Contents

[Overview: 1](#_Toc83544595)

[Question A: 1](#_Toc83544596)

[Instruction cache size (compare 8KB, 32 KB and 64KB, block size and associativity follow the default configuration) 1](#_Toc83544597)

[Question B: 3](#_Toc83544598)

[Data cache size (compare 2KB, 32KB, and 128KB, all 4-way set associative, block size follows the default configuration) 3](#_Toc83544599)

[Question C: 5](#_Toc83544600)

[Data cache associativity (compare 1-way, 2-way, 8-way, all 32KB, block size follows the default configuration) 5](#_Toc83544601)

[Question D: 5](#_Toc83544602)

[L2 cache size (compare 128KB, 512KB, and 1MB, block size and associativity follow the default configuration) 5](#_Toc83544603)

[Question E: 5](#_Toc83544604)

[L1 and L2 block size (compare 32B, 64B, 128B, and 256B, cache size and associativity follow the default configuration) 5](#_Toc83544605)

## Overview:

For this report the impact of cache size on the performance of various programs is assessed. In particular the metrics which are explored are: cache size vs IPC, cache size vs cache misses, and program memory accesses. The primary objective being to determine the performance effect of increasing the cache size vs programs which have more or less cache accesses. Below are the 5 configurations assessed A-E. The simulator contains three cache levels, IL1 (instruction cache 1), DL1 (data cache 1), and (UL2) a unified level 2 cache. The default values are as follows and are used for each simulation except for parameters noted as being varied:

IL1: 256 sets, 32Byte block size, 1-way associativity, replacement policy LRU -> 8KB total

DL1: 256 sets, 32Byte block size, 1-way associativity, replacement policy LRU -> 8KB total

UL2: 1024 sets, 64Byte block size, 4-way associativity, replacement policy LRU -> 256KB total

## Question A:

### Instruction cache size (compare 8KB, 32 KB and 64KB, block size and associativity follow the default configuration)

For these configurations, the IPC results are shown in figure 1. For several programs there is no effect on increasing the IL1 cache size (bzip2, galgel, swim, wupwize), however several other programs experience significant gains by increasing the block size (crafty, equake, mesa).

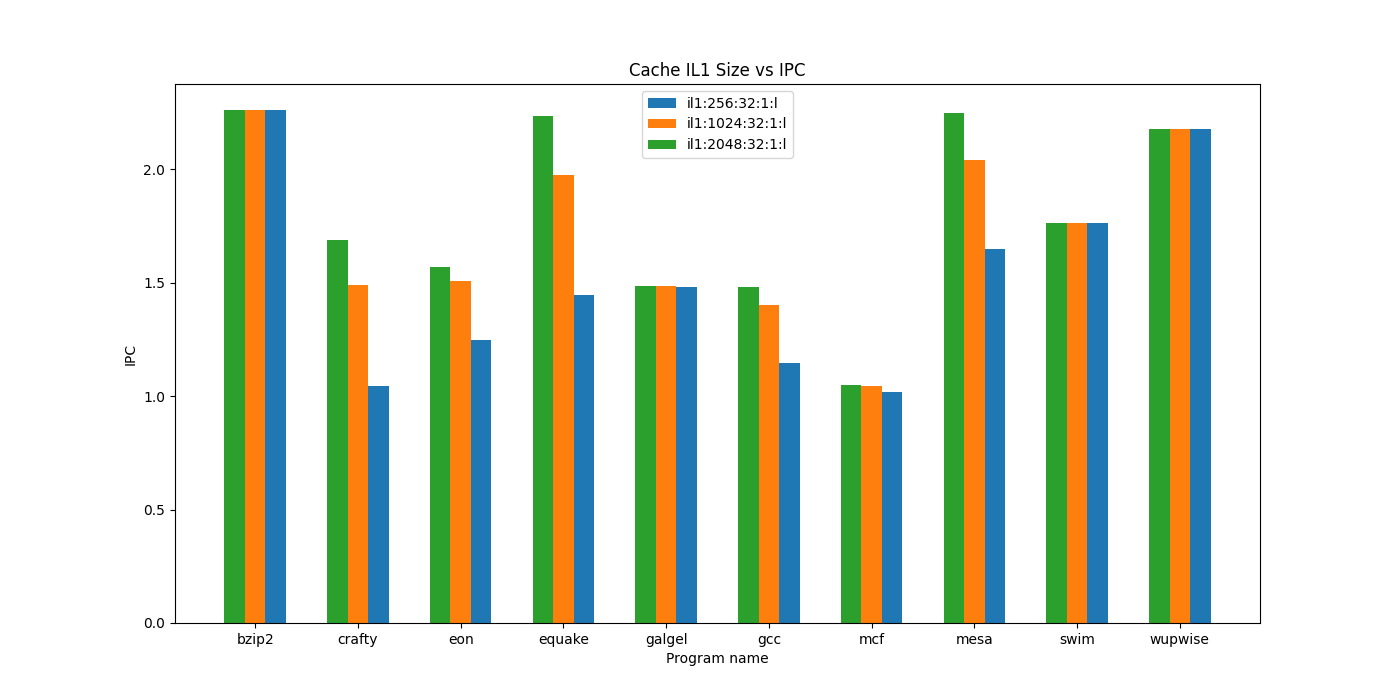


Figure : IL1 Cache Size vs IPC, larger is better

Figure 2, which shows the miss rates for IL1 cache shows that the programs which experience the greatest speedup also have a significant decrease in cache misses as the block size increases. Programs which do not experience this speedup have nearly zero cache misses regardless of the block size.

