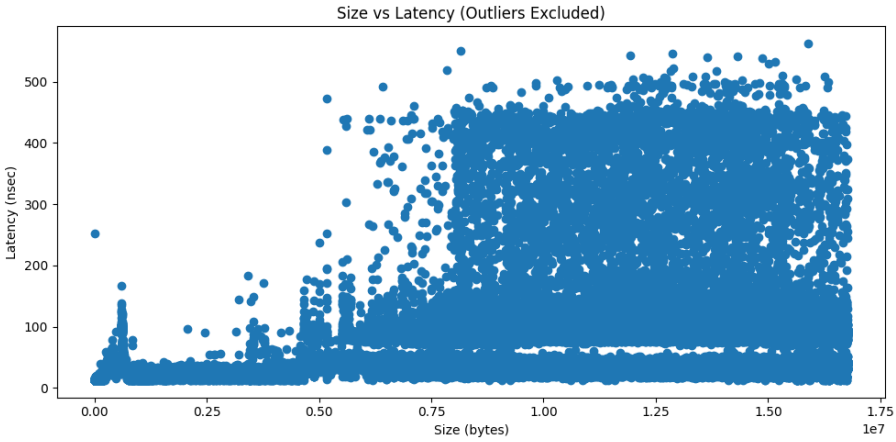

Figure 1: Latency vs. Working Set Size

Here is another plot better showing the density, and also provides an average line for the data.

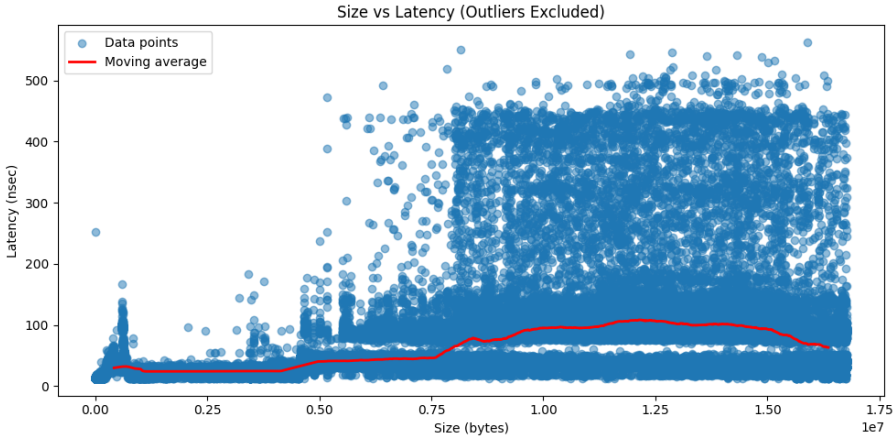

Figure 2: Latency vs. Working Set Size with Average Line

Here is a box plot of the data, showing the distribution of the latencies at three different intervals corresponding to the patterns in the scatter plots.

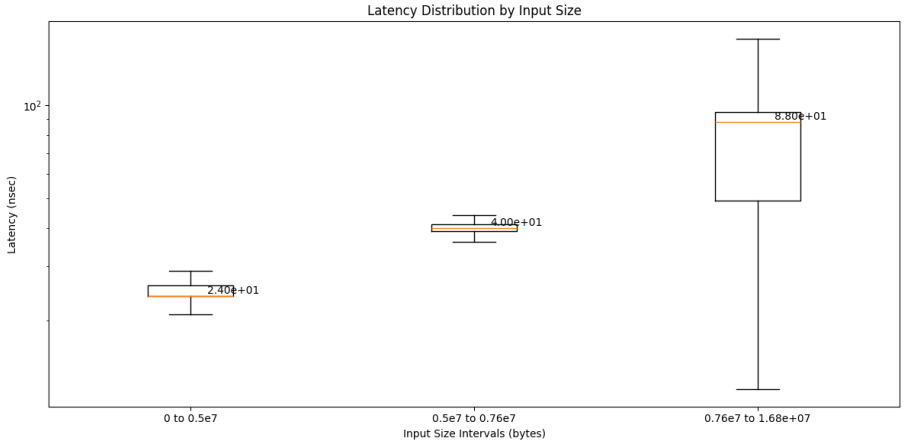

Figure 3: Box Plot of Latencies

2.4 Discussion

The plot in *Figure 1* shows a clear pattern of increasing latency as the working set grows. However, it is clear that there is a lot of noise in the data. We have many low latencies at even large working sets.

That being said, we can still observe a ”staircase-like” behaviour, where the latencies seem to jump to a new upper bound after passed certain size thresholds. The box plot in *Figure 3* has been made corresponding to these thresholds. From 0 to 5×10^6 bytes, the average latency is 24 nanoseconds, from 5×10^6 to 7.6×10^6, the average latency is 40 nanoseconds, and from 7.6×10^6 to 1.69×10^7, the average latency is 88 nanoseconds.

However, if look at the upper bounds instead while ignoring outliers then we have the following:

- From 0 to 5×10^6 bytes, the upper bound is 30 nanoseconds.
- From 5×10^6 to 7.6×10^6, the upper bound is 130 nanoseconds.
- From 7.6×10^6 to 1.69×10^7, the upper bound is 460 nanoseconds.

This could possibly correlate to the L1, L2, and L3 cache levels respectively. However, we never got to see another bump, which we would expect for the main memory. This could be due to the fact that the matrix was not large enough to fill up till main memory.

A possible explanation for the large amount of noise in the data could be due to inconsistent cache eviction. Perhaps the cache eviction policy is not as simple as we assumed, and does not always

evict like a FIFO queue. This could explain why we see low latencies at large working sets, as the cache may not always evict the first row of the matrix.

3