Cache performance evaluation for simulating configurations using SimpleScalar Simulator

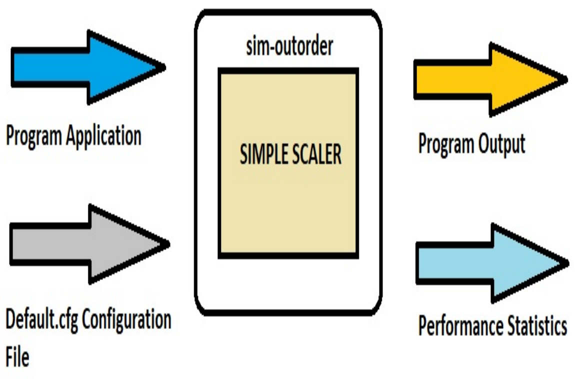
Author: Shashank Bhushan

Abstract - To evaluate the performance metrices of cache by keeping several basic cache configuration parameters 10 benchmarks from SPEC2000 suite were used to test different performance evaluation of the cache after simulation. The results provided a conclusion that by keeping block size greater than 32 had much better positive impact on cache performance during simulation runs.

Index Terms - SIMPLESCALAR “sim-outorder” model binaries, SPEC 2000 benchmark suit.

A. Introduction

An open-source program called SIMPLESCALAR v2.0 is used to mimic real programs in order to gather and assess performance data from simulated microarchitectures. Todd Austin created SIMPLESCALAR at the University of Wisconsin–Madison in its initial form. In a single program execution, SIMPLESCALAR creates cache statistics and profiles for various cache setups. Using default values for the instruction cache size, data cache size, associativity, etc., I am using the simulator to assess the system's performance. According to the benchmark, I discovered that increasing block size and associativity significantly changed the percentage of data cache hits and date cache misses, enhancing cache performance.



B. Memory System Simulators

A system having many tiers of instruction and data caches, each of which can be customized for different sizes and organizations, can be simulated by this simulator. If the impact of cache performance on execution time is not required, then this simulator is excellent for quick cache simulation. The variables that limit system performance are cache hits and cache misses. Cache hits are the number of cache accesses that successfully locate that data, while cache misses are those attempts that fail to do so. The average access time (AAT), also known as average memory access time (AMAT), which, as its name suggests, is the average time it takes to access memory, is influenced by these cache hits and misses.

Calculation of IPC

IPC is calculated by running a predetermined piece of code, figuring out how many machine-level instructions are needed to finish it, and then utilizing high-performance timers to figure out how many clock cycles are needed to finish it on the actual hardware. By dividing the total number of instructions by the total number of CPU clock cycles, the outcome is obtained. The cache will perform optimally when its IPC and frequency are high.

Load/store instructions

The CPU and the memory are both impacted by load and store instructions. Both load and stores must wait for an ALU, or address unit, to calculate their addresses before they can begin to execute. The requested memory data can be fetched by load instructions using the data cache, and it is then made available in a register. The data is then typically written into the designated architectural register to complete the load.

The way that store instructions are executed varies. Stores must wait for their operands to be available after getting their produced addresses.

In contrast to other instructions, a store is finished once operands are accessible.

If the necessary memory block is present in the primary data cache, these instructions are carried out rapidly; otherwise, there are lengthy access times to the secondary cache or main memory.

CPI (Cycles per Instruction)

The CPI statistic measures the average number of CPU cycles needed to complete an instruction, and as a result, it provides insight into how much system delay influenced the active program. Since CPI is a ratio, changes in the number of CPU cycles an application uses or the number of instructions it executes will have an impact. CPI is never greater than 1 (CPI 1). By approaching 1, one may obtain CPI numbers that are considerably higher.

Cache Hit

When a file is requested from a cache and the cache is able to provide the requested file, this is known as a "cache hit." The percentage of references that reach a level and produce hits is known as the level's hit rate.

Cache Miss

When the requested content is absent from the cache, this is known as a cache miss. Miss rate is the proportion of references that are sufficient to cause a miss.

C. Methodology

We alter the numeric values of the Instruction cache configuration and Data cache configuration in configuration files to assess the cache performance.