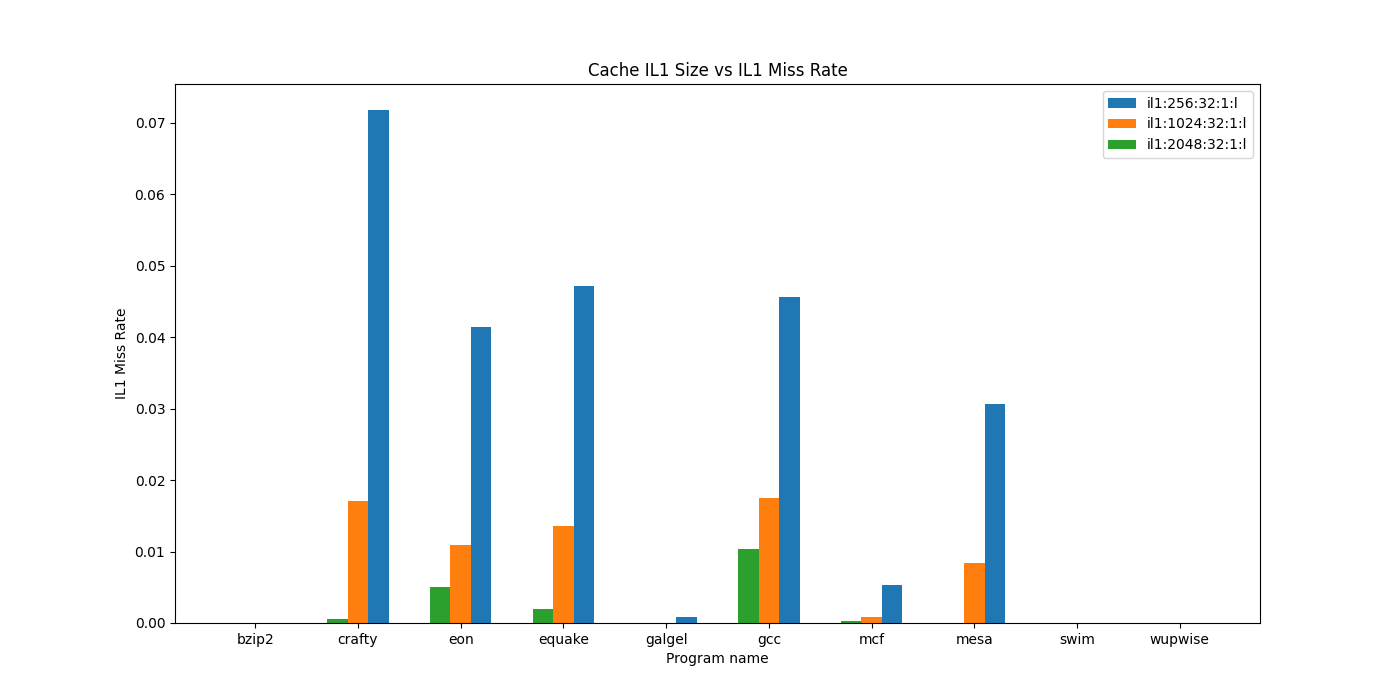


Figure : IL1 Cache Size vs IL1 Miss Rate

Figure 3. shows that these block misses may be a result of higher memory accesses, overall the programs which see the largest decrease in miss rate by increasing the cache size also see some improvements in the number of mem accesses (crafty, eon, mesa). The reason for this is likely that these programs are more complex and have a larger program space.

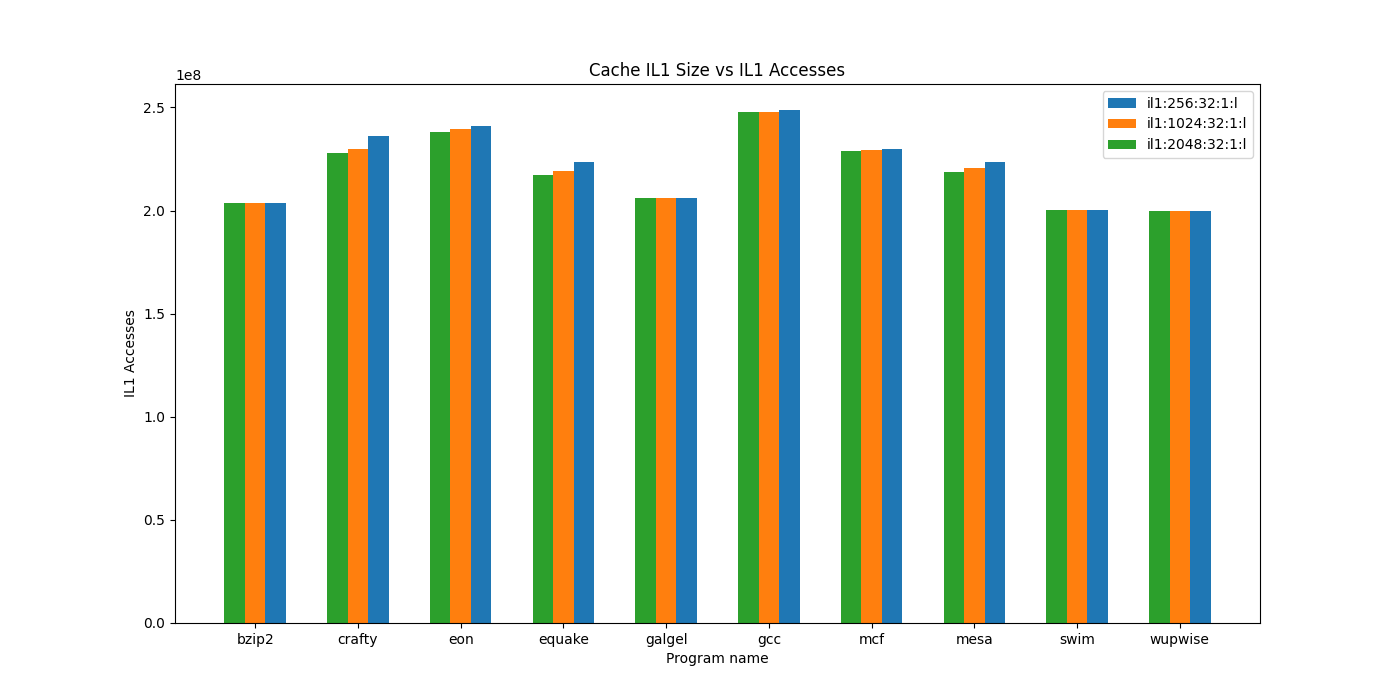


Figure : IL1 Cache Size vs IL1 Accesses

Table 1 shows the program’s ld\_text\_size (code size) at 8K for the various programs tested, which clearly shows that programs that experienced the largest performance gains are correlated with the programs with the largest code size. Three exceptions to this are galgel, swim and wupwise which have large program sizes, but also a comparatively low number of branch instructions taken. Since these programs have a low branch count it is likely that the mode in which they were run avoided much of their program space and thus they experienced fewer misses than expected based on their program size.

Table : Program name vs Code Size and Number of Branches taken at 8KB IL1

|  |  |  |
| --- | --- | --- |
| Program Name | ld\_text\_size (code size) in bytes | Sim\_num\_branches |
| Bzip2 | 196608 | 23326217 |
| Crafty | 442368 | 22871124 |
| Eon | 794624 | 22657053 |
| Equake | 253952 | 33049972 |
| Galgel | 1048576 | 18992383 |
| Gcc | 1990656 | 36328915 |
| Mcf | 163840 | 36298462 |
| Mesa | 917504 | 32911838 |
| Swim | 819200 | 6492449 |
| Wupwise | 819200 | 8163268 |

Overall increasing IL1 cache by increasing the block size can significantly improve the performance of some programs if they have large program spaces which do not fit into the IL1 cache or non-local branching within the program that pages large amounts of program code in and out of IL1 cache.

## Question B:

### Data cache size (compare 2KB, 32KB, and 128KB, all 4-way set associative, block size follows the default configuration)

Modifying the DL1 cache can also improve the performance of some programs as shown in figure 1. Nearly every program’s performance increases except wupwise which will be investigated further.

