# 6100 Project 1 Report

## Cost of Resources

In order to find the best cache configuration, we first consider the tunable parameters. All values are expressed as log base 2. The cache size in bytes (C), block size in bytes (B), number of ways per set (S), page size in bytes (P), Translational Lookaside Buffer (TLB) size in number of translations (T), and the size of memory in number of pages (M). While more resources may be seem desirable for many of these, the power consumption and latency is proportional to their size, so one must allocate resources to where it matters the most.

Although we will be directly modifying C, B, S, P, and T, the metadata required to maintain those sizes will change dependent on the parameters too. The tag store size, as it relates to the cache configuration is shown below.

$$storesize = 2^{(C-B)} * (P + M - (C - S) + 2)$$

Each tag is the size of the physical address minus the size of the block offset and index bits with an additional two bits for the valid and dirty bits. On top of that there are as many tags as cache blocks. It can be seen that a larger cache, physical address space, and associativity drive the tag store size up, while larger blocks limit the size.

The HWIVPT and TLB also have defined sizes based on these parameters. The HWIVPT size is defined below.

$$hwivptsize = 2^{M} * (M + (64 - P) + 1)$$

Each translation requires M bits for the physical frame number, 64 – P bits for the virtual page number, and an extra valid bit. There are as many translations as physical memory frames. A larger physical memory will drive the HWIVPT size up and larger pages will limit its size. The TLB is a similar structure with the following size.

$$tlbsize = 2^{T} * (M + (64 - P) + 1)$$

Each translation in the TLB is the same size and there are $2^{T}$ translations. Again, a higher T and M will drive the TLB size up while larger page sizes works to limit the size.

In general we will work for a solution that limits C to reduce the data and tag store size, S to reduce the cost of high associativity comparisons, T to reduce the size of the TLB, and M to reduce the size of physical memory, HWIVPT size, and TLB size. We will also prefer solutions that have higher B to reduce tag store size, and higher P to reduce HWIVPT and TLB size.

## Search Space Reduction

With a simulator that can report an estimate on how quickly a given cache configuration will run a specific trace of memory accesses, we have the ability to optimize the configuration. Unfortunately, the amount of tunable parameters makes for a very large search space that can take a lengthy time to run. In order to reduce the space to only configurations that are valid candidates, the following was employed.

Cache and block size should be easily narrowed down to the upper range of values. By maximizing the amount of physical/virtual memory resources available to minimize the effect of page and translation faults and holding constant, an insight into good values for C and B can be accomplished.

