****

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**EEL5741-RXPE-1201: Advanced Microprocessor Systems**

**SPRING 2020**

**Benchmark Profiling and Processor Architecture**

**Simulation Project 2**

**EEL5741\_4**

Elvis Fernandez 5866280 efern207@fiu.edu

Kevin Lopez 4054391 klope108@fiu.edu

Narek Vanestyan 6227578 nvane007@fiu.edu

**Submitted to:**

**Professor: Dr. Gustavo Chaparro**

**March 23rd, 2020**

## Table of Contents

[**Table of Contents**](#_5ip77yplas5l) **2**

[**Simulation Instructions**](#_bz9s6w4y7ii7) **3**

[**Introduction**](#_gt8kvgms9wbx) **5**

[**Simulation Solution**](#_pezjkoiwyz4u) **6**

[Part 1: Understanding “sim-cache”](#_ne8zcarw9sy4) 6

[Part 2: Generating Matmul.ss Executable with SimpleScalar](#_53j4xah5si30) 8

[Part 3: Utilizing Sim-cache Simulator to Analyze Cache Performance](#_31c4o32d2w9m) 9

[Part 3\_a: Cache/Block size](#_3tjhb1dq7bpe) 9

[Output files from the Simulation can be found here:](#_mj4347lzwiba) 12

[Part 3\_b: Sim-cache simulation (Cache Associativity).](#_tkegw880jf51) 13

[Output files from the Simulation can be found here:](#_xiwp9c6bm0gx) 15

[Part 3\_c: Unified vs Split Cache](#_xi9u8egxi8ut) 16

[Output files from the Simulation can be found here:](#_1x5ddn8v40ek) 17

[Part 3\_d : Sim-Cache simulation (Replacement Policy)](#_l9saiopf6n30) 18

[Output files from the Simulation can be found here:](#_e1djo7xva85l) 21

[Part 3\_e: Multilevel Cache](#_o6mwbropu9ny) 22

[Output files from the Simulation can be found here:](#_a251uv7j46lu) 23

[Part 3\_f: Three Types of Cache Misses](#_wtzcyahv33ef) 24

[Output files from the Simulation can be found here:](#_obxl8v744v6p) 26

[Part 3\_g: TLB](#_yoz6oio3m0v6) 27

[Output files from the Simulation can be found here:](#_ate2vuei2erz) 28

[**Simulation Summary**](#_6cjwz1yp6ubq) **29**

[**Reference**](#_aqil22ng8r99) **30**

## Simulation Instructions

**Project 2: Memory Hierarchy Performance Analysis**

**Introduction**

In this project, we will use SimpleScalar simulation tools to study the memory hierarchy performance with memory hierarchy organization. There are three tools in SimpleScalar that can be used for this purpose: sim-cache, sim-cheetah, and sim-outorder. Sim-cache and sim-cheetah are for the in-order execution, while sim-outorder is for the out-of-order simulation. In this project, we use sim-cache as it is good enough for this project. For more detailed information on the usage of sim-cache, you can refer to SimpleScalar Tools document (<http://www.simplescalar.com/docs/users_guide_v2.pdf> ).

**What to do**

1. Read <http://www.simplescalar.com/docs/users_guide_v2.pdf> about sim-cache.
2. Compile the matrix multiplication program matmul.c into an executable using gcc and test it using simple examples to make sure the program works correctly. Then remove the output section in your program and recompile it (without using any optimization option) use SimpleScalar compiler.
3. Cache Analysis Assume 50 by 50 matrices for following simulations. If the cache specifications are not given, use the default values set in the SimpleScalar.
   1. Cache/block size Simulate matmul.c with different data cache and block size. Assume a direct mapped cache. Change the cache sizes as 2K, 4K,8K, 16K, 32K, 64K each with different block sizes: 16, 32, 64, 128, 256. Compare and plot the data cache miss rates. (Note: when you set the block size, the block size for the first level must be no larger than that for the lower level.)
   2. Cache associativity Simulate matmul.c with different set associativities. Assume one level data cache with 8K size. Change the degrees of data cache to be 1, 2, 4, 8, 16. Compare and plot the data cache miss rates.
   3. Unified v.s. Split cache Simulate matmul.c with both unified and split cache. Assume one level and 2-way set associative cache with total sizes of 4k, 8k,16k, 32k, 64k, 128k and block size of 32. Compare and plot the cache miss rates for instruction and data memory access.
   4. Block replacement policy Simulate matmul.c with three different block replacement policies. Assume one level data cache with total cache sizes as 4K and 32K, each of which has degree of set associativity as 1, 4, and 32. Compare and plot the data cache miss rates.
   5. Multilevel Cache Simulate the execution of matmul.c with two levels of data cache. Let the first level cache be 8K direct mapped cache, the second level be 2 way set associative caches with sizes as 8k,16k, 32k, 64k,128k. Compare and plot the local as well as the global miss rate for the second level data cache.
   6. Three types of cache misses Simulate the execution of matmul.c with one level of data cache with sizes as 8k,16k, 32k, 64k, 128k and associativity 1,2, 4, 8. Compare and plot the cache miss rate for the capacity and conflict cache misses.
   7. TLB Simulate matmul.c with two levels of data cache. Let the first level cache be 1K direct mapped cache, the second level be 2-way set associative caches with size of 32k. Set data TLB to be directed, 2-way, and fully associative with 4 and 8 entries (Let the page size be 1K). Compare and plot the data TLB misses. Approximately how many pages needs to be accessed for the data?

**What to hand in**

Use tables and figures (using MS Excel for example) to present your experimental results and attach your discussion of these results in ONE pdf file. Again, a summary section is mandatory for all the students to summarize the conclusions and/or any thoughts, experience you get from the projects.

**Reference**

[1] Todd Austin and Doug Burger, SimpleScalar Tutorial, (for tool set release 2.0);

[2] Todd M. Austin, A User’s and Hacker’s Guide to the SimpleScalar Architectural

Research (for tool set release 2.0);

[3] A. LaMarca and R. E. Ladner. "The Influence of Caches on the Performance of

Sorting." Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete

Algorithms, 1997. pp. 370-379. <http://citeseer.ist.psu.edu/article/lamarca96influence.html>

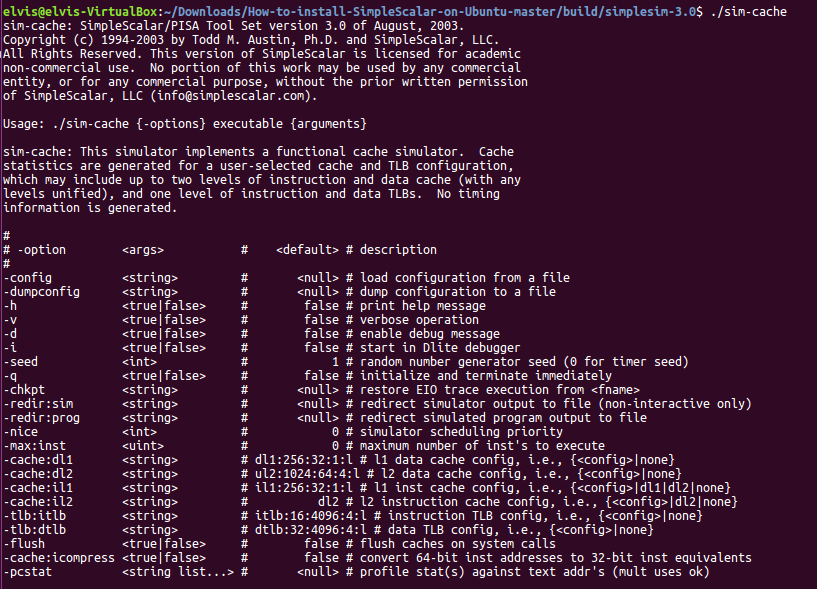
## Introduction

The goal of this simulation project is to use the SimpleScalar simulation tools to study the memory hierarchy performance with memory hierarchy organization. Specifically, we will use sim-cache to profile L1, L2 and TLB cache. Cache performance is influenced by several factors, including cache and block size, cache associativity, replacements policies, the type of data they reference (data or instructions), how many cache levels are implemented within the memory hierarchy, and others. The team will be simulating different scenarios where such factors are varied to understand their impact in a L1, L2 and TLB cache memory and will draw conclusions from the data collected.

## Simulation Solution

### Part 1: Understanding “***sim-cache***”

During [Simulation\_Project\_1](https://drive.google.com/drive/u/0/folders/1Rw1mV0_IEwbVGx4WGTktjORTMBF61AE8) we covered basic understanding of SimpleScalar, github repositories, installation procedures, and profiling simulators commands. Another functional simulation command included within SimpleScalar is sim-cache. Sim-cache is a fast simulator of caches ideal for situations where cache performance on execution time is not needed. Figure 1 shows sim-cache command options.