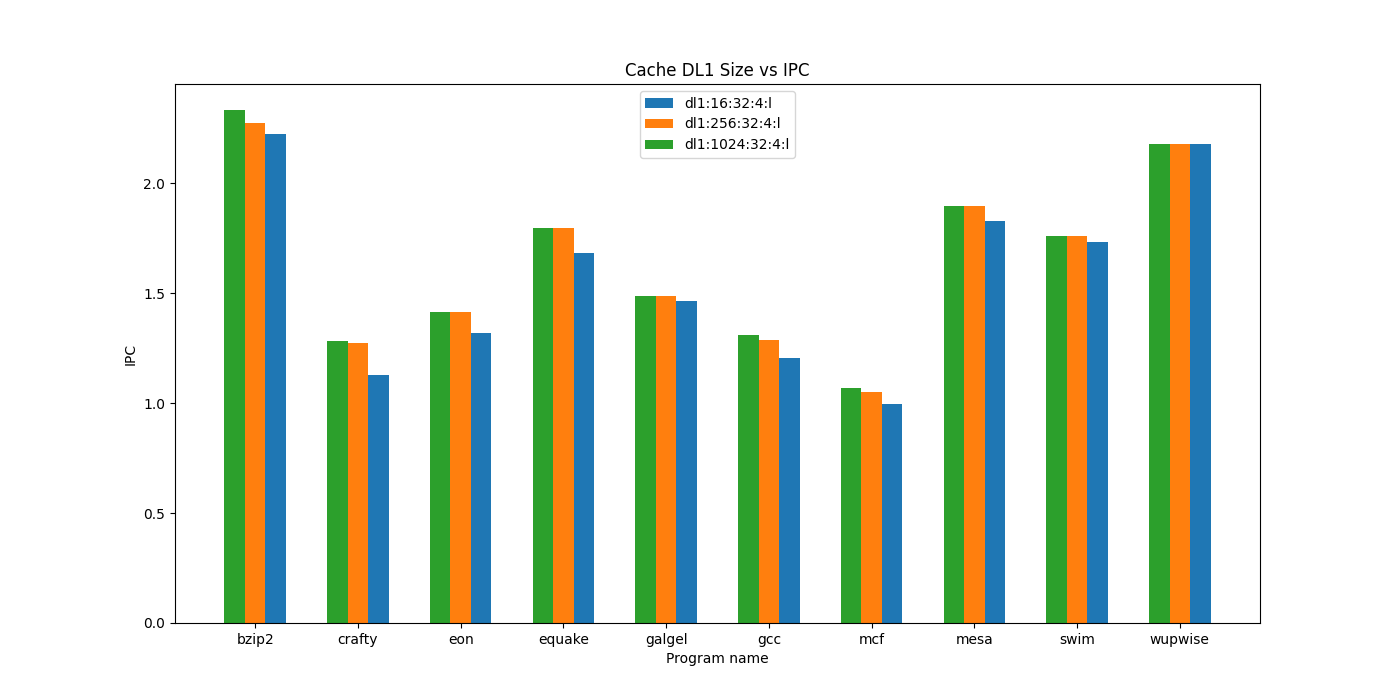


Figure : Cache DL1 Size vs IPC, higher is better

Figure 5 shows that programs which experience the largest performance benefits from increasing DL1 also are more likely to have a significant number of DL1 accesses, with galgel and wupwise both having few DL1 accesses are also those which experienced the fewest improvements (if any) from increasing DL1 size.

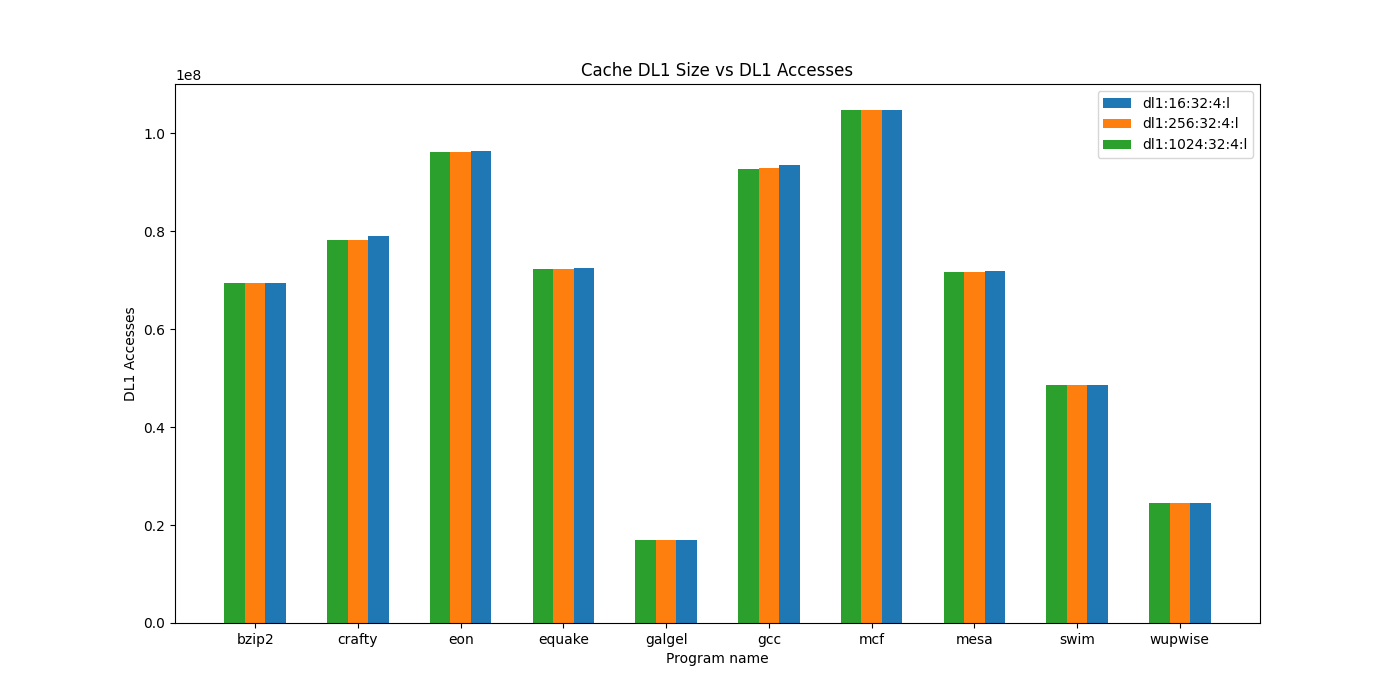


Figure : DL1 Cache Size vs DL1 Accesses

Figure 6, supports these findings by showing that most programs which have a large number of memory accesses to DL1 also experienced fewer misses to DL1 once cache was increased. This indicates that the program’s data cannot be effectively stored in DL1 until the cache size is increased.