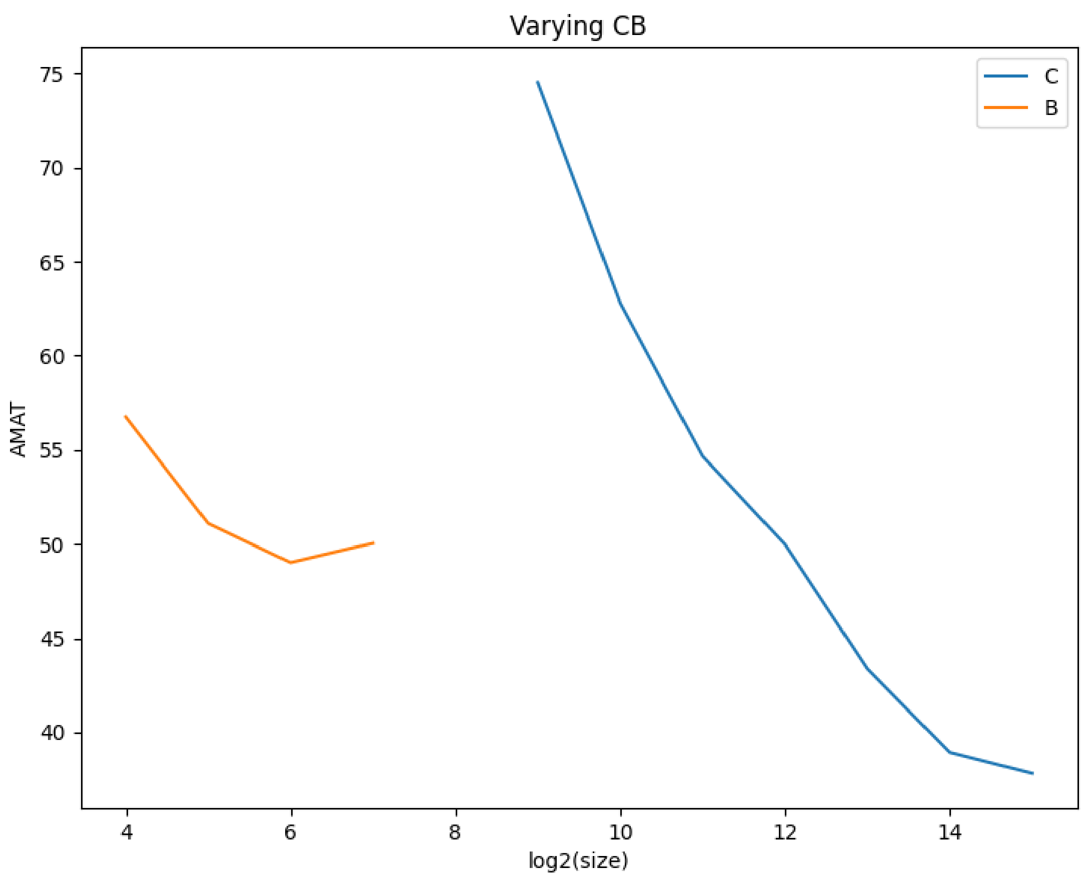

**Figure 1.** Average access time over all traces with cache size and block size varied. (plot_cb_trim.py)

All possible values for C and B were run while holding physical/virtual memory resources at the maximum values for both a fully associative and directly mapped cache. The averages are plotted above, showing that a block size of 16 Bytes can be safely eliminated from the possible candidates, as well as the bottom half of the caches sizes (512B – 2KiB) can also be eliminated.

A few more assumptions can be made to decrease the total space. Namely, we will limit S to (C-B-5) to allow the TLB to have at least 32 entries, otherwise we will suffer from translation faults. With a smaller range of C (12-15), B (5-7), S ((C-B-5)-(C-P)), P(9-min(14, C)), T (5-(C-B-S)), with the exception of M having a rather large range, a more extensive search can be made. By holding M to the maximum amount of physical memory to reduce the effect of page faults on the results, all configurations within this range were run.