The values are in the format:

il1:512:32:1:I

Where,

il1: name (Instruction cache),

512: number of sets,

32: block size,

1: associativity

I: page replacement policy (Least Recently Used)

Internal changes

SIMPLESCALAR's initial configuration file was hardcoded into the simulator. The settings used to gauge performance were contained in the "default.cfg" file. Sim-out order is another name for an instruction-level simulator. In one simulation, which took around a minute to complete, I was able to carry out 100,000,000 instructions.

D. EXPERIMENTAL RESULTS

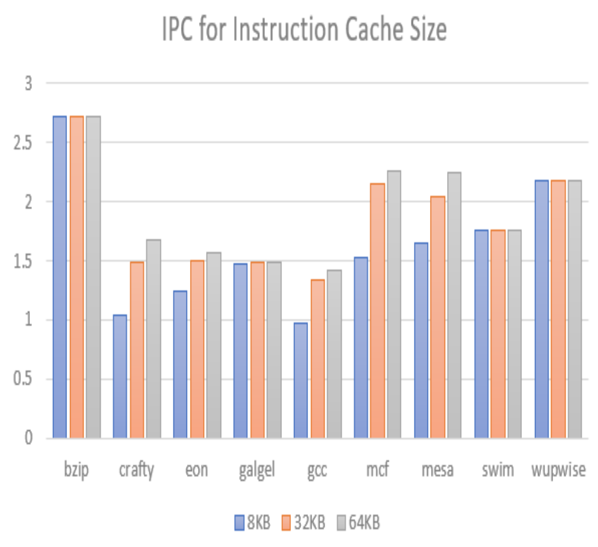
D.1 Experimental Setup

A component of our experimental setup was the SimpleScalar v3.0 simulator. Using the configuration files, which contained information about the number of sets, block size, associativity, and replacement policy, I examined the cache's performance. To execute each setup on nine distinct SPEC2000 benchmarks, I utilized a shell script. bzip, crafty, eon, gcc, mcf, mesa, swim, galgel, and wupwise were the benchmarks utilized. In each benchmark, I ran twenty separate simulations. I evaluated the effectiveness of load/store instructions, instructions per cycle, cache hits, miss rates, and cycles per instruction for each simulation.

D.2 Comparative Analysis for Instruction cache size (compare 8KB, 32 KB and 64KB, block size and associativity follow the default configuration)

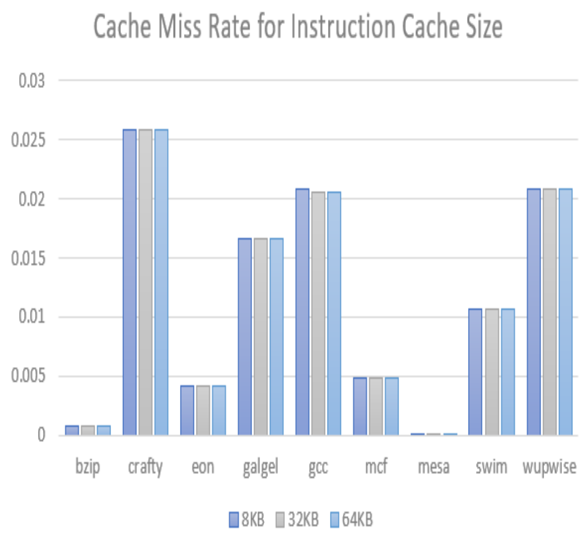
D.2.1 Instructions per Cycle (IPC)

IPC is highest for bzip benchmark and lowest for gcc benchmark.



