Anas Hamed - ahamed@buffalo.edu - ID: 50295901 - Graduate Level
Professor Ramalingam Sridhar
CSE 590 - Computer Architecture
6 May 2019

Computer Architecture Project #2

Introduction

This project aims to measure the performance of an X86 processor for different cache parameters. The way this was done was by simulating the execution of an arbitrarily large number of instructions, then measuring the performance by taking note of various metrics relating to cache performance using benchmarks. The simulation tool that was used to set up the architecture and run the simulations is Gem5, a highly sophisticated simulation platform designed for the very purpose of analyzing processor performance. The following benchmarks were used in the simulations:

- 401.bzip2
- 429.mcf
- 456.hmmer
- 458.sjeng
- 470.lbm

The performance of the architecture will be discussed and analyzed through the results obtained from the simulations using the aforementioned benchmarks. First, the commands and programs used to obtain the results will be discussed. Then, each of the following sections will go through a specific parameter of cache memory, and draw conclusions on the effects of the parameter on the overall performance of the processor.

For each of the cases, one parameter is varied while the others are held constant. The metrics that pertain to the variable parameter are then plotted for each of the benchmarks.

For each parameter discussed, certain benchmarks that exhibit a relatively high variance in values will be considered for analysis. In addition to these measurements, all the metrics that were obtained for each and every benchmark for every case where a certain parameter has been varied are provided along with this

report. These have been extracted directly from the stats file that was generated by the simulation platform.

Programs and Commands

In order to execute a simulation for a single benchmark, the developers of Gem5 offer a command line utility that accepts flags which correspond to the setup of the simulation. For each benchmark, the following command was used for different parameters:

```
time build/X86/gem5.opt -d "$HOME/bin" configs/example/se.py -I "100000000" -c "$BENCHMARK" -o "$BENCHMARK_ARGS" --caches --l2cache --l1d_si
ze="$L1D_SIZE"  --l1i_size="$L1I_SIZE"  --l2_size="$L2_SIZE" --l1d_assoc="$L1D_ASSOC" --l1i_assoc="$L1I_ASSOC" --l2_assoc="$L2_ASSOC" --cachel
ine_size="$CACHE_BLOCK_SIZE"
```

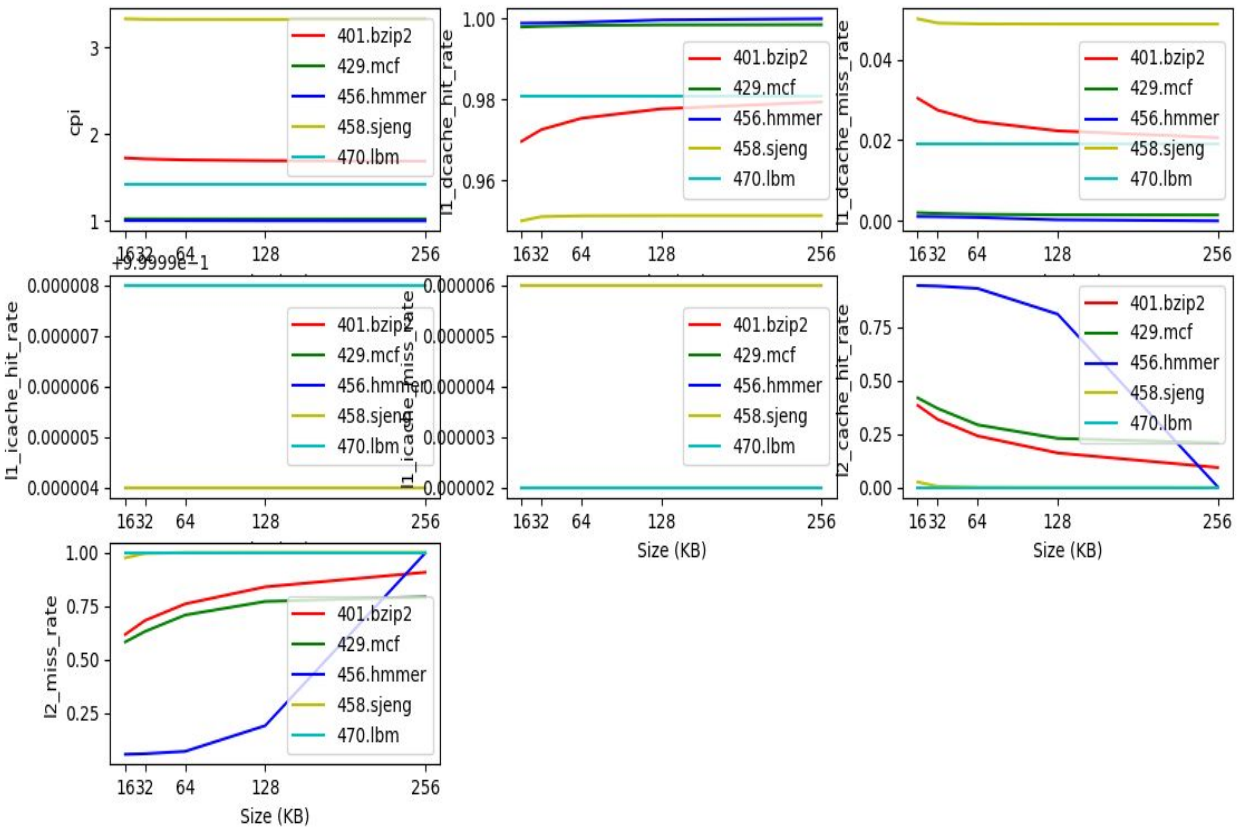
The values of the variables provided to the flags were changed for each execution to reflect different values. Upon issuing the command, a text file containing the measurements is generated, and from this file is where all the data that will be discussed is extracted.