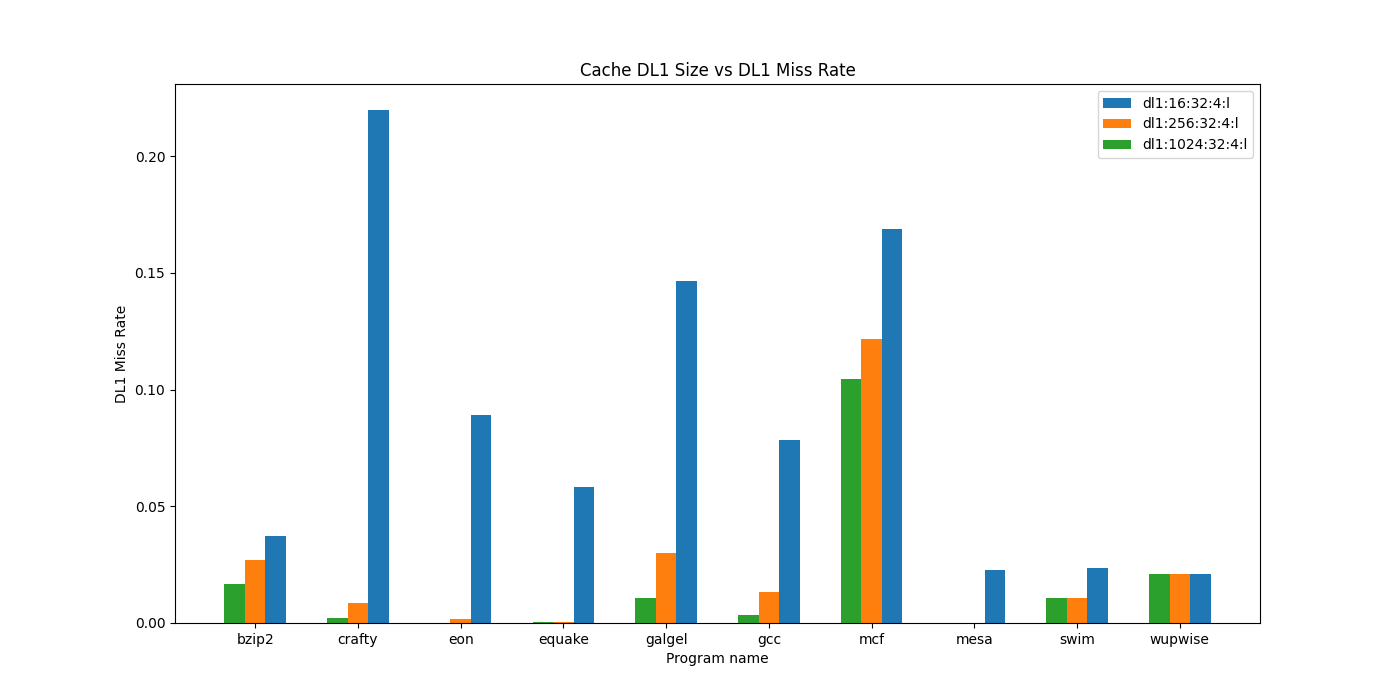


Table 2 also shows each program’s data segment size (static data) and total number of pages allocated (dynamic data) by the program. Programs with large ld\_data\_size, but smaller relative dynamic memory such as crafty and galgel see significant improvements in their performance by increasing the DL1 cache size, probably because they access many of these values internally and once the cache increases to from 2KB to 32KB their internal data can be more effectively stored and further increases do not improve the miss rate much. Other programs which allocate large amounts of dynamic memory such as mcf and wupwise do not see as much improvements to increasing the cache size, possibly due to the large size of their memory space or due to non-local memory accesses which cause intermittent paging as they switch between accessing large arrays. Further investigation into other cache replacement policies might improve the performance of mcf and wupwize to avoid this.

Table : Program ld\_data\_size (static memory) and page\_mem (dyanmic memory) statistics

|  |  |  |
| --- | --- | --- |
| Program Name | Ld\_data\_size (bytes) | Mem.page\_mem (bytes) |
| Bzip2 | 146544 | 8464k |
| Crafty | 1163616 | 2440k |
| Eon | 216592 | 1184k |
| Equake | 76704 | 1616k |
| Galgel | 138670480 | 8000k |
| Gcc | 306320 | 4792k |
| Mcf | 80448 | 189336k |
| Mesa | 91168 | 2512k |
| Swim | 199810704 | 28376k |
| WupWize | 184522544 | 18560k |

## Question C:

### Data cache associativity (compare 1-way, 2-way, 8-way, all 32KB, block size follows the default configuration)

Associativity has been shown to increase program performance by enabling a reduction in miss rates. However, the improvements are known to be minimal in many cases if the cache size is large enough already. Figure 6 shows that although increasing the associativity (without increasing the cache size) can increase performance the improvements are insignificant in most cases particularly past between 1-way and 2-way.

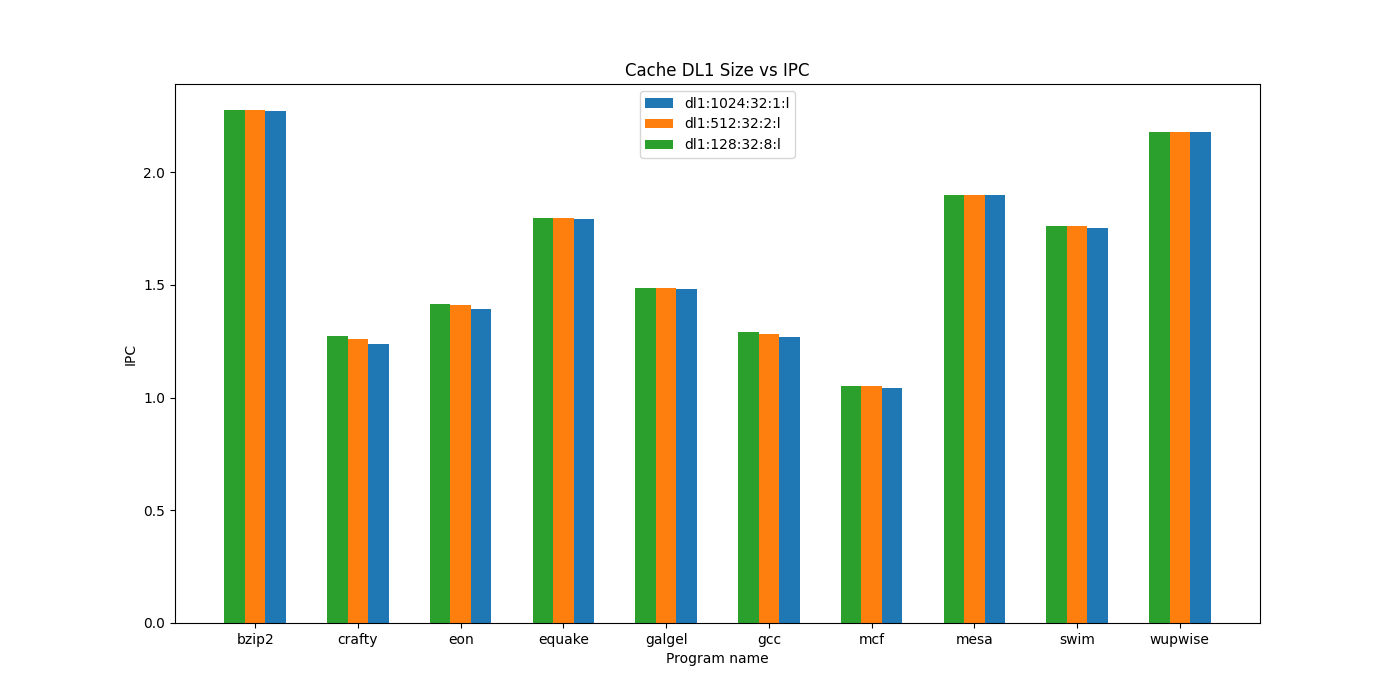


Figure : DL1 Cache Size vs IPC

Figure 7 shows that the total number of accesses still remains the same as expected. But also that programs with higher accesses are not always the ones which benefit, it depends upon the program.