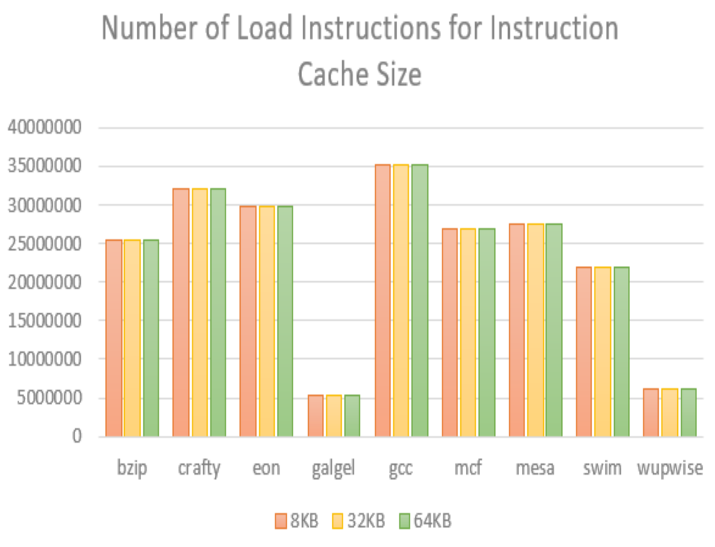
D.2.2 Cache Miss Rate

Cache Miss Rate is highest for crafty benchmark and lowest for mesa benchmark.



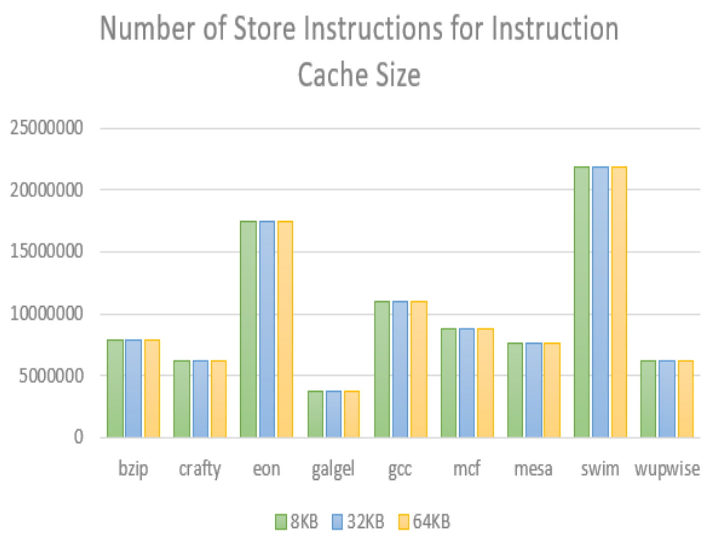
D.2.3 Number of Load Instructions

It is seen that Number of load instructions are highest for gcc benchmark and lowest for galgel benchmark.



D.2.4 Number of Store Instructions

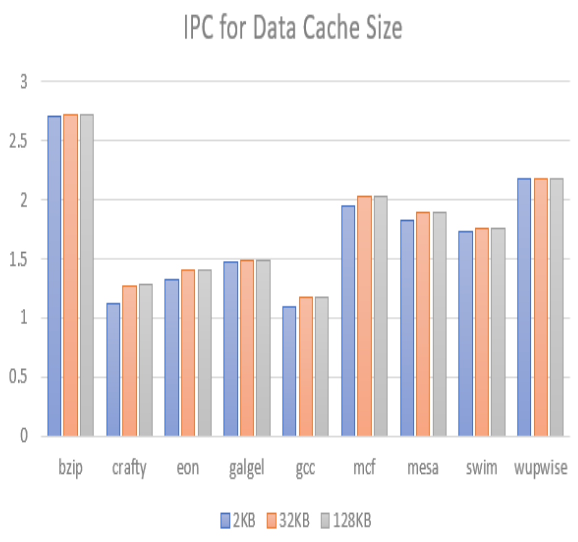
Number of store instructions are highest for eon benchmark and lowest for galgel benchmark.