**Fig.1. Sim-cache command options within SimpleScalar.**

The sim-cache {-options} configuration format is shown in figure 2. If no option is specified the commands defaults to:

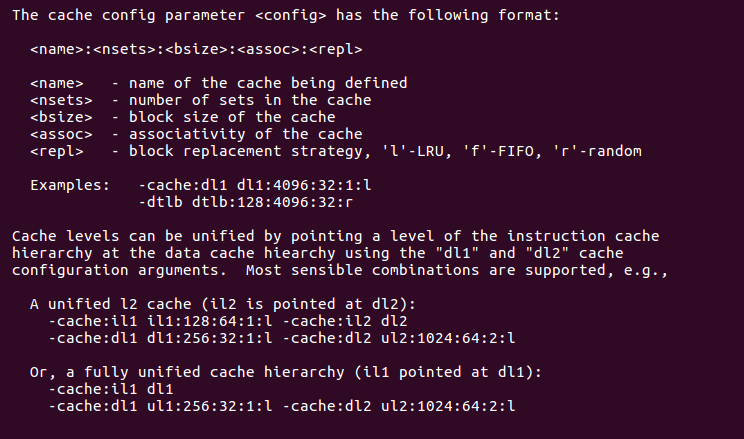
**L1 instruction cache: il1:256:32:1:l (8 KB)**

**L1 data cache: dl1:256:32:1:l (8 KB)**

**L2 uniﬁed cache: ul2:1024:64:4:l (256 KB)**

**instruction TLB: itlb:16:4096:4:l (64 entries)**

**data TLB: dtlb:32:4096:4:l (128 entries)**



**Fig. 2. The cache configuration parameter <config> format**:

Note that if more than one option is to be modified, it will need to be sent in the same command line separated with space. For example:

**./sim-cahe -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l *executable <arguments>***

### Part 2: Generating Matmul.ss Executable with SimpleScalar

To cross compile matmul.c with SimpleScalar:

1. Download/save matmul.c into simplesim-3.0.
2. Then you can send the following commands to generate optimized and non-optimized SimpleScalar binary.ss files:
   1. For Optimized matmu02.ss file:
      1. elvis@elvis-VirtualBox:~/Downloads/How-to-install-SimpleScalar-on-Ubuntu-master/build/simplesim-3.0$ sslittle-na-sstrix-gcc -O2 -x c matmul.c -o matmul02.ss
         1. (A matmul02.ss file will be created at the root dir)
   2. Non-optimized matmul.ss file:
      1. elvis@elvis-VirtualBox:~/Downloads/How-to-install-SimpleScalar-on-Ubuntu-master/build/simplesim-3.0$ sslittle-na-sstrix-gcc -x c matmul.c -o matmul.ss
         1. (A matmul.ss file will be created at the root dir)

### Part 3: Utilizing Sim-cache Simulator to Analyze Cache Performance

#### Part 3\_a: Cache/Block size

This part is looking for the miss rates of various configurations of cache. We run the sim-cache simulation and change the dl1 (l1 data cache) config parameters for 30 different scenarios. We have six cache sizes and each of them will be run using five different block sizes. The cache config parameter has the following format:

<name>:<nsets>:<bsize>:<assoc>:<repl>

Where nsets = (cache size / block size),  
bsize = block size,  
assoc = associativity,  
repl = replacement policy.

As cache size and block size are the only things we are changing, we will be using default values for everything else. However, the instructions for the project say that “the block size for the first level must be no larger than that for the lower level”. This means we also have to change the dl2 (l2 data cache) config parameters so that instead of using the default value of 64 for the block size, we will need to use 256 as that value will satisfy the above requirement from the project instructions. Everything else is unchanged.

The first command we run is for 2k cache size and 16B block size:

./sim-cache -cache:dl1 dl1:128:16:1:l -cache:dl2 ul2:1024:256:4:l matmulO0.ss 50 50 50

The last command we run is for 64k cache size and 256B block size:

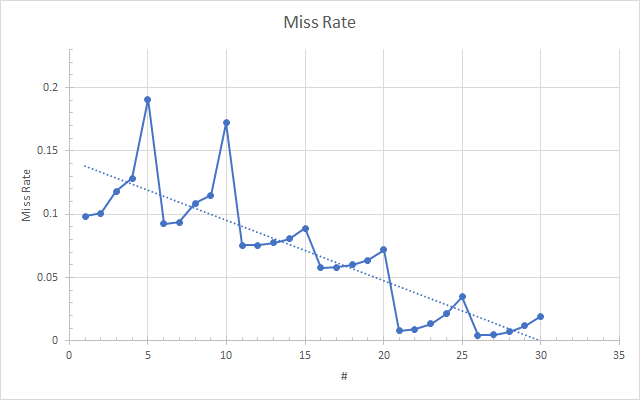
./sim-cache -cache:dl1 dl1:256:256:1:l -cache:dl2 ul2:1024:256:4:l matmulO0.ss 50 50 50

See the results in Table 1 below for all 30 scenarios:

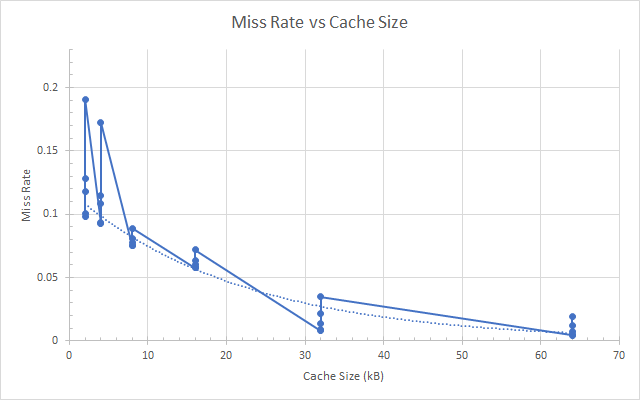
**Table 1. Results for 30 scenarios**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Command** | **Cache size (kB)** | **# of Sets** | **Block Size (B)** | **Miss Rate** |
| **1** | dl1:128:16:1:l | 2 | 128 | 16 | 0.0983 |
| **2** | dl1:64:32:1:l | 2 | 64 | 32 | 0.1006 |
| **3** | dl1:32:64:1:l | 2 | 32 | 64 | 0.1181 |
| **4** | dl1:16:128:1:l | 2 | 16 | 128 | 0.1283 |
| **5** | dl1:8:256:1:l | 2 | 8 | 256 | 0.1905 |
| **6** | dl1:256:16:1:l | 4 | 256 | 16 | 0.0925 |
| **7** | dl1:128:32:1:l | 4 | 128 | 32 | 0.0935 |
| **8** | dl1:64:64:1:l | 4 | 64 | 64 | 0.1084 |
| **9** | dl1:32:128:1:l | 4 | 32 | 128 | 0.1149 |
| **10** | dl1:16:256:1:l | 4 | 16 | 256 | 0.1721 |
| **11** | dl1:512:16:1:l | 8 | 512 | 16 | 0.0754 |
| **12** | dl1:256:32:1:l | 8 | 256 | 32 | 0.0756 |
| **13** | dl1:128:64:1:l | 8 | 128 | 64 | 0.0774 |
| **14** | dl1:64:128:1:l | 8 | 64 | 128 | 0.0807 |
| **15** | dl1:32:256:1:l | 8 | 32 | 256 | 0.089 |
| **16** | dl1:1024:16:1:l | 16 | 1024 | 16 | 0.0575 |
| **17** | dl1:512:32:1:l | 16 | 512 | 32 | 0.0579 |
| **18** | dl1:256:64:1:l | 16 | 256 | 64 | 0.06 |
| **19** | dl1:128:128:1:l | 16 | 128 | 128 | 0.0637 |
| **20** | dl1:64:256:1:l | 16 | 64 | 256 | 0.072 |
| **21** | dl1:2048:16:1:l | 32 | 2048 | 16 | 0.0081 |
| **22** | dl1:1024:32:1:l | 32 | 1024 | 32 | 0.0093 |
| **23** | dl1:512:64:1:l | 32 | 512 | 64 | 0.0133 |
| **24** | dl1:256:128:1:l | 32 | 256 | 128 | 0.0215 |
| **25** | dl1:128:256:1:l | 32 | 128 | 256 | 0.0349 |
| **26** | dl1:4096:16:1:l | 64 | 4096 | 16 | 0.0044 |
| **27** | dl1:2048:32:1:l | 64 | 2048 | 32 | 0.0049 |
| **28** | dl1:1024:64:1:l | 64 | 1024 | 64 | 0.0072 |
| **29** | dl1:512:128:1:l | 64 | 512 | 128 | 0.0118 |
| **30** | dl1:256:256:1:l | 64 | 256 | 256 | 0.0195 |

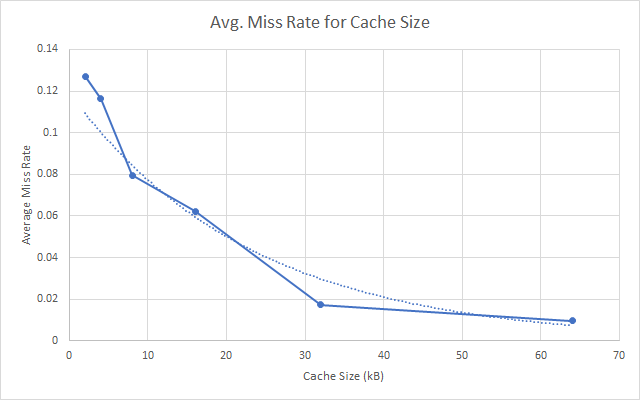
Figure 3,4 and 5 are graphs for Miss Rate. Figure 3 has a linear trendline, while Figure 4 and 5 have exponential trendlines. This is done as the data points themselves behave in a linear and exponential manner, respectively.