Since we are testing the architecture for a large number of cases, executing the commands that correspond to each case manually is quite tedious, so the process of changing the values and executing each command was done using scripts. Using Python and Bash scripts, the commands were executed, and the generated files were properly renamed to identify each case, and the relevant measurements were subsequently extracted from the output files.

The details of the scripts are not relevant to the simulation, so they will not be discussed here in detail. All the supporting scripts and output files are provided along with this report, and the flow of execution is apparent from the names of the scripts and the commands they contain. The figures shown in the following sections were generated using Python's matplotlib library.

L1 Data Cache Size

The size of L1 data cache was varied across the following values: 16KB, 32KB, 64KB, 128KB, and 256KB. The results obtained from each of the benchmarks are plotted in the figures below:



The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

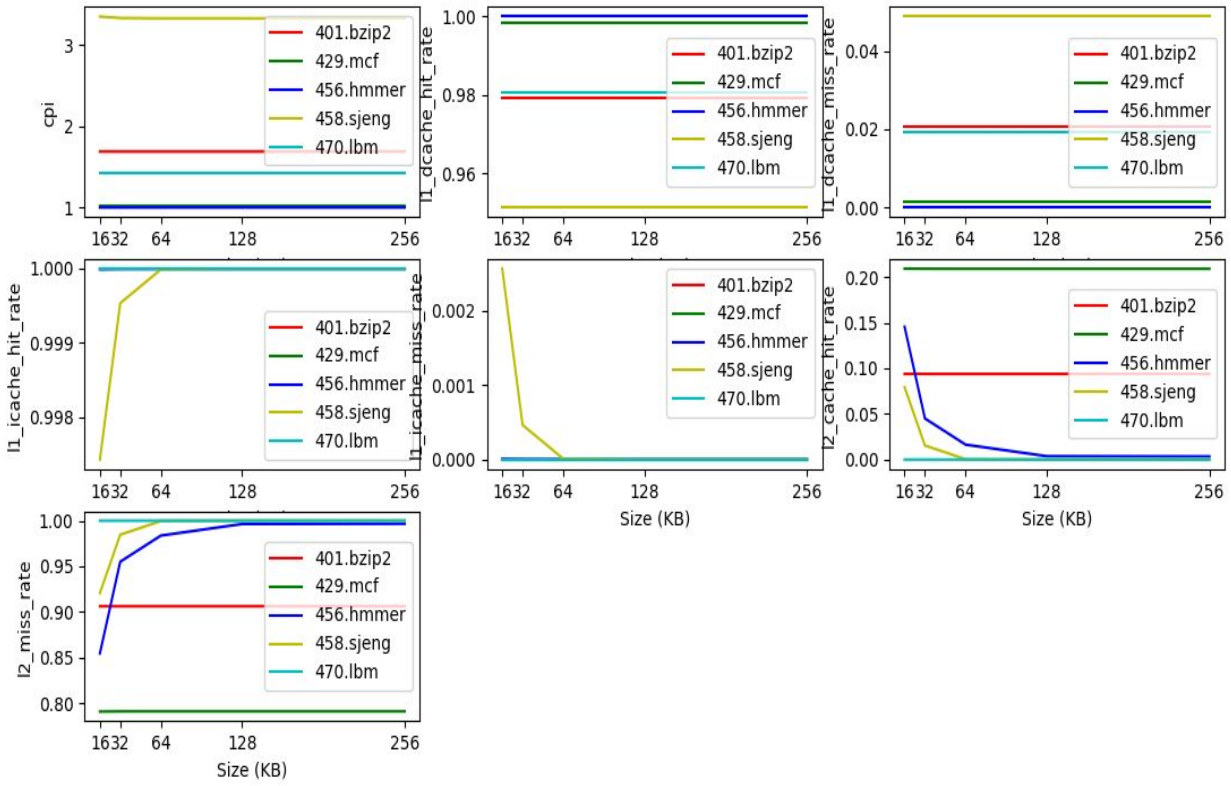
```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_l1d_cache_1
28kB configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/be
nchmark -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache
--l1d_size=128kB --l1i_size=256kB --l2_size=4MB --l1d_assoc=16 --l1i_assoc=16 --
l2_assoc=16 --cacheline_size=128
```

As the figures show, increasing the size of the data cache results in a direct improvement in the hit rates of the L1 data cache, as well as the CPI. This is expected, as the size of the cache determines how much data it can hold, and thus has a strong impact over the hit rate. The increase in the miss rate of the L2 cache can be attributed to the L1 cache having a better performance; it means that L1 does not have to access L2 as much, making the latter more prone to misses.

Aside from benchmark 401.bzip2, all the benchmarks show only a slight variation in the measured values as far as the L1 metrics are concerned, but they all follow the same trend.

L1 Instruction Cache Size

The instruction cache was varied across the same values as the data cache, and the results obtained from the simulation are plotted in the figures below:



The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_l1i_cache_1
28kB configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/be
nchmark -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache
--l1d_size=256kB --l1i_size=128kB --l2_size=4MB --l1d_assoc=16 --l1i_assoc=16 --
l2_assoc=16 --cacheline_size=128
```

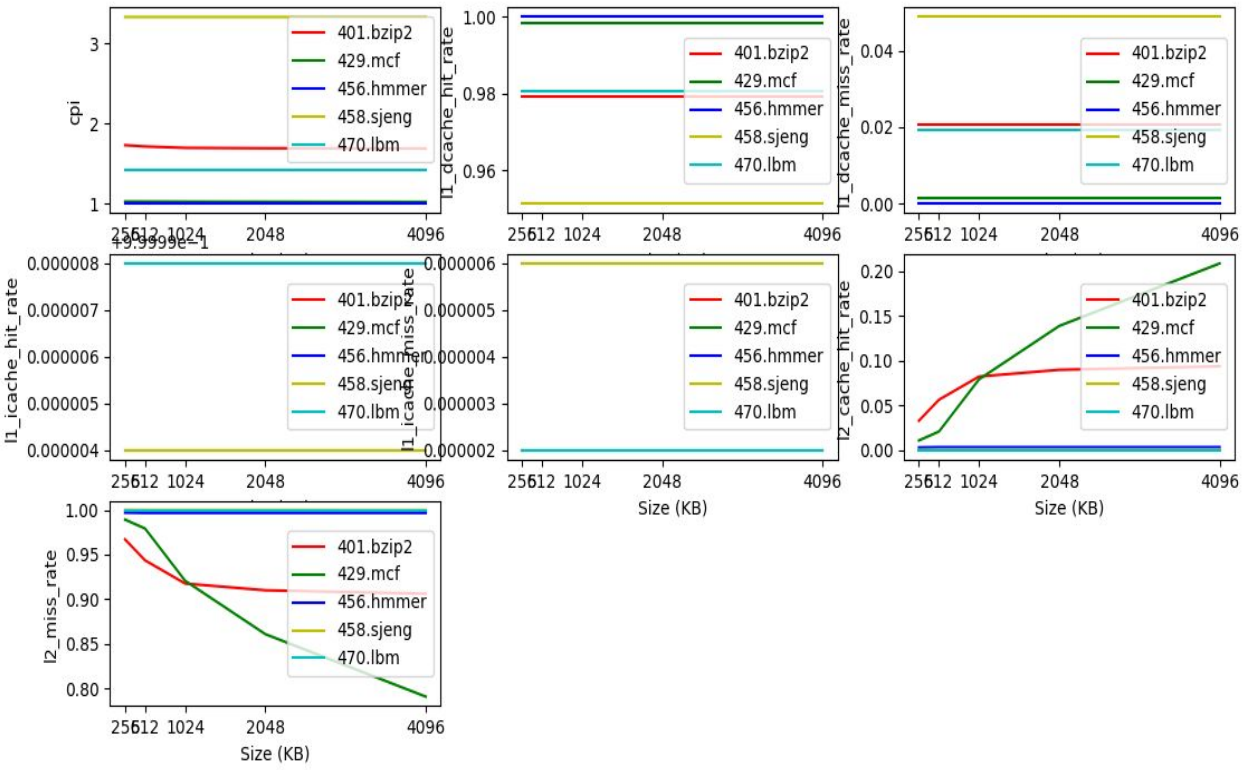
The figures show a small improvement in the hit rate when the size increases from 16KB to 64KB, but the hit rate and the CPI remain somewhat constant beyond 64KB. It can also be seen that the miss rate of the L2 cache also increases up to an L1 cache size of 64KB. This increase, much like the previous case where we increased the size of the L1 data cache, could also be attributed to the fact that

the L1 instruction cache does not need to access the L2 cache as much. The L2 cache, as a result, experiences a slight degradation in performance.

The CPI is not as affected as it was when the size of the L1 data cache was varied, which seems to suggest that the data cache has a more powerful effect on the CPI. This is probably because there are usually more data cache misses than instruction misses, which makes the former more determinant of the CPI, since CPI is derived from the number of misses for the three different types of caches.

L2 Cache Size

The size of the L2 cache was varied across the following values: 256KB, 512KB, 1MB, 2MB, and 4MB. The block size used in this case was 128B. The measurements obtained from the simulation follow:



The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_l2_cache_4M
B configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/bench
mark -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache --l
1d_size=256kB --l1i_size=256kB --l2_size=4MB --l1d_assoc=16 --l1i_assoc=16 --l2_
assoc=16 --cacheline_size=128
```