D.3 Comparative Analysis for Data cache size (compare 2KB, 32KB, and 128KB, all 4-way set associative, block size follow the default configuration)

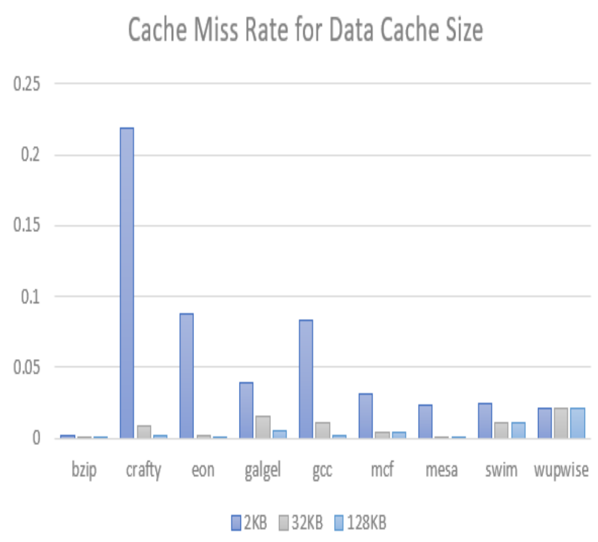
D.3.1 Instructions per Cycle (IPC)

IPC is highest for bzip benchmark and lowest for gcc benchmark.



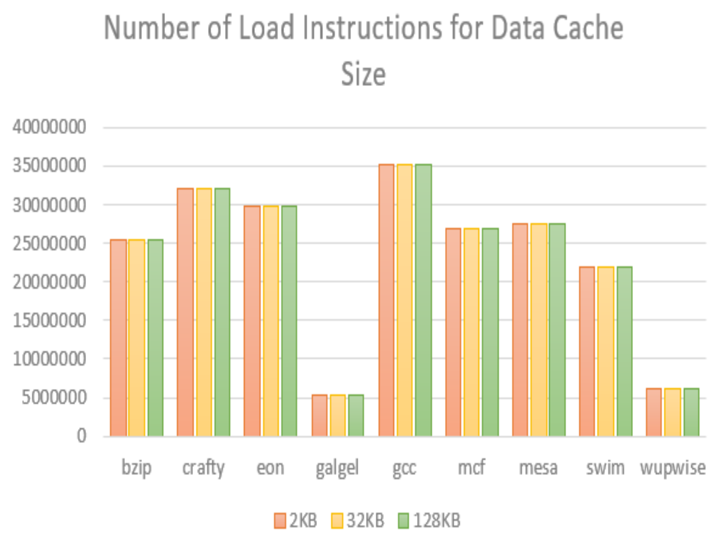
D.3.2 Cache Miss Rate

Cache Miss Rate is highest for crafty benchmark and lowest for bzip benchmark.



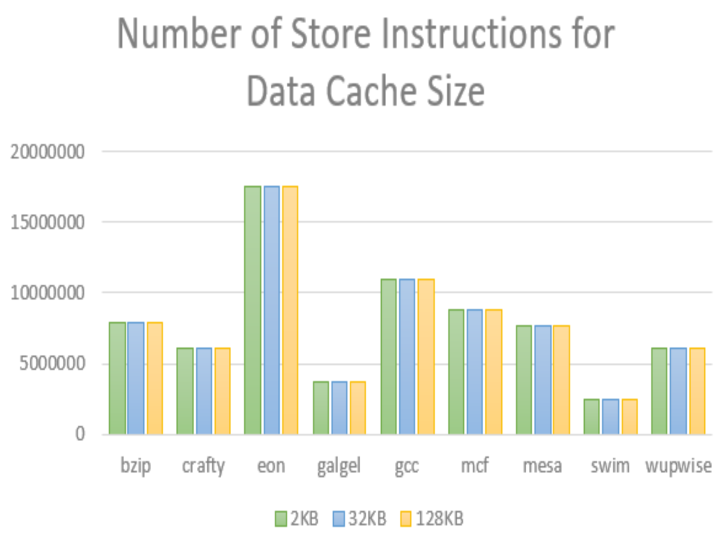
D.3.3 Number of Load Instructions

It is seen that Number of load instructions are highest for gcc benchmark and lowest for galgel benchmark.