**Fig. 3.Miss Rate with Linear trendline**.



**Fig. 4.Miss Rate with exponential trendline**.



**Fig. 5. Miss Rate with exponential trendline**.

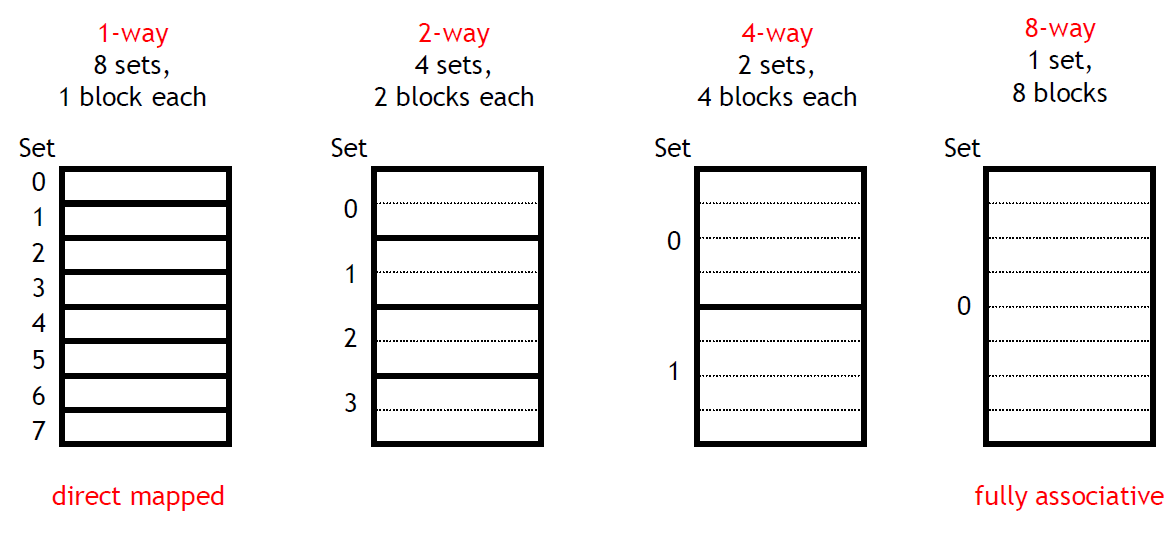
We can see that the Miss Rate seems to depend quite a bit on the Cache Size. The third graph shows the average Miss Rate in each Cache Size category (i.e. the Average Miss Rate for the same Cache Size with the five different block sizes explained at the beginning of this part). One observation we can make is that the Average Miss Rate drops steeply between 2k and 8k Cache Sizes, then drops at a much lower rate between 8k and 32k, then after 32k it virtually levels off and the higher cache size yields very little benefit.

#### Output files from the Simulation can be found here:

<https://drive.google.com/drive/u/0/folders/1i-MhlK1VjocNDGKyBRdDepwTgj3rR9d0>

#### 

#### Part 3\_b: Sim-cache simulation (Cache Associativity).

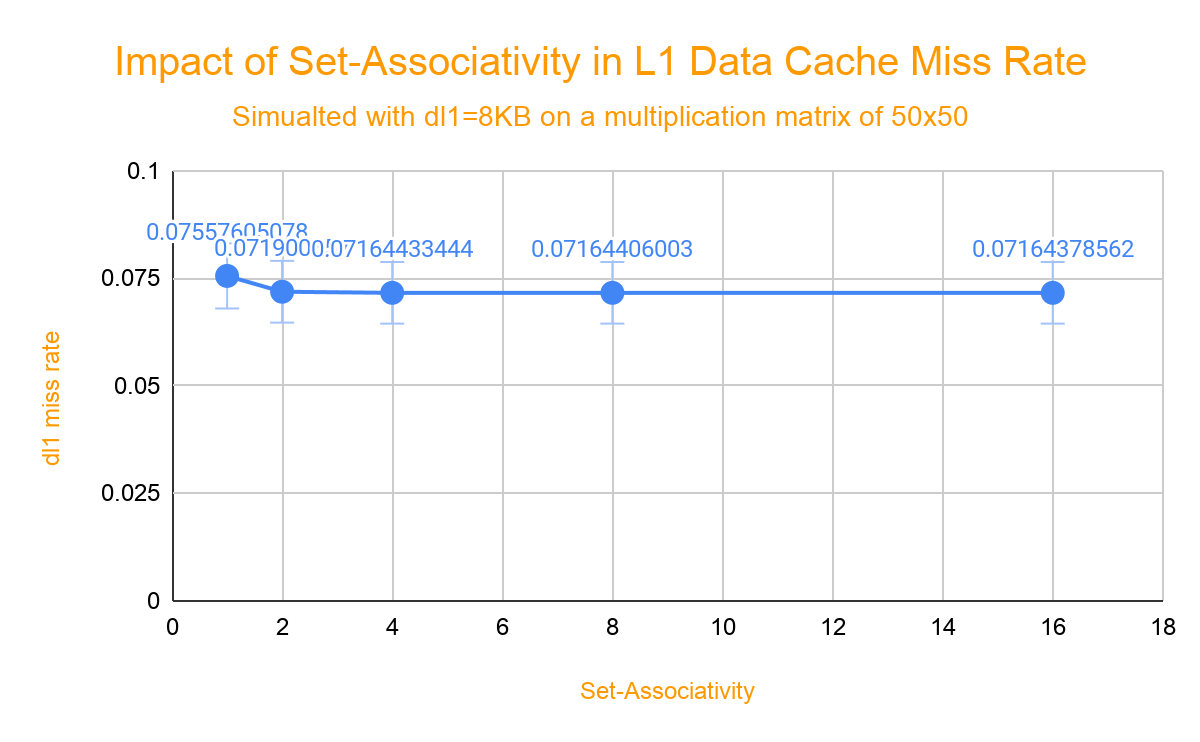
Figure 6 shows a representation of cache associativity. Cache is divided into groups of blocks called *sets*. Each memory address is mapped to exactly one set in cache, and data can be placed in any block within a set. If each set has blocks, the cache is an -way associative cache. Note that 1-way associativity corresponds to a direct mapped cache, where 1 block corresponds to a set.

**Fig. 6. Explaining cache associativity. 1 way( direct mapped) to the left & 8 way (fully associativity)**

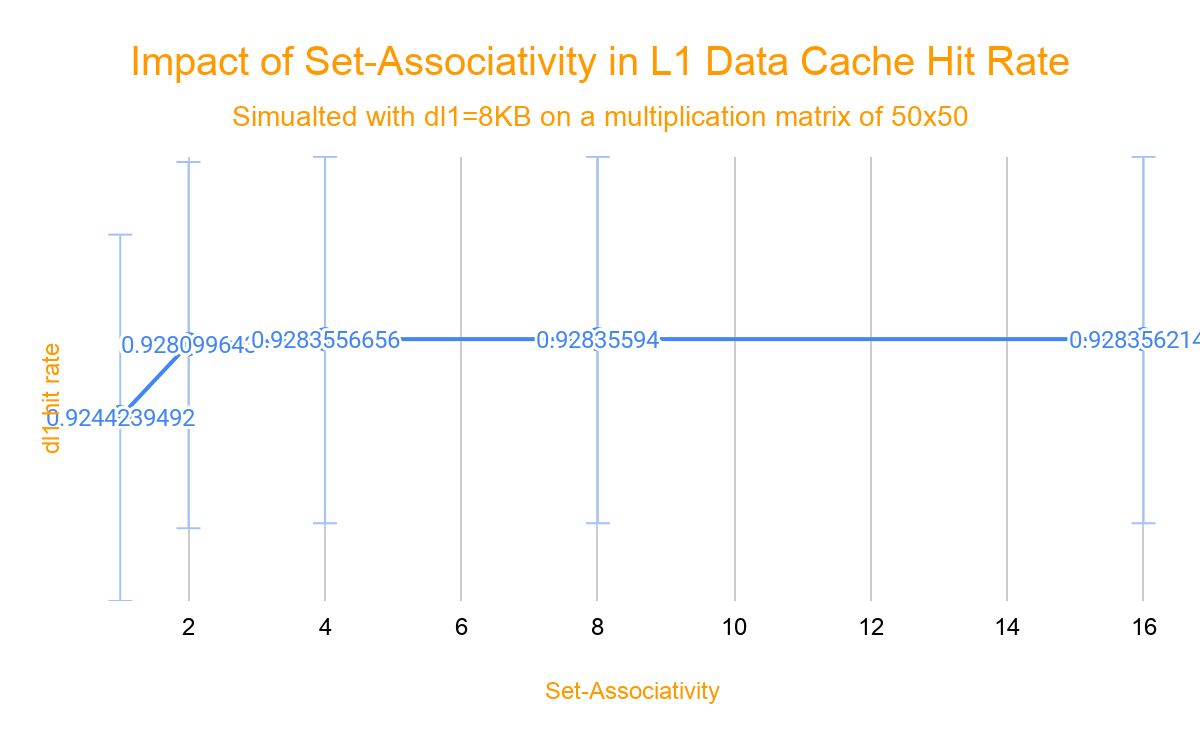
For this simulation let Data Cache be 8KB and a matmul.ss matrix of 50 x 50We will be changing the associativity to be 1, 2, 4, 8, 16. Table 2 contain the commands used to do so:

**Table 2. Set-associativity vs Hit and Miss Rate**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Assoc. | dl1 accesses | dl1 hits | dl1 hit rate | dl1 misses | dl1 miss rate | Commands |
| 1 | 3644210 | 3368795 | 0.9244239492 | 275415 | 0.07557605078 | ./sim-cache -cache:dl1 dl1:256:32:1:l matmul.ss 50 50 50 |
| 2 | 3644210 | 3382190 | 0.928099643 | 262020 | 0.071900357 | ./sim-cache -cache:dl1 dl1:128:32:2:l matmul.ss 50 50 50 |
| 4 | 3644210 | 3383123 | 0.9283556656 | 261087 | 0.07164433444 | ./sim-cache -cache:dl1 dl1:64:32:4:l matmul.ss 50 50 50 |
| 8 | 3644210 | 3383124 | 0.92835594 | 261086 | 0.07164406003 | ./sim-cache -cache:dl1 dl1:32:32:8:l matmul.ss 50 50 50 |
| 16 | 3644210 | 3383125 | 0.9283562144 | 261085 | 0.07164378562 | ./sim-cache -cache:dl1 dl1:16:32:16:l matmul.ss 50 50 50 |



**Fig. 7 Set-associativity vs miss rate for L1 data cache.**



**Fig. 8 Set-associativity vs hit rate for L1 data cache.**

We can observe in Figures 7 & 8 how increasing associativity contributes to reducing the miss rate and improves cache hit rate. From Direct mapped to 2-way associativity there is significant improvement, but from 2-way to 4-way, 8-way, to 16-way, there is still improvement although more conservative. Increasing associativity also provides flexibility to implement different replacement policies and to fully utilize the cache memory. However, lets understand that set-associativity is a trade off between direct mapped and fully associativity. With the increase in associativity, there is also an increase in complexity and power consumption and there is an increase in latency.

##### Output files from the Simulation can be found here:

<https://drive.google.com/drive/u/0/folders/11b9d4YEGnquWBwU_ewhLJVA-vWyvkXnf>

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

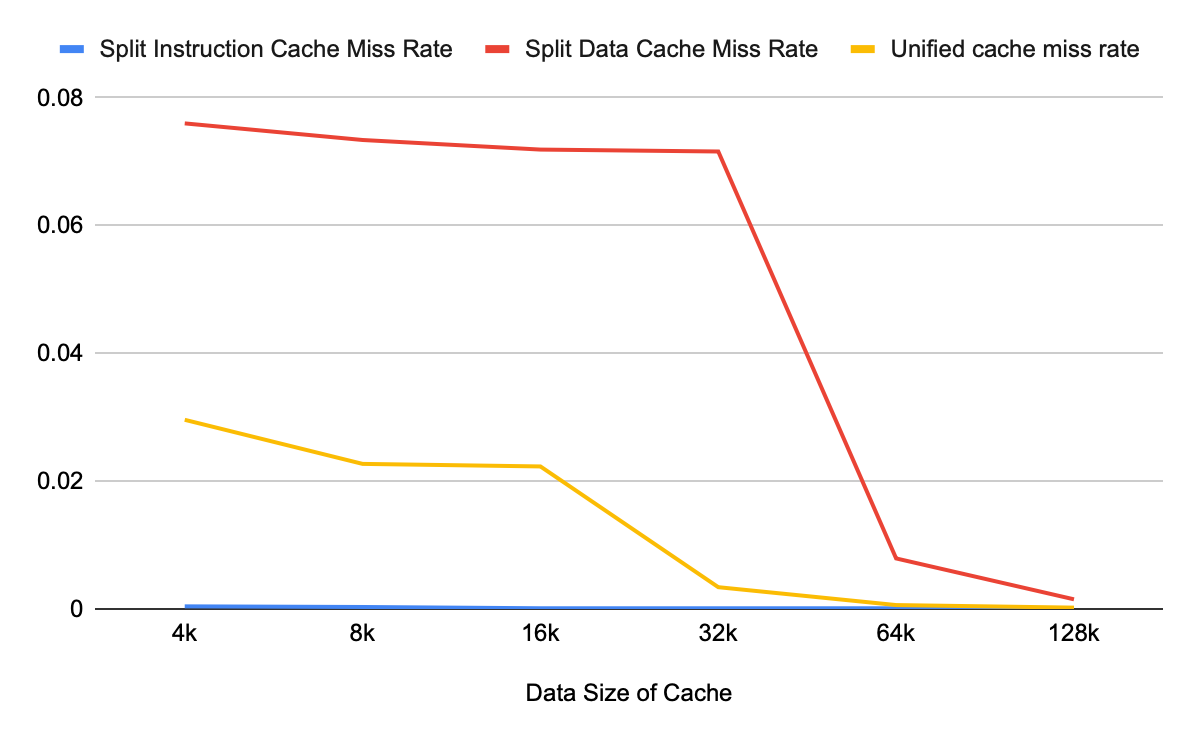
#### 

#### Part 3\_c: Unified vs Split Cache

Unified cache is when both instruction and data cache are joint into one cache. While split cache is when instruction and data cache are split into two distinct caches. Instruction cache is dedication to holding instructions while the data cache is dedication to holding data. Both instruction and data cache are considered to be a single cache because they both are hardware managed caches for the same address space at the same memory level hierarchy. Cache that is not split into these two is called a unified cache.

**Table 3: Results from simulation Split vs Unified Cache.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Size of Cache | Type | Split Instruction Cache Miss Rate | Split Data Cache Miss Rate | Type | Unified cache miss rate |
| 4k | split | 0.0004 | 0.076 | Unified | 0.0296 |
| 8k | split | 0.0003 | 0.0734 | Unified | 0.0227 |
| 16k | split | 0.0001 | 0.0719 | Unified | 0.0223 |
| 32k | split | 0.0001 | 0.0716 | Unified | 0.0034 |
| 64k | split | 0.0001 | 0.0079 | Unified | 0.0006 |
| 128k | split | 0.0001 | 0.0015 | Unified | 0.0002 |



**Fig. 9 Split instruction and data cache miss rate.**

In Figure 9 we see that the split instruction and data cache miss rate add up to a larger miss rate than that of the unified cache miss rate. This may be because unified cache performs load balancing. That means that if the current program needs more data than cache, it will automatically allocate more data addresses. Advantages to split is that it can do two references at once, one to instruction and another to data. For L1 split is a better pick because you do not care much about your miss ratio, you care more about the accessing times and this architecture provides better accessing times.