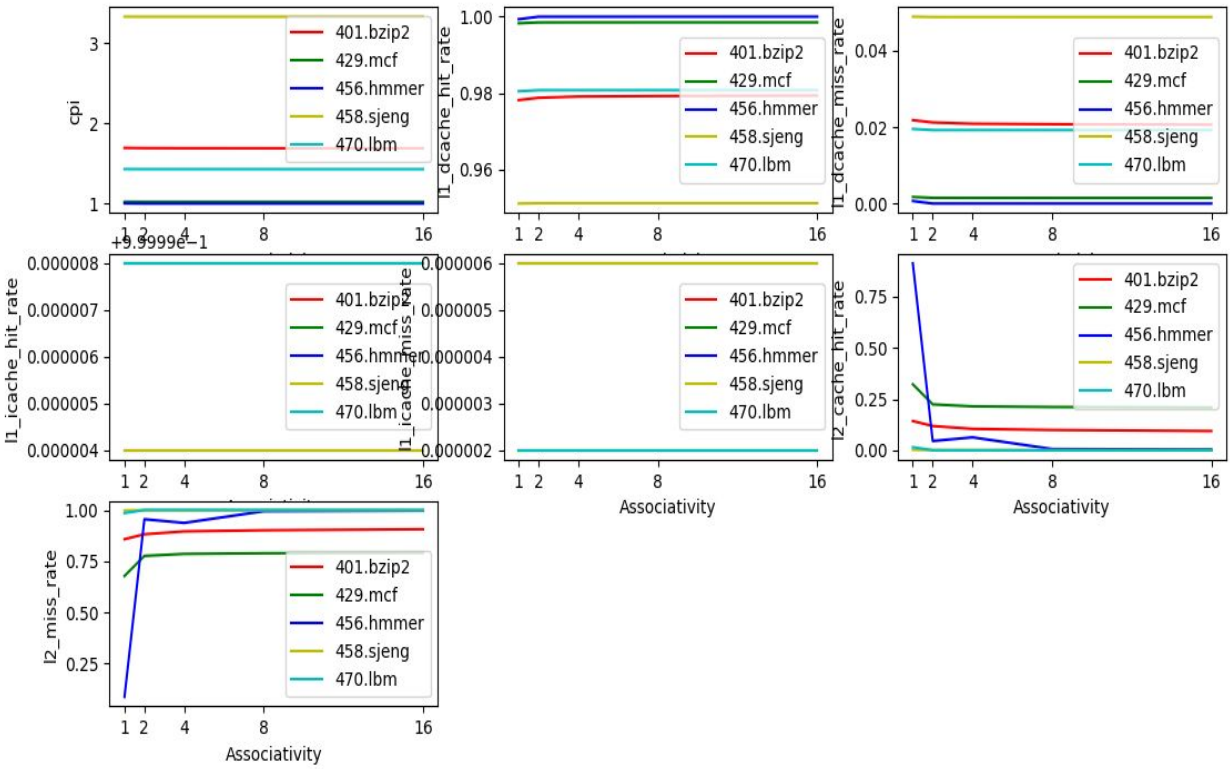
It is instantly obvious from the 429.mcf benchmark that the size of the L2 cache yielded a great improvement in the hit rate of the L2 cache. The CPI was not affected as much, but a slight decrease can be observed for the 401.bzip2 benchmark. As the plot of the results for the 429.mcf benchmark show, the performance of the L2 cache improves steadily up until the highest size that was considered, and may improve even more beyond a size of 4MB.

The L2 cache seems to have a bigger effect on the CPI than does the L1 instruction cache, and as much of an effect as the L1 data cache. Although there was an apparent increase in the hit rate of the L2 cache, the improvement only goes as far as making the hit rate climb up to nearly 0.2. This is a monumental improvement though, as the the hit rate was as low as 0.01 when the the size was kept at 256KB.

L1 Data Cache Associativity

The associativity of the L1 data cache was considered in the case of a direct mapped cache all the way up to 16-way associativity, with the increase being in multiples of two. As the graphs below illustrate, there is no great improvement beyond using a two-way associative cache. This seems to justify the popular choice of 2-way and 4-way associative caches used in commercial designs. The benchmarks do show a slight improvement in the hit rates and the CPI, but the improvement is negligible.

An interesting thing to note here is the degradation in performance of the L2 cache. All the benchmarks seem to agree that the miss rate for the L2 cache increases the most when the associativity of the L1 cache is increased from 1 to 2. This is a result of the direct improvement in the performance of the L1 cache, which, as we saw in the previous sections, tends to worsen the performance of the L2 cache, as it does not get accessed as much, meaning that it is more prone to misses.