D.3.4 Number of Store Instructions

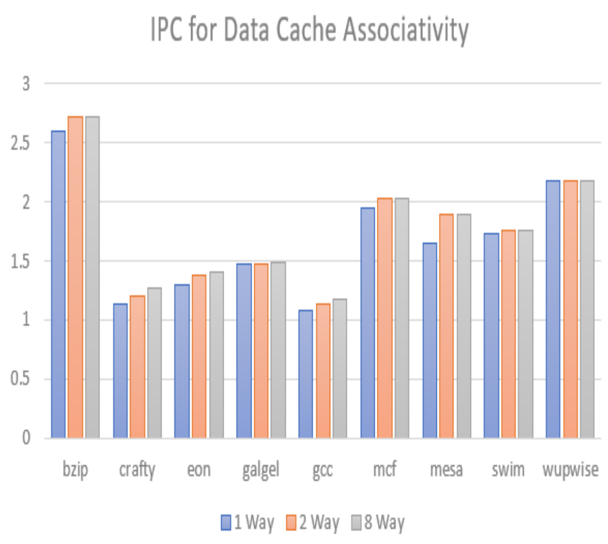
Number of store instructions are highest for eon benchmark and lowest for swim benchmark.



D.4 Comparative Analysis for Data cache associativity (compare 1-way, 2-way, 8-way, all 32KB, block size follow the default configuration)

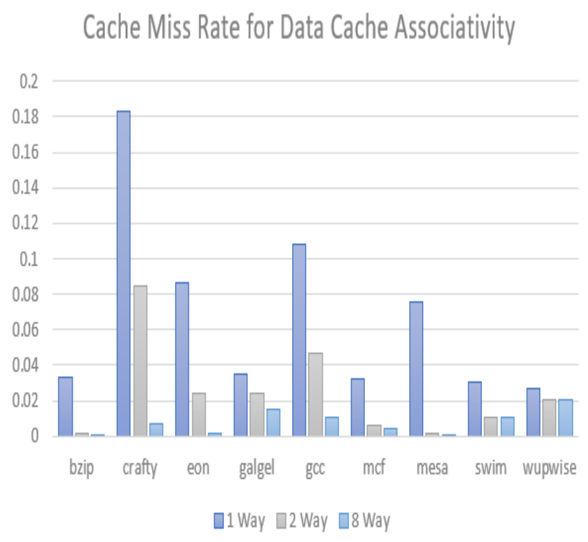
D.4.1 Instructions per Cycle (IPC)

IPC is highest for bzip benchmark and lowest for gcc benchmark.



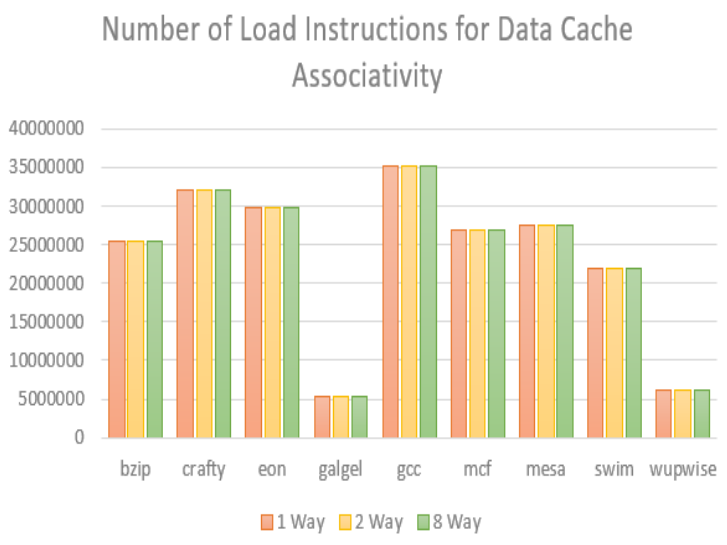
D.4.2 Cache Miss Rate

Cache Miss Rate is highest for crafty benchmark and lowest for mesa benchmark.