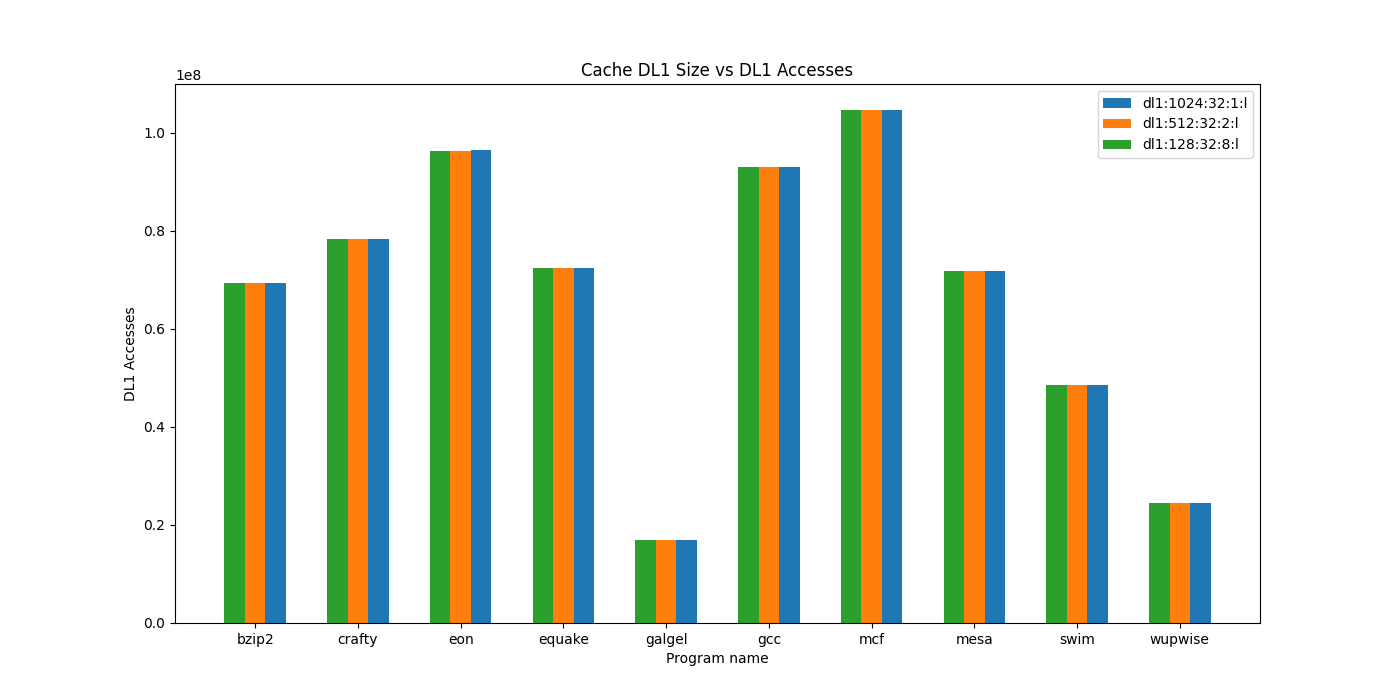


Figure : DL1 Cache Size vs DL1 Accesses

Figure 8 shows that the miss rate is indeed decreased, but only by a small amount for most programs which explains the reasons behind the lacklustre increase in performance for these programs. Crafty, eon and gcc are the three that see the most improvement in cache misses and are also the programs which see the largest increase in IPC performance. Programs like wupwize and bzip2 see little improvement in cache misses and thus see little IPC improvement.

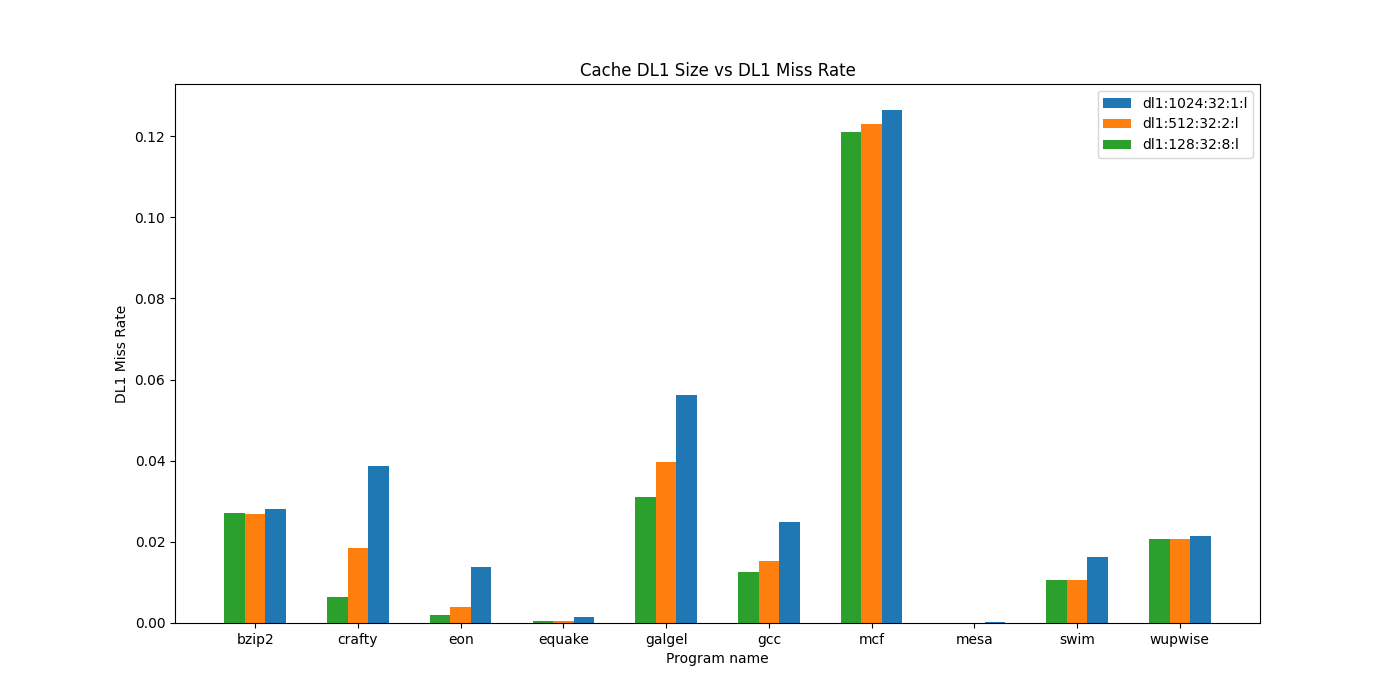


Figure : DL1 Cache Size vs DL1 Miss Rate

## Question D:

### L2 cache size (compare 128KB, 512KB, and 1MB, block size and associativity follow the default configuration)

Figure 9 shows the UL2 cache size vs IPC and only a few programs benefit from increasing the UL2 size among those which see the performance increases are bzip2, crafty, gcc and mcf.