##### Output files from the Simulation can be found here:

<https://drive.google.com/drive/u/2/folders/1olNoOxvtDk-2VS1RwMFueetQQ40_M270>

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

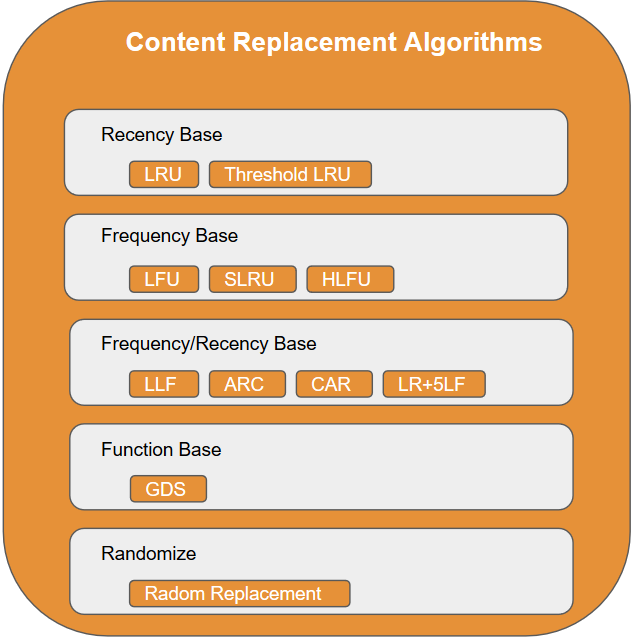
#### 

#### 

#### 

#### Part 3\_d : Sim-Cache simulation (Replacement Policy)

Replacement policies or algorithms are used to attain optimized use of Cache. When cache is full, then replacement policies decide which piece of data is replaced in order to make space for new data that is currently being used. An efficient algorithm is that which can take less time and number of cache misses are low and also balancing cost [1]. There are many replacement policies for cache , including Least Recently Used (LRU), Least Frequently Used (LFU), History Least Recently Used (HLFU), Segmented LRU (SLRU), First In First Out (FIFO), Random Replacement (RR), Clock with Adaptive replacement (CAR), Greedy Dual Size (GDS), LR+5LF Algorithm, and Lowest Latency First (LLF). These replacement algorithms are sometimes classified in several classes as shown in figure 10



**Fig. 10. Cache replacement algorithms by classes.**

SimpleScalar sim-cache accepts three different replacement policy options:

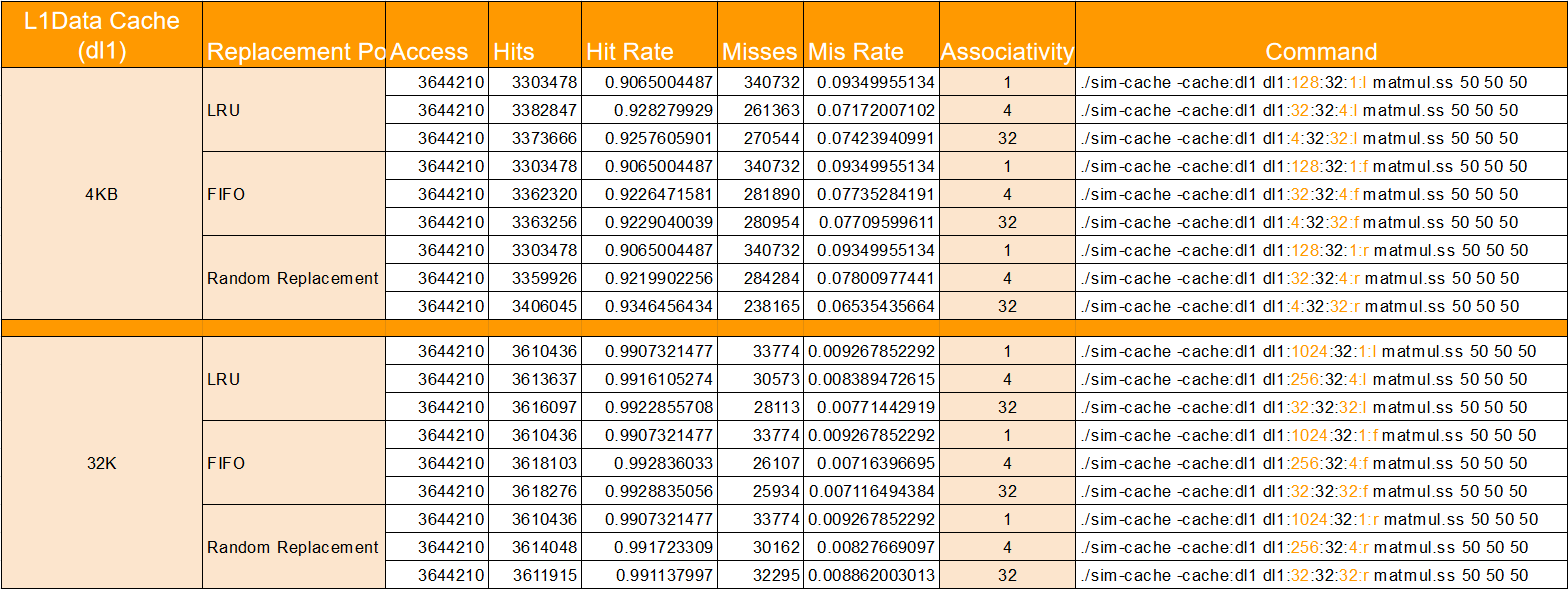
***l* [LRU]:** This algorithm discards the least recently used item from the cache in order to make space for the new data item. In order to achieve this, history of all data items, that is which data item is used when, is kept.

***f* [FIFO]:**The first in first out algorithm removes the page that has not been used for a long time. It treats the pages as a circular buffer, and pages are removed in a round robin fashion.

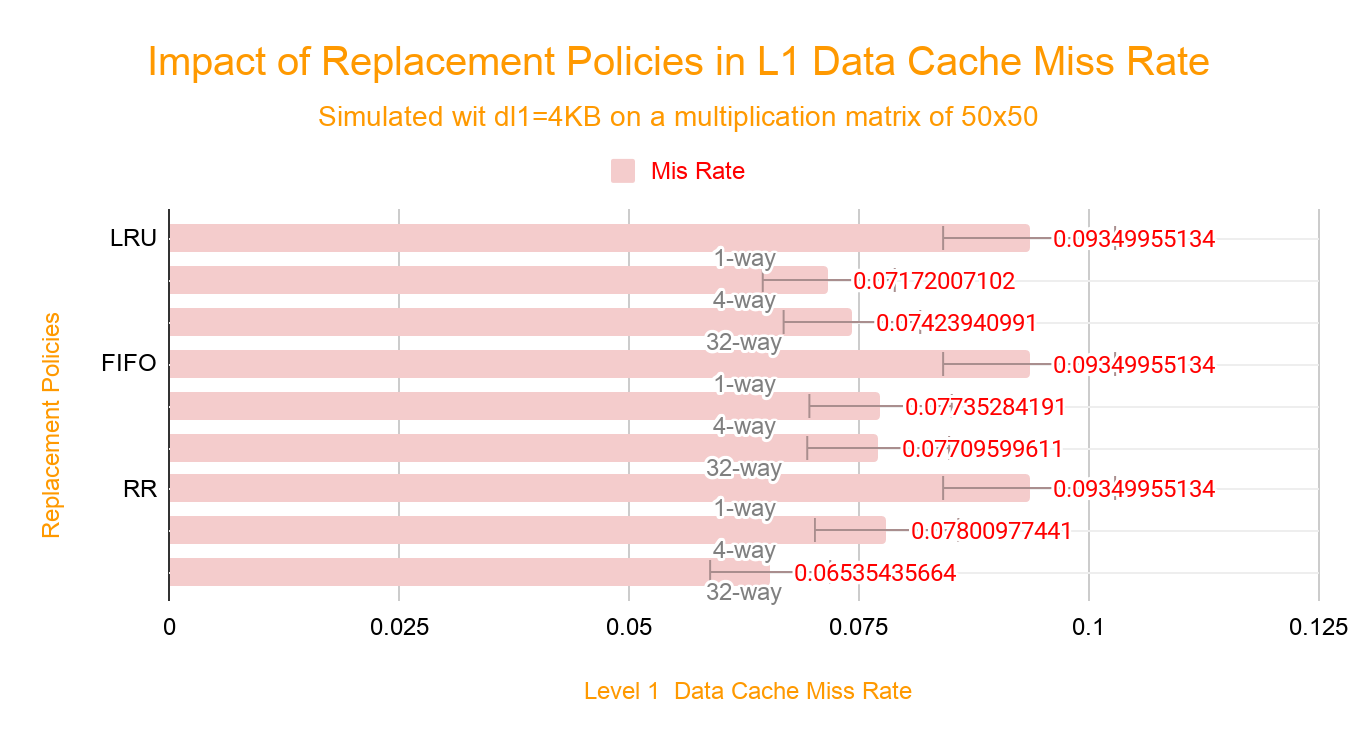
***r* [Random Replacement]**: This algorithm randomly selects any of the data items from the cache and replaces it with the desired one. This algorithm does not need to keep track of the history of the data contents and it does not need any data structure. Due to which it consumes less resources, therefore its cost is less as compare to other algorithms

For this part we are considering two different Level 1 Data Caches and three different associativity for each. Let dl1 be 4KB and 32KB respectively and 1,4,32 the associativity levels for the simulation. The simulation will use LRU, FIFO, and RR replacement policies to analyze the impact in the miiss rates. Table X shows the team calculated hits, misses, hit and miss rates obtained from simulation.