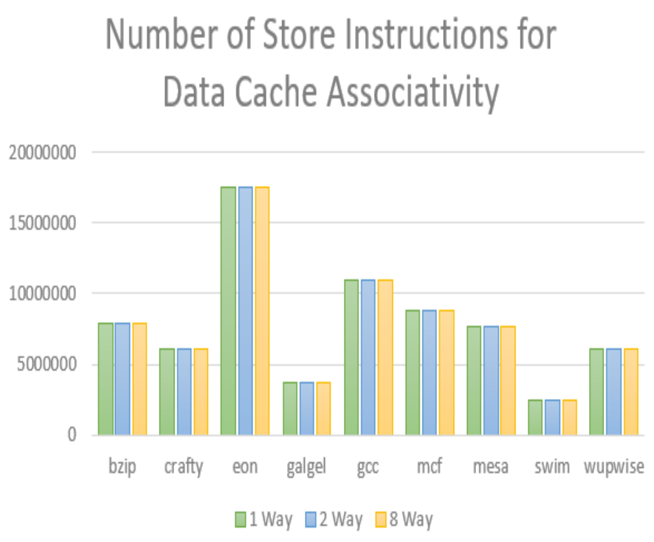
D.4.3 Number of Load Instructions

It is seen that Number of load instructions are highest for gcc benchmark and lowest for galgel benchmark.



D.4.4 Number of Store Instructions

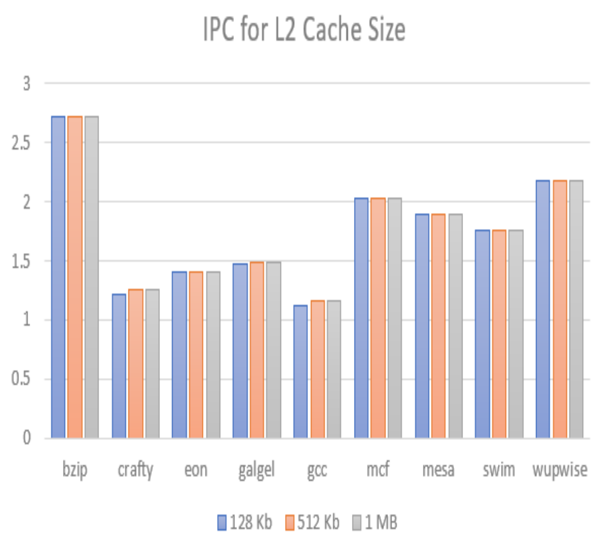
Number of store instructions are highest for eon benchmark and lowest for swim benchmark.



D.5 Comparative Analysis for L2 cache size (compare 128KB, 512KB, and 1MB, block size and associativity follow the default configuration)

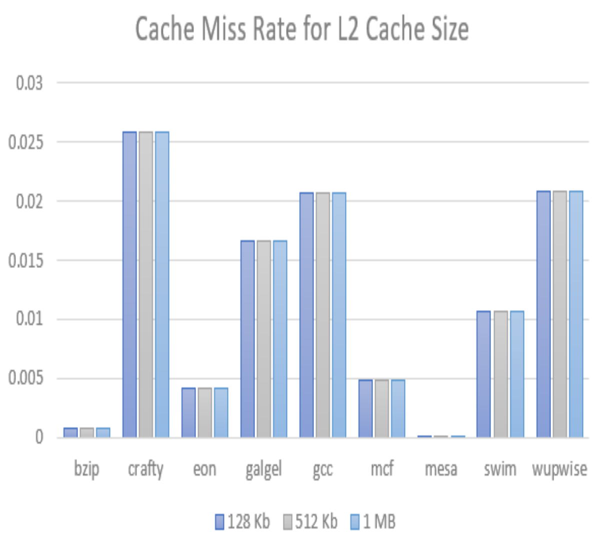
D.5.1 Instructions per Cycle (IPC)

IPC is highest for bzip benchmark with values around 2.7 and lowest for gcc benchmark with values around 1.3.