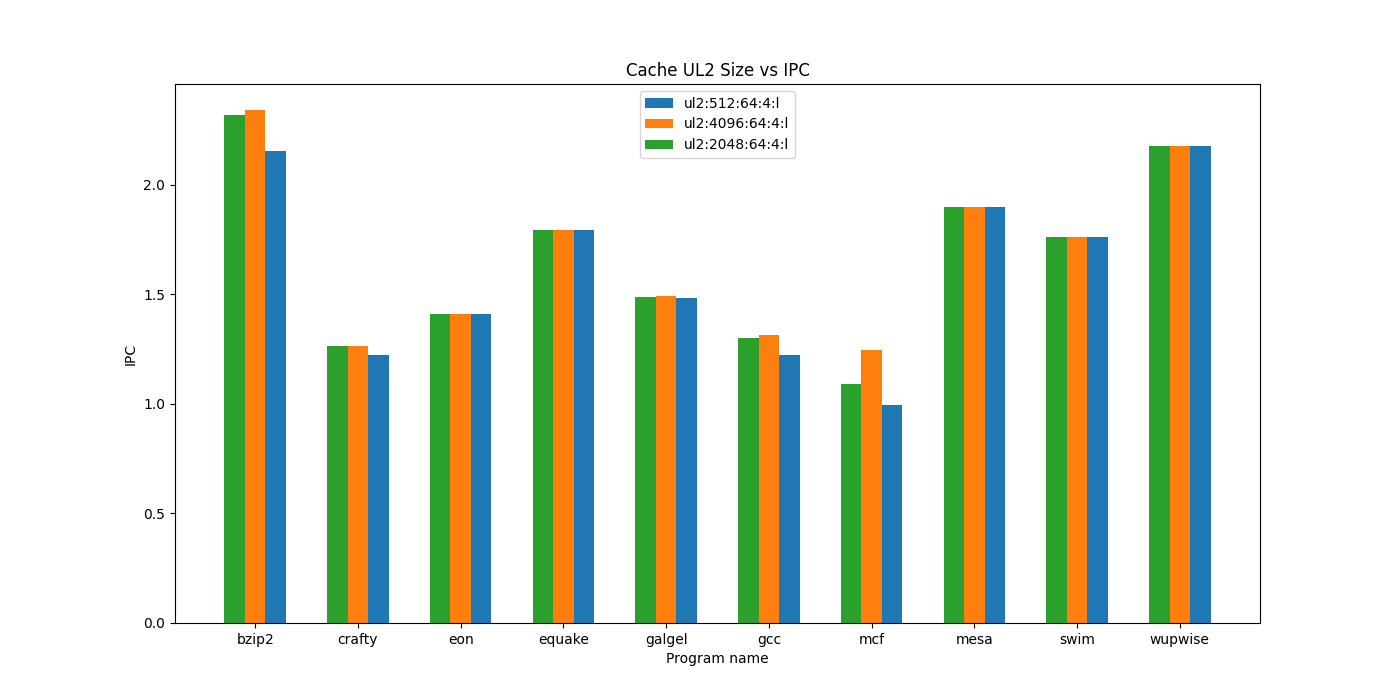


Figure : UL2 Cache Size vs IPC

Figure 10 shows that higher UL2 access rates do not necessarily indicate that increasing the UL2 size will improve performance as bzip2 and mesa have similar access rates yet only bzip2 shows improvement in performance overall

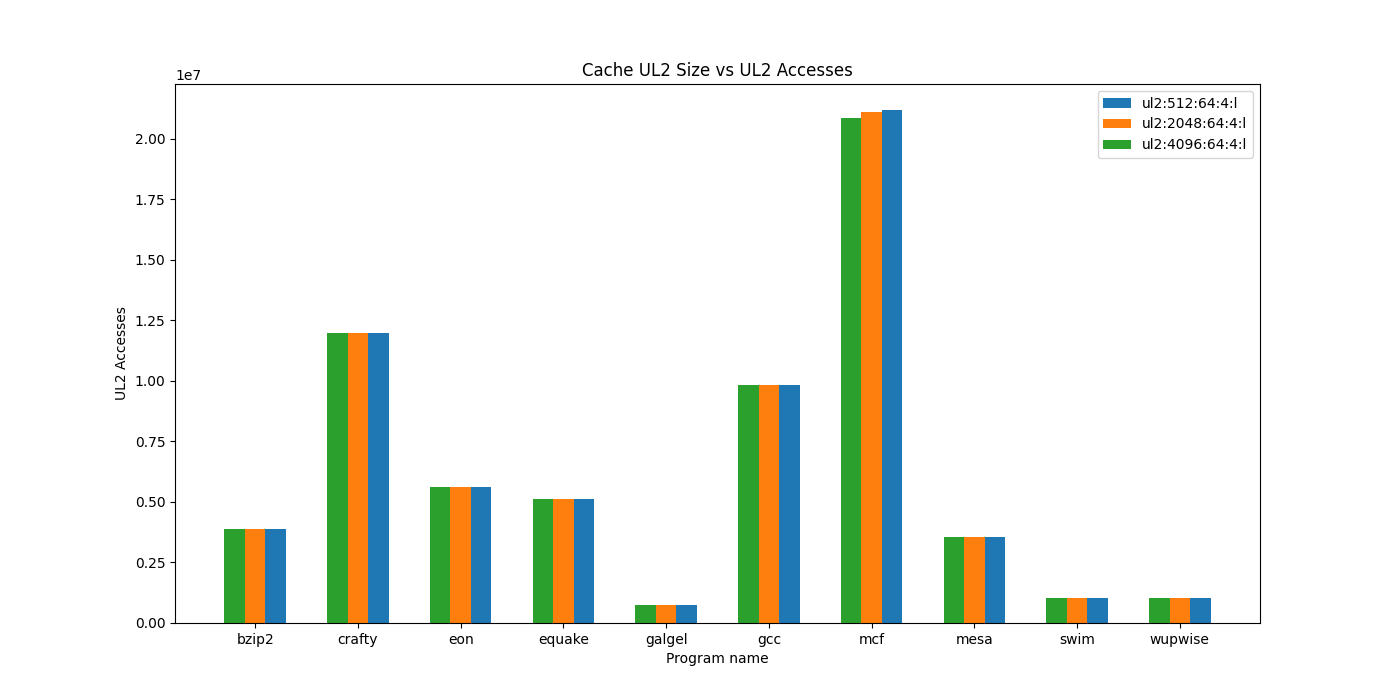


Figure : UL2 Cache Size vs UL2 Accesses

This is explained however when looking at the UL2 miss rates (figure 11), bzip2 experiences a substantial decrease in UL2 miss rate compared to mesa, and this holds true for all programs which see performance increases: bzip2, crafty, gcc, and mcf. Galgel being the outlier as it does see performance increases, but not nearly as significant as expected based on program accesses, this is explained by its low level of UL2 accesses compared to other programs in general.