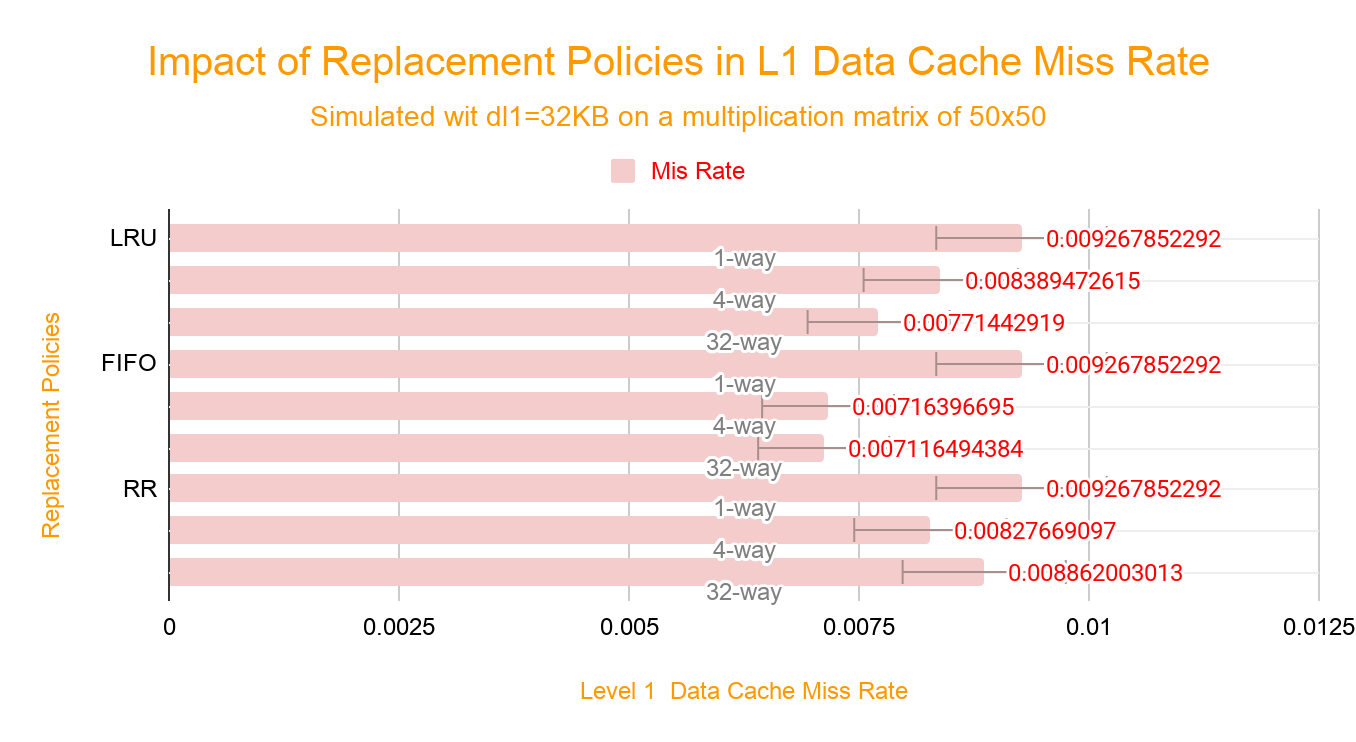
**Table 4. Replacement Policies impact in different size Level 1 Data Cache**



Figures 11 and 12 show the impact of the replacement policies in the miss rate for L1 data caches. As was observed in Part3\_b, higher associativity typically reduces miss rate. However, in some instances, a higher associativity shows worse miss rate as in figure 11 where a 4KB with 32-way associativity and LRU policy showed higher miss rate than the 4-way associativity. Similarly, for a 32KB with 32-way associativity and RR policy, the miss rate was higher than for a 4-way associativity. In summary, for a small L1 data cache (4KB) RR algorithm seems to provide the lower miss rate while for mid-side data cache (32KB) FIFO algorithm seems to provide the lower miss rate.

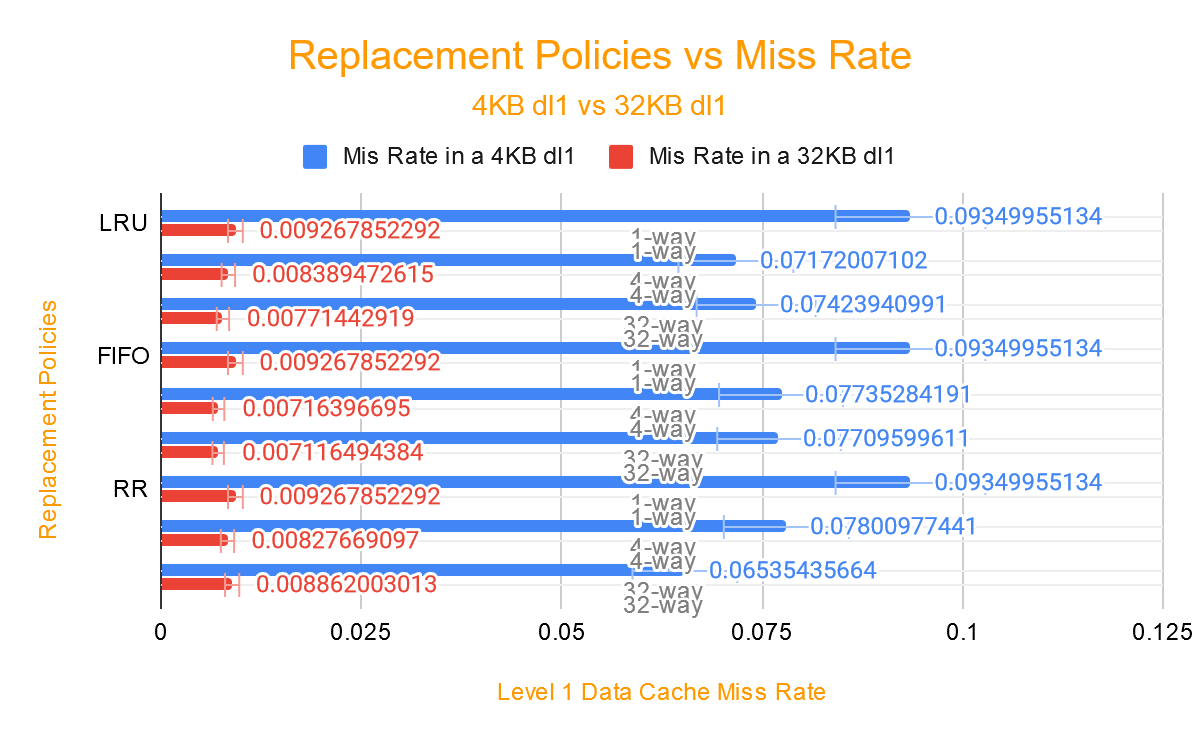


**Fig. 11 Impact of replacement policies in a 4KB L1 data cache.**



**Fig. 12 Impact of replacement policies in a 32KB L1 data cache.**

When analyzing both L1 data cache together (Figure 13), we observed how larger cache provides a much lower miss rate (by a factor of 10) and that for such cases the FIFO algorithm offers the lowest miss rate. Let's keep in mind that these results were observed on a specific program and that memory access in a different program can alter the previous observations. Also, higher associativity and/or cache size do always come with a trade off in terms of cost, complexity, power consumption, latency, etc.



**Fig.13 Impact of replacement policies in a 4KB and in a 32KB L1 data cache.**

##### Output files from the Simulation can be found here:

[**https://drive.google.com/drive/u/0/folders/1ts5gjjeDTIEOXLpB2kA8EtI\_sgh1HIff**](https://drive.google.com/drive/u/0/folders/1ts5gjjeDTIEOXLpB2kA8EtI_sgh1HIff)