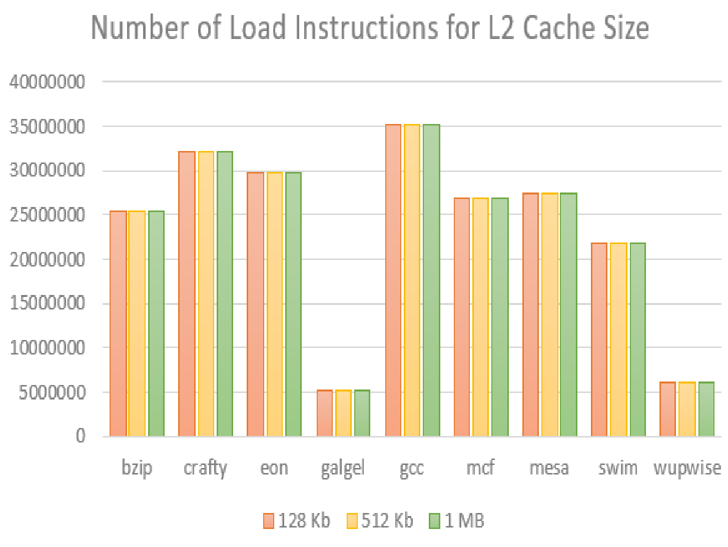
D.5.2 Cache Miss Rate

Cache Miss Rate is highest for crafty benchmark and lowest for mesa benchmark.



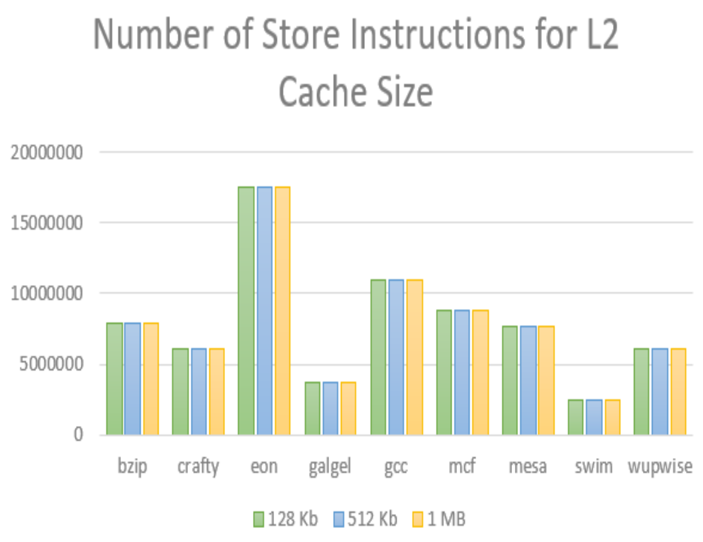
D.5.3 Number of Load Instructions

It is seen that Number of load instructions are highest for gcc benchmark and lowest for galgel benchmark.



D.5.4 Number of Store Instructions

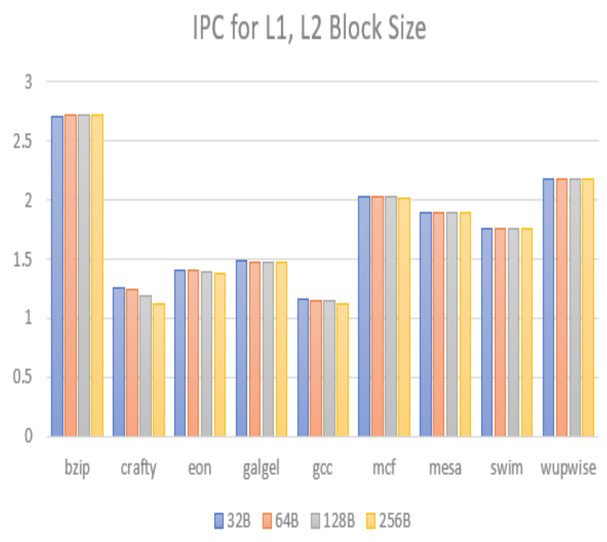
It is seen that Number of store instructions are highest for eon benchmark and lowest for swim benchmark.