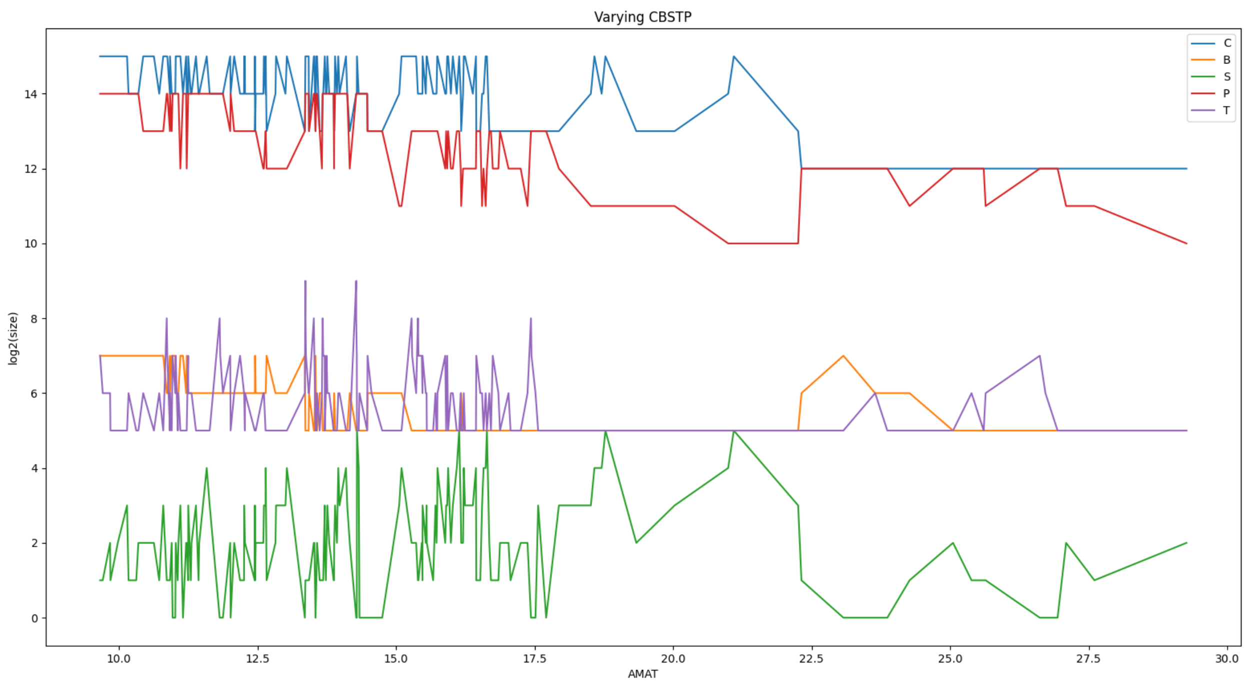

**Figure 2.** Varying configurations of C, B, S, T, P sorted by average access time over all traces. (plot_cbstp_choice.py)

The plot above shows the different configurations plotted with respect to average access time over all traces (note AMAT is on x-axis and further left is better!). It can be easily seen that low values for P (red line) and C (blue line) do not show up in the best runs and would be bad choices. It can also be seen that the highest values for S (green line) and T (purple line) do not appear in the best runs and would also be bad choices. All of the best runs have a block size of 128 Bytes (orange line) and a page size of 16 KiB. Although these extrema can be pointed out, there is a lot of variance in the configurations for top performers for most of the parameters.

## <u>**Best Overall Configuration**</u>

To chose the best configuration overall, the top 50 performers were selected from the previous experiment. The size of physical memory was varied over the entire possible range for each of these samples and run on every trace. **Figure 3** shows that M is not very correlated with the performance, meaning we can probably get away with choosing a value on the lower end. This plot also illuminates which configuration for CBSPTM we should choose. Starting from the left, we want to choose the first configuration that has reasonable values for C, S, T, and M, the parameters we said we want to limit. We could also try and maximize B and P, but all of the best runs have the same values with the block size at 128 Bytes and page size at 16 KiB, making our job a bit easier. The first value with reasonable parameters is C=15, B=7, S=1, P=14, T=6, and M=14 with an average AMAT over all the traces at 9.71 ns.

Using the formulas defined in the section <u>**Cost of Resources**</u>, this tag store will require 4096 bits (512 Bytes), the HWIVPT will require 1,064,960 bits (130 KiB), and the TLB will require 4160 bits (520 Bytes).