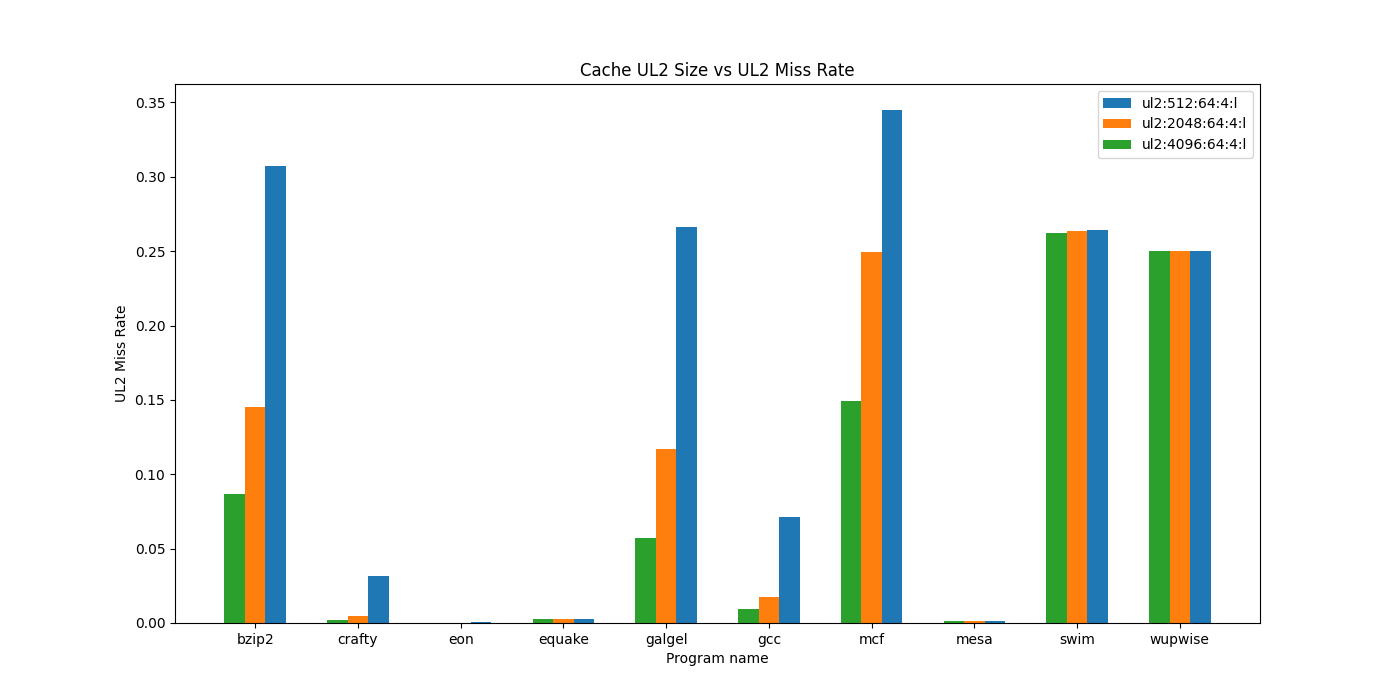


Figure : UL2 Cache Size vs UL2 Miss Rate

The reason for lack of performance increases or decreases in cache miss rate in UL2 for other programs such as swim and wupwise cannot be dismissed. Investigating further both swim and wupwise allocate significant dynamic memory [18560k, 28376k] respectively compared to other programs like bzip2 and galgel [8464k, 8000k] respectively. However, mcf allocates 189336k, which is more than swim or wupwise. It is likely that mcf has significantly more data locality than swim or wupwise which enables it to benefit more from a larger UL2 cache while swim and wupwise lack good data locality resulting in a large number of misses regardless of cache size.

## Question E:

### L1 and L2 block size (compare 32B, 64B, 128B, and 256B, cache size and associativity follow the default configuration)

By increasing the block size the performance gains are not consistently reliable. For some programs larger block sizes can result in significant performance improvements, see bzip, for other programs smaller block size of 32 results in best performance, see equake.