D.6 Comparative Analysis for L1 and L2 block size (compare 32B, 64B, 128B, and 256B, cache size and associativity follow the default configuration)

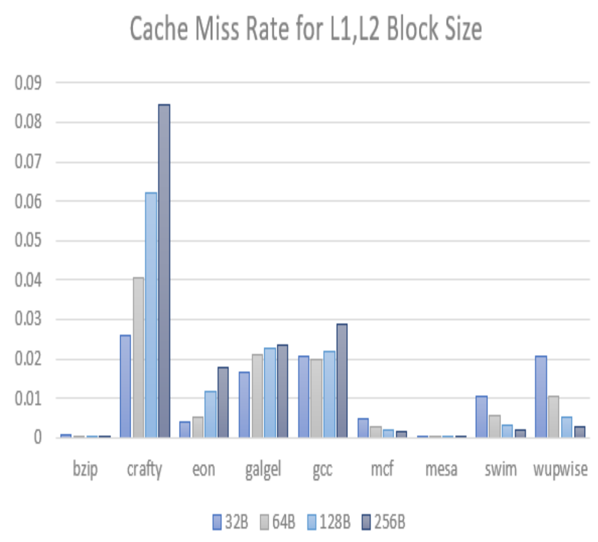
D.6.1 Instructions per Cycle (IPC)

IPC is highest for bzip benchmark with values around 2.6 and lowest for gcc benchmark with values around 1.25.



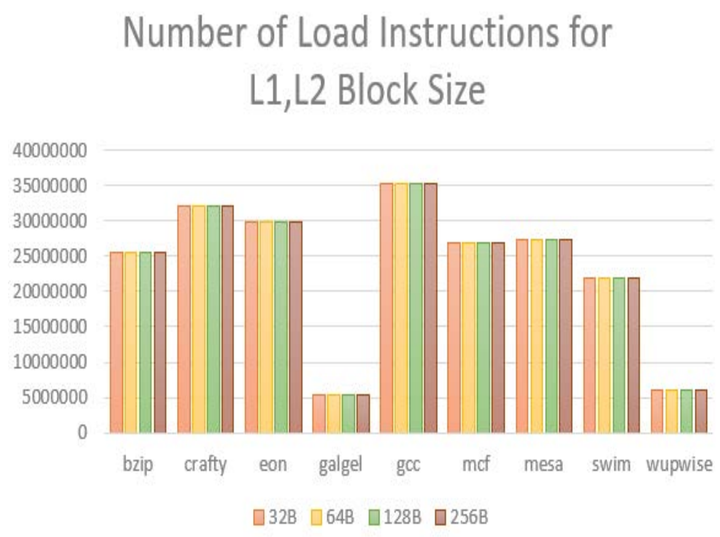
D.6.2 Cache Miss Rate

Cache Miss Rate is highest for crafty benchmark and lowest for mesa benchmark.



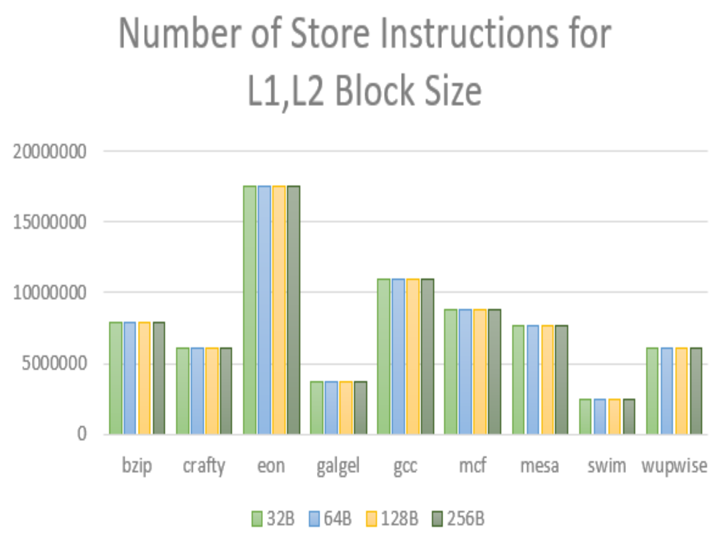
D.6.3 Number of Load Instructions

Number of load instructions are highest for gcc benchmark and lowest for galgel benchmark.



D.6.4 Number of Store Instructions

Number of store instructions are highest for eon benchmark and lowest for swim benchmark.



E. Conclusion

As part of this project's work, we were able to compile and analyse data that helped us understand changes in IPC, Miss Rate, and the number of load and store instructions that occur when different parameters, such as cache size of data and instructions, associativity of sets, and block size, are changed.