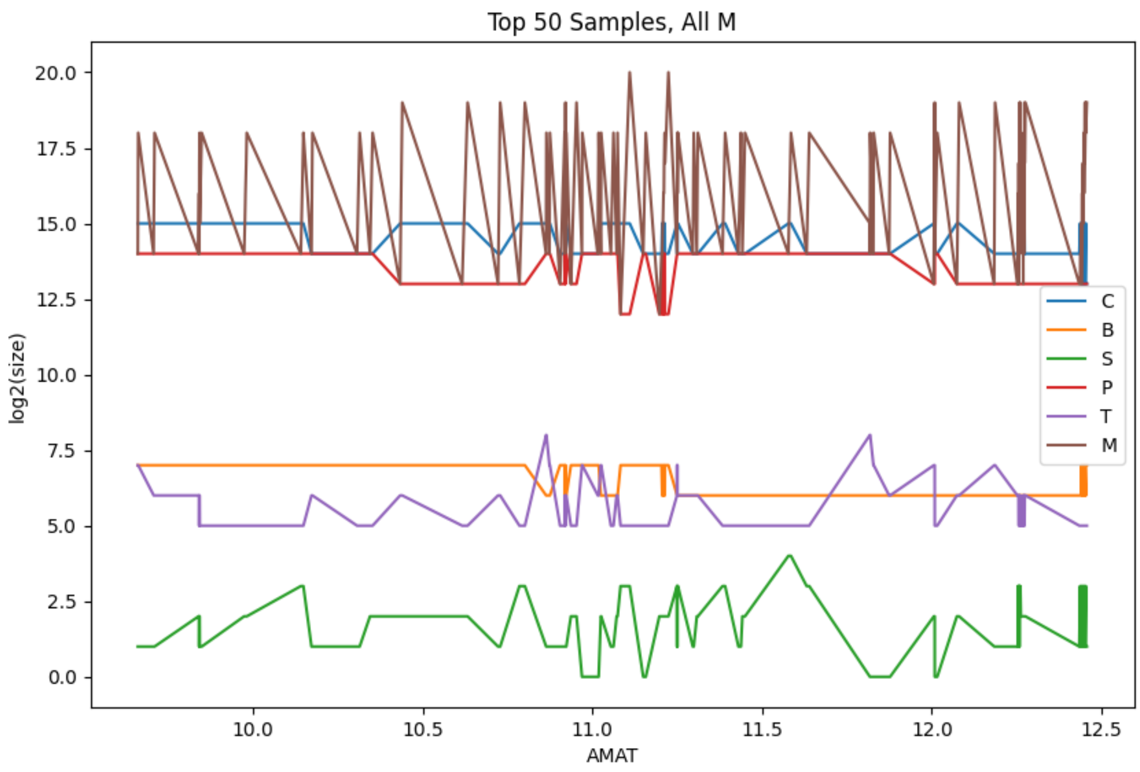

**Figure 3.** Varying configurations of M over the top 50 performers in the C, B, S, T, P variations. (plot_final_choice.py)

## <u>Validation</u>

To make sure that a local minima wasn't found after reducing the search space, the entire search space was run over all quickly computable traces (this ended up being every trace except linpack).

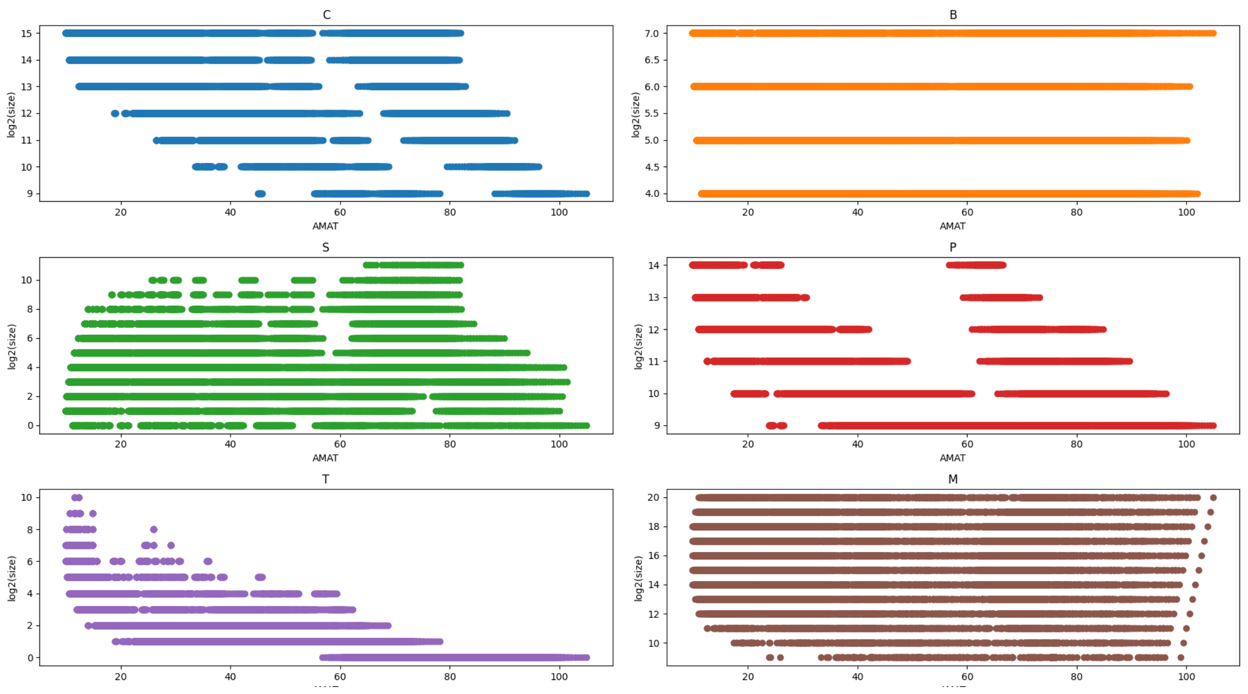

**Figure 4.** Average AMAT plotted for each parameter in the entire configuration space for all run-able traces. (validation_all_configs.py)

Our chosen configuration (C=15, B=7, S=1, P=14, T=6, and M=14) was a top performer in this dataset as well, confirming that we have found a truly top performing cache configuration.