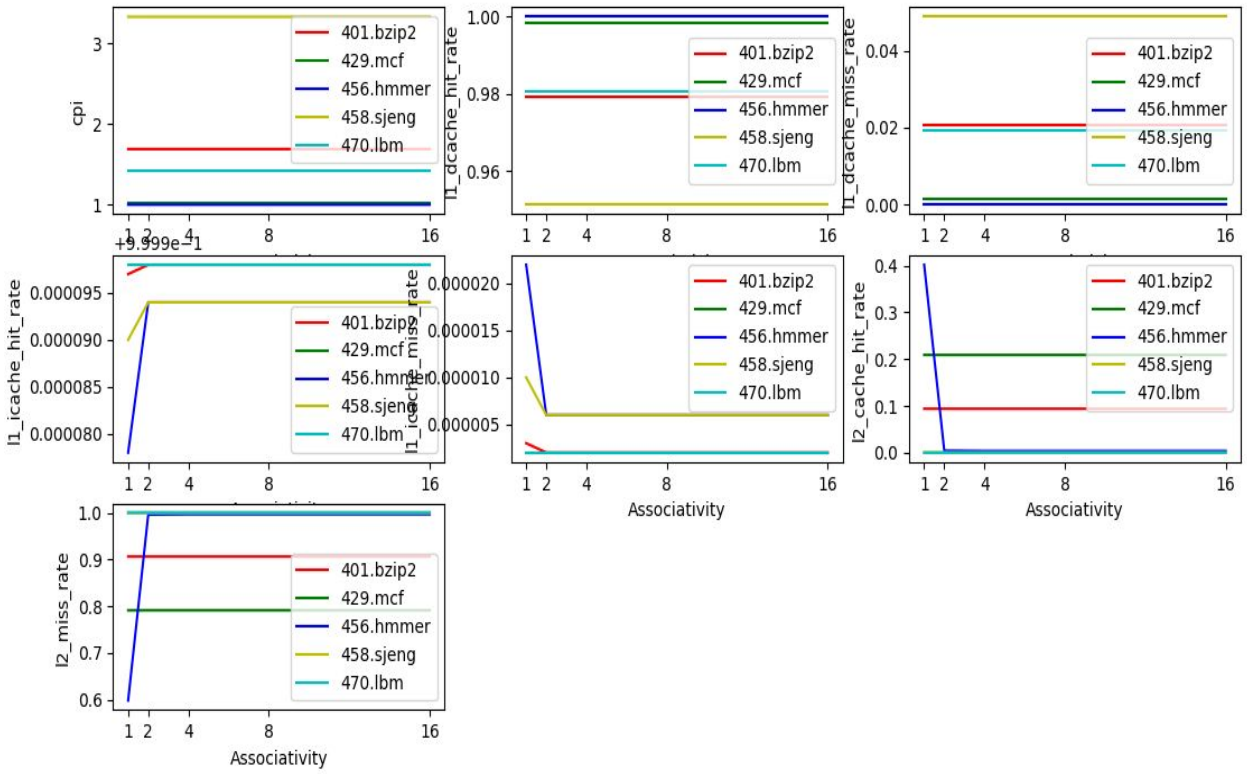


The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_l1d_assoc_2
 configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/benchm
ark -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache --l1
d_size=256kB --l1i_size=256kB --l2_size=4MB --l1d_assoc=2 --l1i_assoc=16 --l2_as
soc=16 --cacheline_size=128
```

L1 Instruction Cache Associativity

The associativity of the L1 instruction cache is considered next. As with the data cache, the greatest improvement can be seen in going from a direct mapped cache to a two-way associative cache. The L2 cache experiences the same drop in performance as it did when the parameters corresponding to the L1 cache were improved. These results further prove that L2 cache performance tends to decline as the miss rate of the L1 cache decreases.

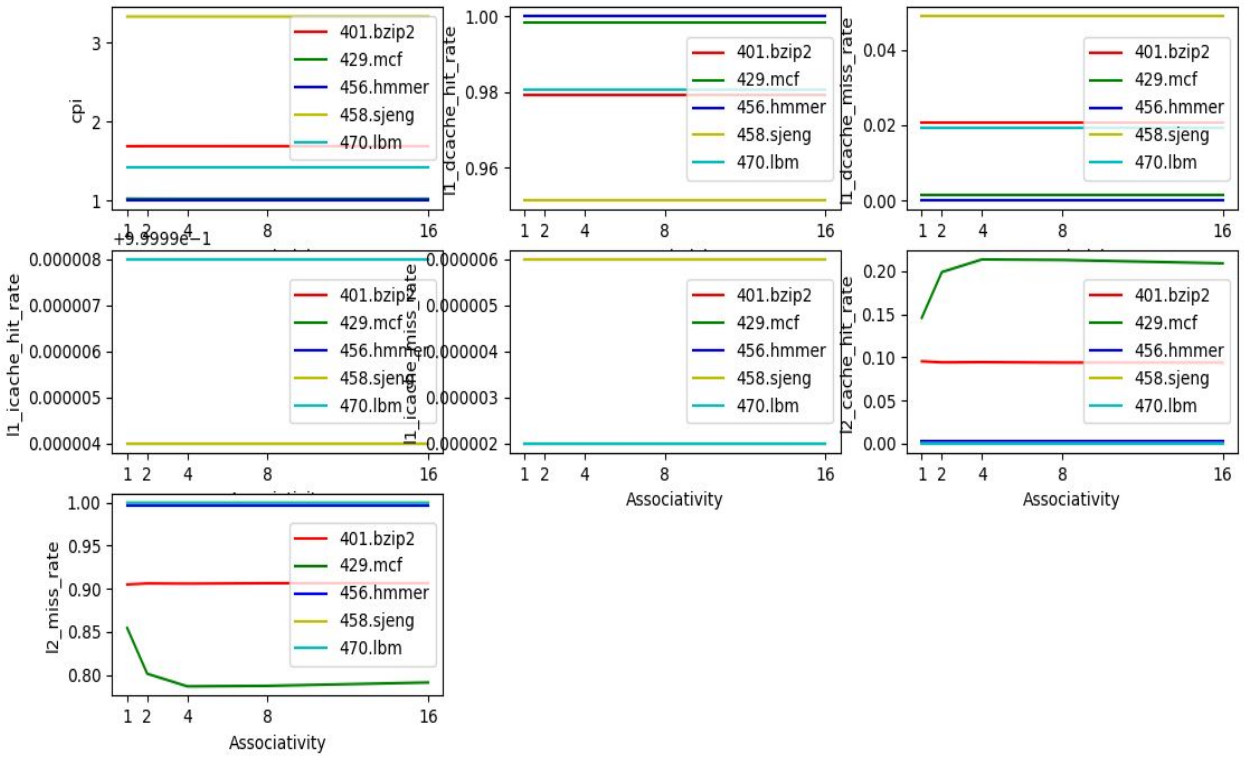


The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_l1i_assoc_2
 configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/benchm
ark -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache --l1
d_size=256kB --l1i_size=256kB --l2_size=4MB --l1d_assoc=16 --l1i_assoc=2 --l2_as
soc=16 --cacheline_size=128
```

L2 Cache Associativity

For the L2 cache, the numbers in the graph below suggest that the highest benefit can be yielded when the associativity is set to four. There does not seem to be a very significant improvement beyond this value. So we can see that the L2 cache is similar to the L1 cache in terms of improvement attributed to an increase in associativity. But upon comparing the graphs for L1 and L2, the latter does seem to benefit slightly more. This may be attributed to the fact that L2 is more affected by conflict misses. The following graph shows the difference in performance for different associativities for a block size of 128B:



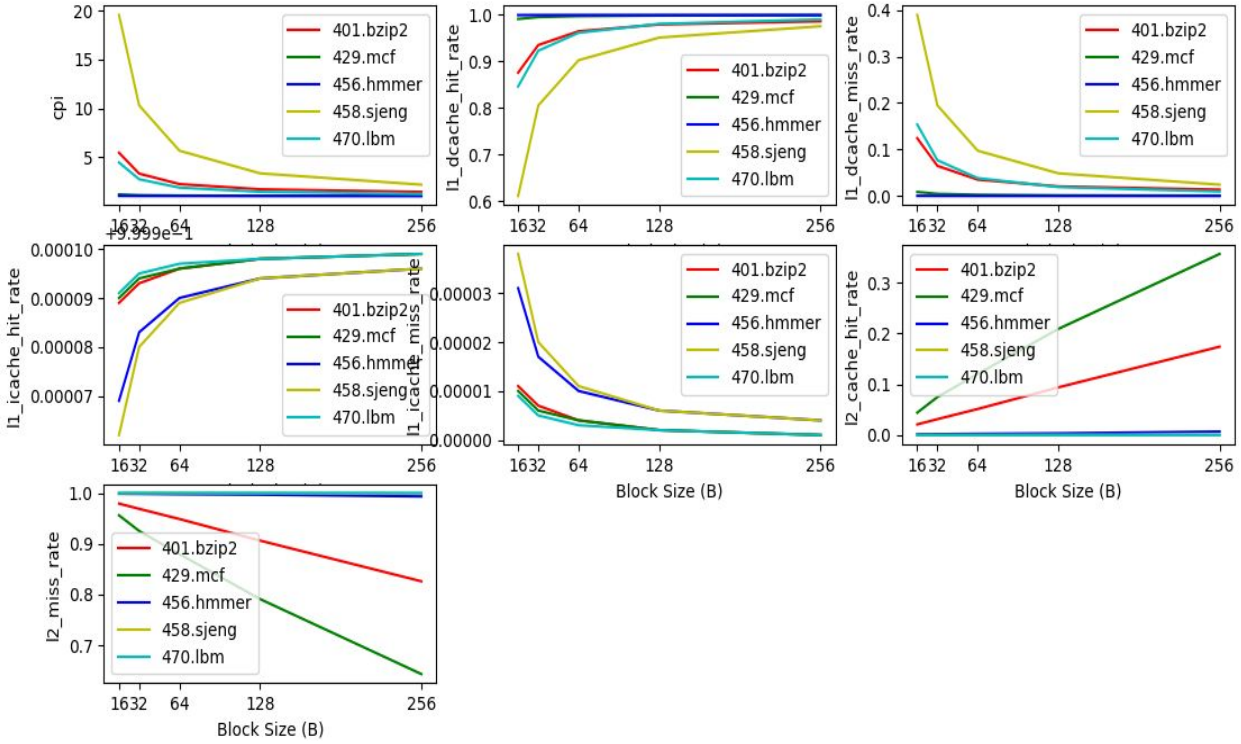
The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_l2_assoc_2
configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/benchma
rk -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache --l1d
_size=256kB --l1i_size=256kB --l2_size=4MB --l1d_assoc=16 --l1i_assoc=16 --l2_as
soc=2 --cacheline_size=128
```

Cache Block Size

The cache block size is what seems to have the greatest effect on the performance of the cache. The values for the block size were varied across the following values: 16B, 32B, 64B, 128B, and 256B. At 16B, it can be seen that the CPI is quite high, and the hit rate is not very good. As the size is increased, a great improvement can be seen for all the metrics up to 128B, then the improvement starts to decrease in effect. This is no surprise, as it is expected that the bigger the block size, the more data that can be fetched from the main memory for each trip, therefore less misses occur overall.

The figures below show the different trends obtained for the data cache for different choices of block size:



The command used to start the simulation for one of the benchmarks is shown below. Executions for different benchmarks were generated by only changing the name of the benchmark and the benchmark arguments.

```
command line: build/X86/gem5.opt -d /home/csgrad/ahamed/bin/sims/sim_blk_size_12
8 configs/example/se.py -I 100000000 -c /util/gem5/benchmark/401.bzip2/src/bench
mark -o /util/gem5/benchmark/401.bzip2/data/input.program --caches --l2cache --l
1d_size=256kB --l1i_size=256kB --l2_size=4MB --l1d_assoc=16 --l1i_assoc=16 --l2_
assoc=16 --cacheline_size=128
```

Conclusion

As we saw from the experiments, the different values that were used for the cache parameters reflected greatly on their performance. Some parameters, like the block size, had a significant impact on all the metrics, while others, like associativity only had an effect on one specific component. What's more, as we saw from the numbers yielded in the associativity experiments, increasing the parameters results in diminished returns after certain values. The gain in performance that comes from increasing the associativity from 1 to 2 is far more appreciable than the gain that comes from increasing it from 8 to 16. This also applies to all of the other parameters.