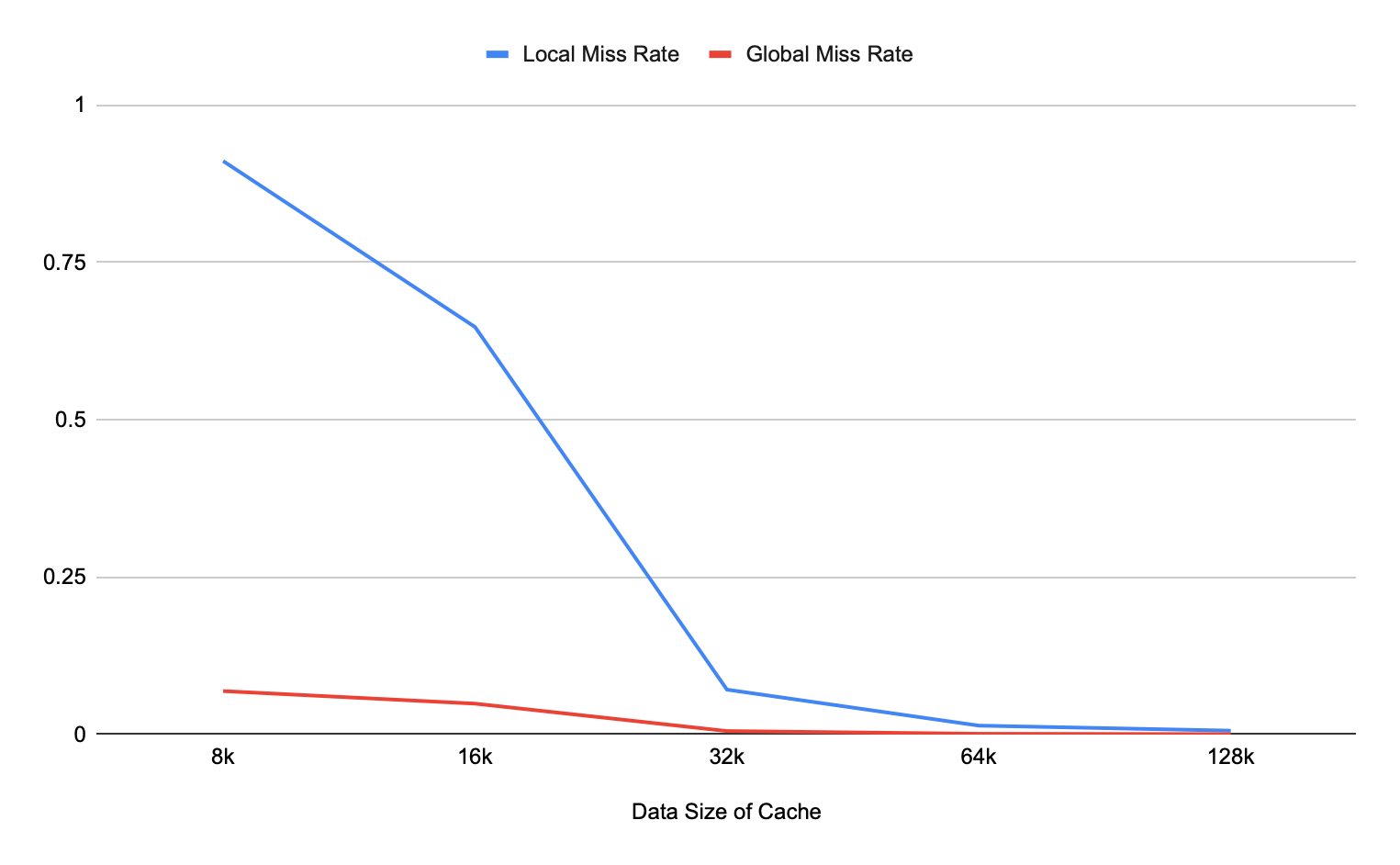
#### 

#### Part 3\_e: Multilevel Cache

Miss rate can be either local miss rates or global miss rates. The local miss rate refers to the percentage of miss accesses in a particular layer of cache. The global miss rate refers to the combined percentage of all accesses that miss at the cache. Therefore the local miss rate is their level 1 cache miss rate or level 2 cache miss rate depending on what you are looking into. In our case we are looking into level 2 cache miss rates. As for the global miss rates, this can be calculated with the following formula:

**Table 5: Results for multilevel cache**

|  |  |  |
| --- | --- | --- |
| Data Size of Cache | Local Miss Rate | Global Miss Rate |
| 8k | 0.9113 | 0.06889428 |
| 16k | 0.6475 | 0.048951 |
| 32k | 0.0712 | 0.00538272 |
| 64k | 0.014 | 0.0010584 |
| 128k | 0.006 | 0.0004536 |

****

**Fig.14 Multi Level with different L1 data cache vs miss rate.**

##### Output files from the Simulation can be found here:

[**https://drive.google.com/drive/u/2/folders/1goaaal5GfNuYY4a-ierKJ24\_qOdrTKv2**](https://drive.google.com/drive/u/2/folders/1goaaal5GfNuYY4a-ierKJ24_qOdrTKv2)

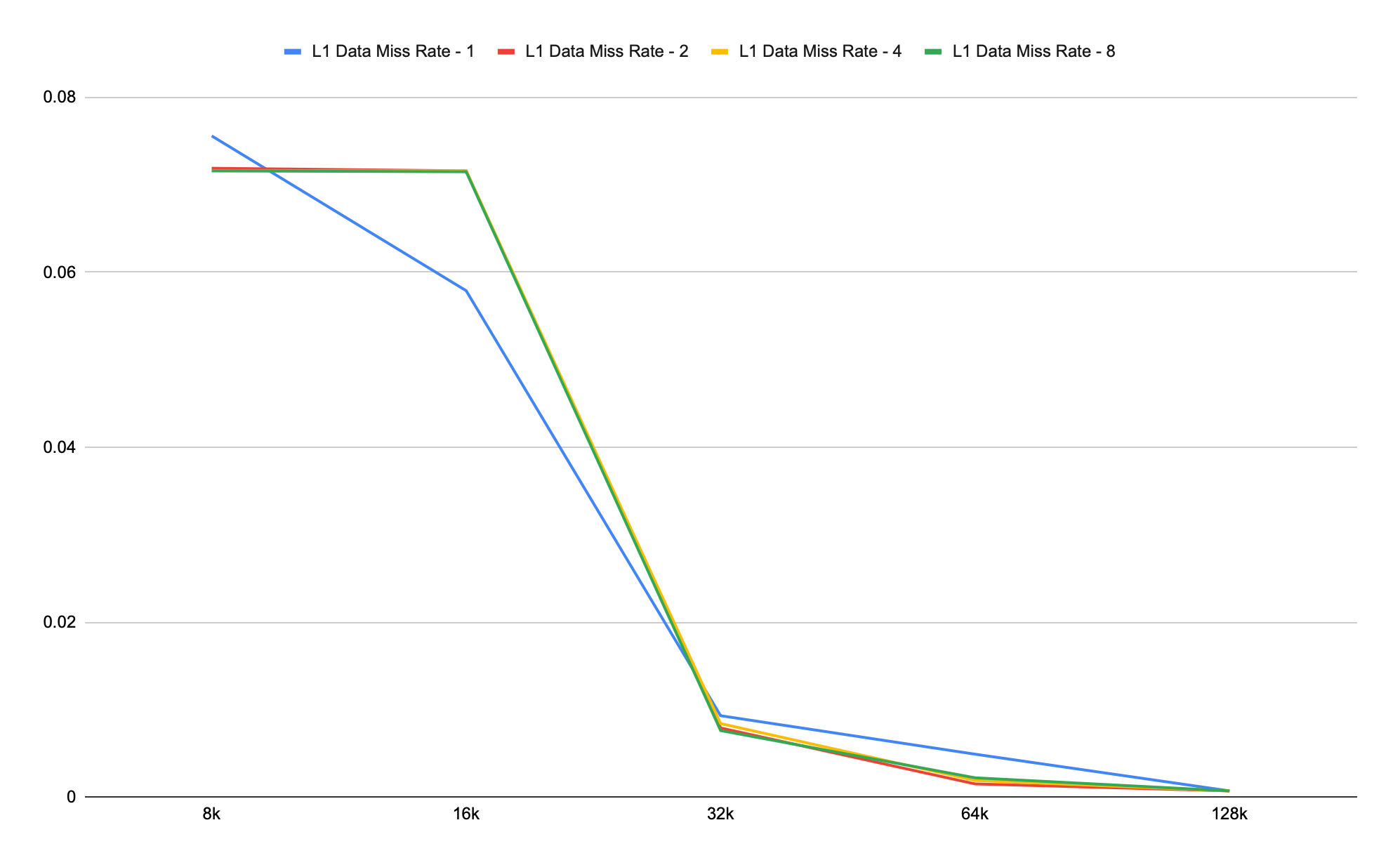
#### Part 3\_f: Three Types of Cache Misses

The three types of cache misses or otherwise known as the three C’s are the following:

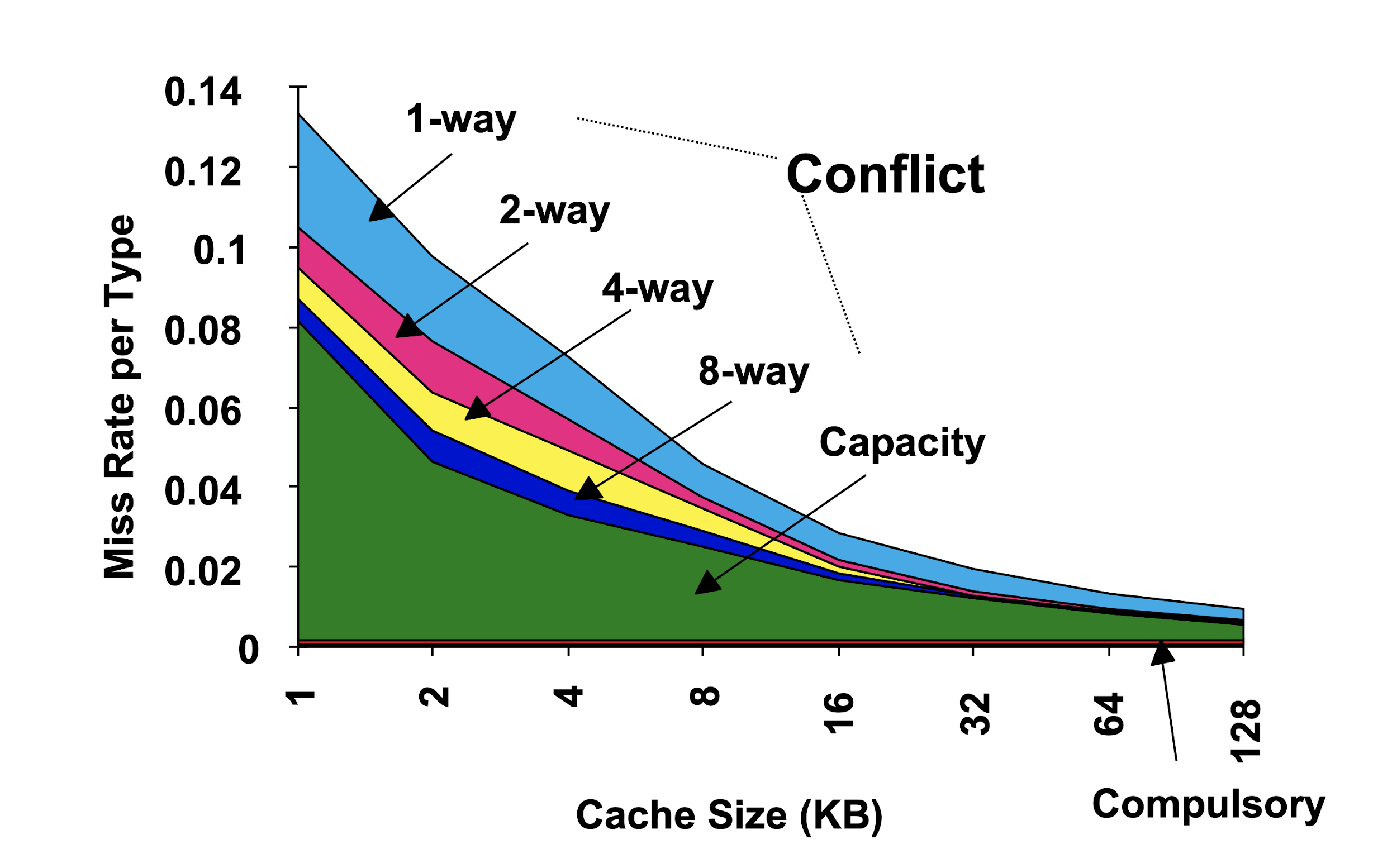
* Compulsory: Every first access to a block. This is due to the cache being empty therefore no hits are possible. They are also known as cold-start misses or first-reference misses.
* Capacity: Occur when cache cannot contain all the blocks needed during execution of a program. Therefore capacity misses will occur because of blocks being discarded and later retrieved.
* Conflict: Occur if the block placement strategy is set associative or direct mapped. Conflict misses will occur because a block may be discarded and later retrieved if too many blocks map to its set. The idea is that hits in a fully associative cache that become misses in an n-way set-associative cache are due to more than n requests on some popular sets. They are also known as collision misses.

**Table 6: Results for Three Types of Cache Misses**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Size of Cache | Assoc. | L1 Data Miss Rate - 1 | Assoc. | L1 Data Miss Rate - 2 | Assoc. | L1 Data Miss Rate - 4 | Assoc. | L1 Data Miss Rate - 8 |
| 8k | 1 | 0.0756 | 2 | 0.0719 | 4 | 0.0716 | 8 | 0.0716 |
| 16k | 1 | 0.0579 | 2 | 0.0716 | 4 | 0.0716 | 8 | 0.0715 |
| 32k | 1 | 0.0093 | 2 | 0.0079 | 4 | 0.0084 | 8 | 0.0076 |
| 64k | 1 | 0.0049 | 2 | 0.0015 | 4 | 0.0019 | 8 | 0.0022 |
| 128k | 1 | 0.0007 | 2 | 0.0007 | 4 | 0.0007 | 8 | 0.0007 |



**Fig. 15 Data Miss Rate 1 vs Data Miss Rate 2 vs Data Miss Rate 4 vs Data Miss Rate 8**



**Fig. 16 Sample Chart for Compulsory, Capacity & Conflict**

**Taken from:** [**http://home.ku.edu.tr/comp303/public\_html/Lecture15.pdf**](http://home.ku.edu.tr/comp303/public_html/Lecture15.pdf)

We can compare Fig.15 to Fig.16 to get the graph requested. We can say that anything below the graph collection is the capacity misses leaving anything above the graph collection to be conflict misses.

##### Output files from the Simulation can be found here:

[**https://drive.google.com/drive/u/2/folders/1iI3AGlrPXOt3OpCm90Vm1\_75rQRqtGvs**](https://drive.google.com/drive/u/2/folders/1iI3AGlrPXOt3OpCm90Vm1_75rQRqtGvs)

#### Part 3\_g: TLB

This part is looking to simulate the TLB (Translational Lookaside Buffer) and how the different levels of associativity and entry sizes will affect the TLB Misses.

As in part 3\_a, the default values for the config parameters are used when the instructions do not provide any input. The only config we change in this section is the one for the TLB. The commands used to run the six scenarios are as follows:

./sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 ul2:256:64:2:l -tlb:dtlb dtlb:256:1024:1:l matmulO0.ss 50 50 50

./sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 ul2:256:64:2:l -tlb:dtlb dtlb:128:1024:2:l matmulO0.ss 50 50 50

./sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 ul2:256:64:2:l -tlb:dtlb dtlb:64:1024:4:l matmulO0.ss 50 50 50

./sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 ul2:256:64:2:l -tlb:dtlb dtlb:128:1024:1:l matmulO0.ss 50 50 50

./sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 ul2:256:64:2:l -tlb:dtlb dtlb:64:1024:2:l matmulO0.ss 50 50 50

./sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 ul2:256:64:2:l -tlb:dtlb dtlb:32:1024:4:l matmulO0.ss 50 50 50

Table 7 below shows the results.

**Table 7: TLB simulation results.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Command** | **Page Size (kB)** | **Entries** | **Associativity** | **TLB Misses** |
| **1** | dtlb:256:1024:1:l | 1 | 4 | 1 | 99 |
| **2** | dtlb:128:1024:2:l | 1 | 4 | 2 | 99 |
| **3** | dtlb:64:1024:4:l | 1 | 4 | 4 | 99 |
| **4** | dtlb:128:1024:1:l | 1 | 8 | 1 | 10312 |
| **5** | dtlb:64:1024:2:l | 1 | 8 | 2 | 5103 |
| **6** | dtlb:32:1024:4:l | 1 | 8 | 4 | 99 |

In the data graphed below in Figure 17, it can be observed that TLB Misses depend heavily on the entry sizes.



**Fig. 17 TLB miss rate simulation ( TLB miss vs entry size)**

##### Output files from the Simulation can be found here:

<https://drive.google.com/drive/u/0/folders/100yL1tFqmKV_irVCCIkkgTE8QbRN0Iar>

## Simulation Summary

In this simulation assignment, we got familiar with using the sim-cache tool in SimpleScalar. We ran many different scenarios involving caches of different sizes, associativities, and other factors. The matmul.c file we had used in the first simulation assignment was reused here for the cache simulations. We learned that the hit and miss rates of cache can vary either by large amounts or by small amounts based on which way the cache is configured.

Among the factors evaluated were cache and block sizes, associativity, block replacement policies, and TLB.

One must keep in mind that despite improvements in hit rate, there can be downsides, such as increased power consumption, depending on the situation. Another thing to keep in mind is that when improvement occurs due to a change in one factor, there might be a degradation in the hit rate if a different factor is changed. All factors have to be evaluated one-by-one and also together to ensure the highest performance.

## Reference

[1] Qaisar Javaid, Ayesha Zafar, Muhammad Awais, Munam Shah. Cache Memory: An Analysis on Replacement Algorithms and Optimization Techniques. Mehran University Research Journal of Engineering and Technology, Mehran University of Engineering and Technology, Jamshoro, Pakistan,

2017, 36 (4), pp.831-840. hal-01700364

[2] Jacob, Bruce, David Wang, and Spencer Ng. Memory systems: cache, DRAM, disk. Morgan Kaufmann,

2010