## Best Configuration for Each Trace

Now, instead of looking at the average AMAT over all the traces, each trace is analyzed individually to find the best configuration for that workload. Using the same method as described in **Best Overall Configuration** the following results were obtained.

| | |
|---|---|
| GCC: | (C=15, B=7, S=2, P=14, T=6, M=14) |
| leela: | (C=15, B=6, S=1, P=14, T=5, M=14) |
| linpack: | (C=15, B=7, S=1, P=14, T=5, M=14) |
| matmul_naive: | (C=15, B=7, S=1, P=14, T=6, M=14) |
| matmul_tiled: | (C=15, B=7, S=1, P=14, T=6, M=14) |
| mcf: | (C=15, B=7, S=1, P=14, T=6, M=14) |

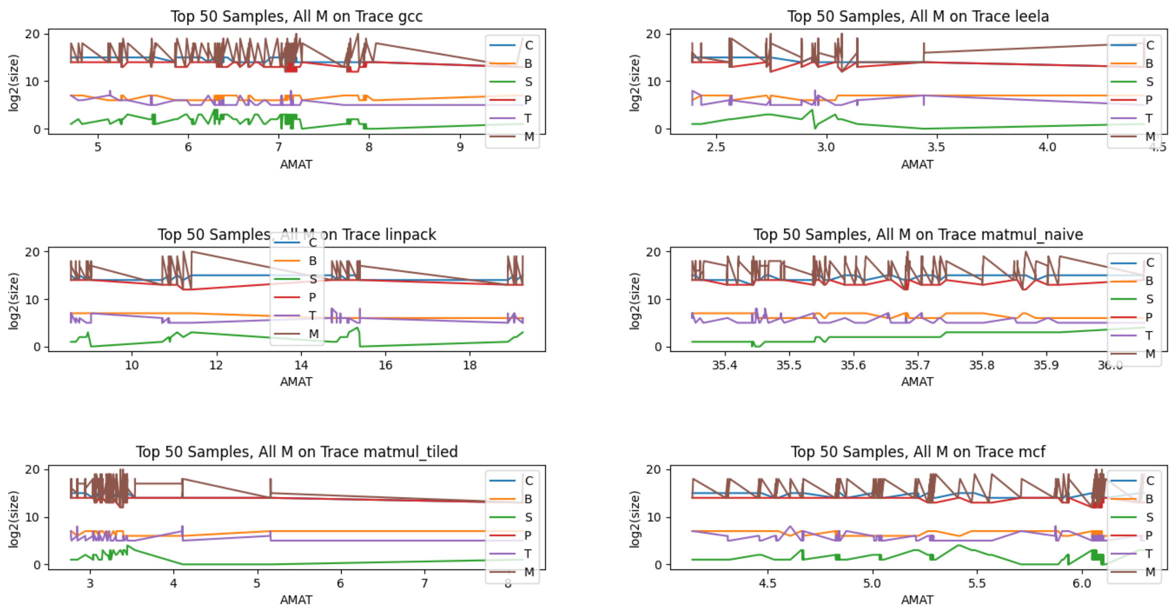

**Figure 5.** Top 50 samples from C, B, S, T, P experiment with varying M for each separate trace.

For the most part, our choice for best overall cache applies well to each of the individual traces. However, the GCC trace seemed to benefit more significantly from adding more associativity and TLB size and the Leela trace seemed to benefit from a smaller block size and wasn't able to make use of a larger TLB. Linpack also didn't benefit from a larger TLB.

## Conclusion

An optimal cache can be hypothesized by running simulations on real memory traces and some data analysis techniques. The cache proposed here balances the size to reduce power consumption and latency as well as insure a close to optimal hit rate.