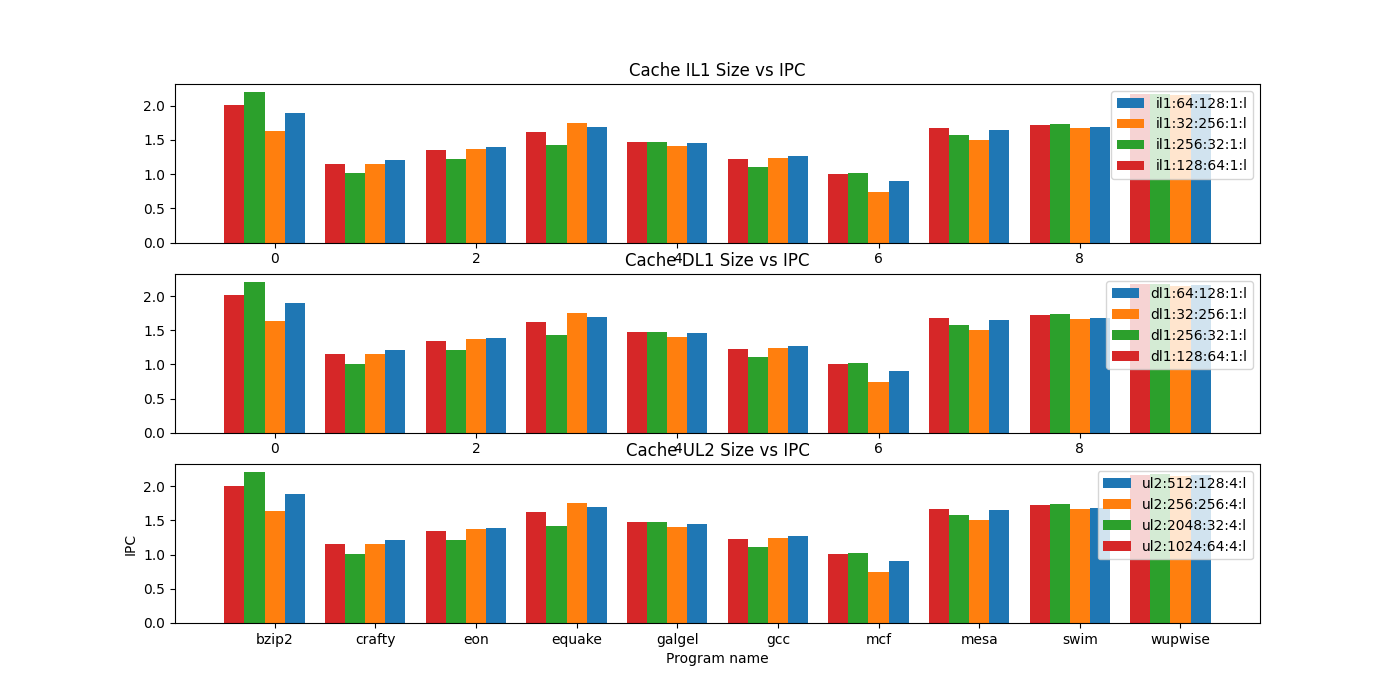
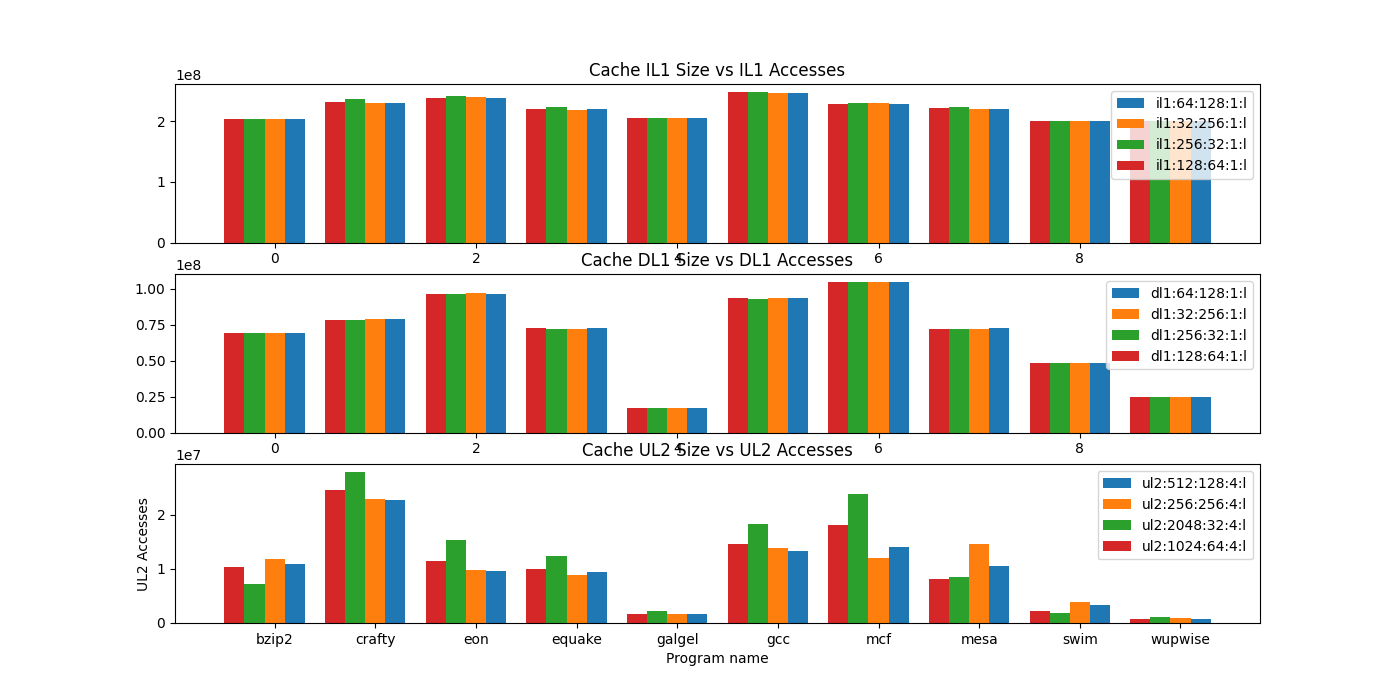


Figure : IL1, DL1 and UL2 Cache Sizes vs IPC

Figure 13 shows that block size can have an effect on the number of memory accesses, but primarily at the UL2, where again the results are inconsistent between the programs.

Figure : IL1, DL1 and UL2 Cache Sizes vs Number Accesses

For the miss rate however 32B block sizes usually result in the largest miss rate, followed by 64, 128 and 256. This is particularly true for IL1 cache. Thus the recommended cache size of IL1 is 256 based on the results. For DL1 the data shows that on average 32 byte blocks result in the lowest overall miss rate, while 256 byte caches can result in more misses. For UL2 cache results are again mixed, but the larger block sizes of 128 or 256 bytes seems to provide optimal reduction in miss rates.

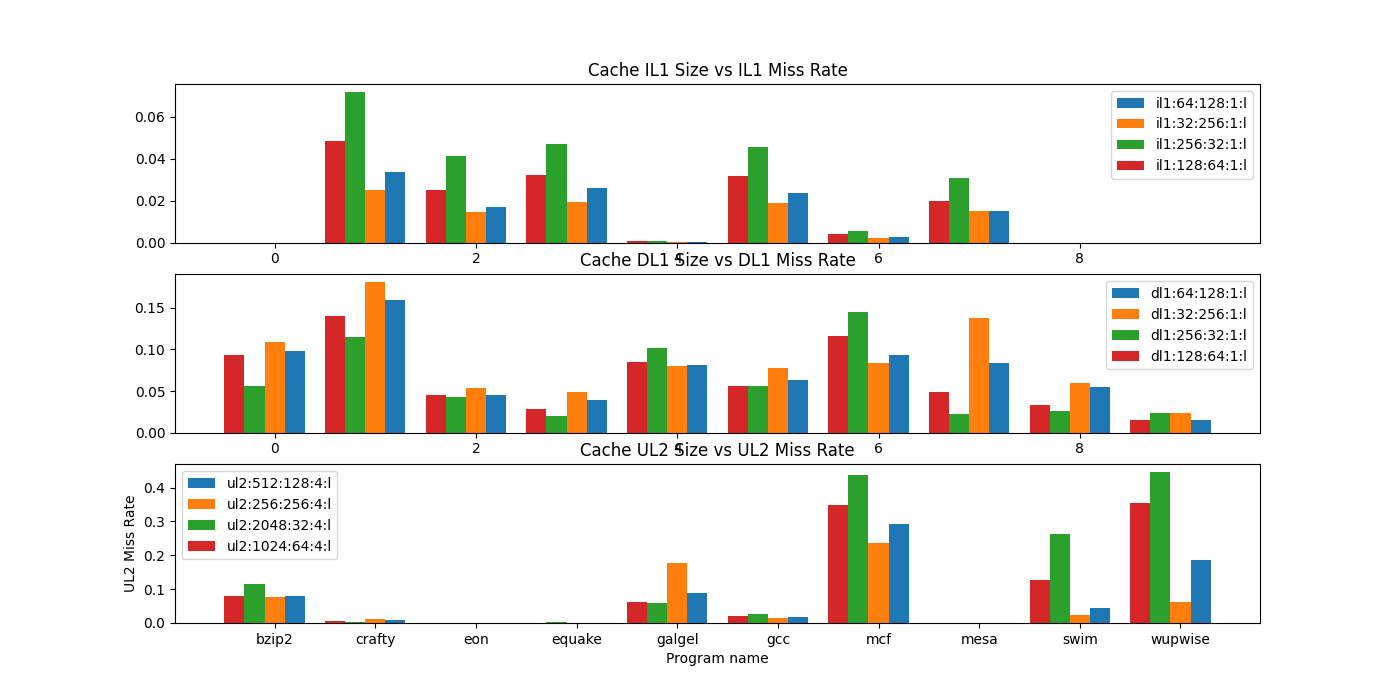


Figure : IL1, DL1 and UL2 Cache Sizes vs Miss Rates