PROJECT 1

PLAWAN KUMAR RATH(PKR140030)
SINDHUJA VENKATESAN(SXV143530)
CS6304

TABLE OF CONTENTS

INTRODUCTION

The Project was a simulatoin activity which required the usage of the SimpleScalar platform to simulate the various possible cache configurations. The main aim of this project involved learning the implementation of Simplescalar platform, implementing a given cache configuration using the platform and running it across 3 well known benchmarks, then it was required to consider various possible cache splits, associativities, block sizes, etc to find out an optimal configuration of cache which gave lowest CPI. The benchmarks that we used for simulation and to gather data sets for our computations were the GCC Benchmarks, the Anagram Benchmarks and the GO Benchmarks. For finding optimal configuration, we also considered set-associativities of 1,2,4,8 or fully associative, replacement policy among FIFO,LRU and Random policy and Block Size of either 64bytes or those of 32 bytes. Considering all these combinations, we calculated all their CPIs and then analyzed them to find the optimal configuration. Another task of the project was to assign costs to the various cache components and then find optimal configurations across all benchmarks considering both cost and CPI as our data set. This project highlights the changes that occur to the CPI when various cache configurations are changed. It also shows us the cost and CPI tradeoffs.

APPROACH

This project was assigned as a group project for two people. Our group members were Plawan Kumar Rath and Sindhuja Venkatesan. We completed the assignment by dividing the various tasks into groups and delegating the groups equally between the both of us. Plawan installed SimpleScalar in his machine and so did Sindhuja. Sindhuja then ran the simplescalar to obtain the values of all necessary parameters for calculation of CPI, and these were used by Plawan to complete the Part 2 of the project. For the Part 3, Plawan wrote 9 scripts which generated 9 output files for various cache splits across 3 different benchmarks(namely Go,GCC and Anagram) taking all possible configurations for each cache split. Plawan then generated the excel sheets which contained the parameters needed for CPI calculation. Sindhuja then took those excel sheets and calculated the CPIs and plotted the graphs, in the process finding optimal CPIs. Sindhuja then went on to define a cost function, by assigning appropriate weights to the various components. Plawan then used this cost function and the CPIs calculated before to find the optimal configurations considering both CPI and Cost together. Then, finally the report was prepared mutually, by both Plawan and Sindhuja.

This approach was used in our group because, with this approach both of us got to perform at least some task in each part and in the process both of us gained equal amount of knowledge and exposure from the project.

PART 1

This stage involved setting up of SimpleScalar and testing whether simplescalar and the benchmarks were installed and working properly. At this stage, we both installed Simplescalar and tested the different benchmarks. We took some time to familiarize ourselves well with the platform and test the benchmarks. We had the option of using four benchmarks, namely GCC Benchmarks, GO Benchmarks, ANAGRAM Benchmarks and COMPRESS95 Benchmarks. At this stage we discovered some issues with the COMPRESS95 benchmarks and after discussion and upon consultation with the teaching assistant we decided to use GCC,GO and ANAGRAM benchmarks for the rest of our project statement.

PART 2

In this part, we calculate the CPI for the three individual benchmarks. Our baseline configuration is the Alpha 21264 EV6 configuration:

- Cache levels: Two levels.

- Unified caches: Separate L1 data and instruction cache, unified L2 cache.

- Size: 64K Separate L1 data and instruction caches, 1MB unified L2 cache.

- Associativity: Two-way set-associative L1 caches, Direct-mapped L2 cache.

- Block size: 64 bytes.

- Block replacement policy: FIFO.

Also, given L1 miss penalty = 5 cycles and L2 miss penalty = 40 cycles.

The CPI has been calculated using the following formula:

CPI = CPI ideal + 5* (L1InsMissRate * (L1Ins Access/Total Ins) + L1DataMissRate * (L1Data access/total Ins)) + 40 * (L2MissRate * (L2 access/Total Ins))

Given below are the simulation results and the CPI for each of the three benchmarks, for the given configuration:

GCC Benchmarks

{cs6304-32:~/Project_1/simplesim-3.0} ./sim-cache -cache:dl1 dl1:512:64:2:f -cache:il1 il1:512:64:2:f -cache:il2 dl2 -cache:dl2 ul2:16384:64:1:f -tlb:itlb none -tlb:dtlb none benchmarks/cc1.alpha -O benchmarks/1stmt.i

*sim: ** simulation statistics ***

sim_num_insn 337330187 # total number of instructions executed

sim_num_refs 121894242 # total number of loads and stores executed

sim_elapsed_time 36 # total simulation time in seconds

sim_inst_rate 9370282.9722 # simulation speed (in insts/sec)

il1.accesses 337330187 # total number of accesses

il1.hits 335742191 # total number of hits

il1.misses 1587996 # total number of misses

il1.replacements 1586972 # total number of replacements

il1.writebacks 0 # total number of writebacks

il1.invalidations 0 # total number of invalidations

il1.miss_rate 0.0047 # miss rate (i.e., misses/ref)

il1.repl_rate 0.0047 # replacement rate (i.e., repls/ref)

il1.wb_rate 0.0000 # writeback rate (i.e., wrbks/ref)

il1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)

dl1.accesses 124104359 # total number of accesses

dl1.hits 122789983 # total number of hits

dl1.misses 1314376 # total number of misses

dl1.replacements 1313352 # total number of replacements

dl1.writebacks 416880 # total number of writebacks

dl1.invalidations 0 # total number of invalidations *dl1.miss_rate 0.0106 # miss rate (i.e., misses/ref)*

dl1.repl_rate 0.0106 # replacement rate (i.e., repls/ref)

dl1.wb_rate 0.0034 # writeback rate (i.e., wrbks/ref)

dl1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)

ul2.accesses 3319252 # total number of accesses

ul2.hits 2892370 # total number of hits

ul2.misses 426882 # total number of misses

ul2.replacements 410498 # total number of replacements

ul2.writebacks 138069 # total number of writebacks

ul2.invalidations 0 # total number of invalidations

ul2.miss_rate 0.1286 # miss rate (i.e., misses/ref)

ul2.repl_rate 0.1237 # replacement rate (i.e., repls/ref)

ul2.wb_rate 0.0416 # writeback rate (i.e., wrbks/ref)

ul2.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)

ld_text_base 0x0120000000 # program text (code) segment base

ld_text_size 1564672 # program text (code) size in bytes

ld_data_base 0x0140000000 # program initialized data segment base

ld_data_size 277104 # program init'ed `.data' and uninit'ed `.bss' size in bytes

ld_stack_base 0x011ff9b000 # program stack segment base (highest address in stack)

ld_stack_size 16384 # program initial stack size

ld_prog_entry 0x0120025f70 # program entry point (initial PC)

ld_environ_base 0x011ff97000 # program environment base address address

ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian

mem.page_count 785 # total number of pages allocated

mem.page_mem 6280k # total size of memory pages allocated

mem.ptab_misses 613823 # total first level page table misses *mem.ptab_accesses 926309129 # total page table accesses*

mem.ptab_miss_rate 0.0007 # first level page table miss rate

Using the formula given above:

CPI = 1.0936145

ANAGRAM BENCHMARKS:

{cs6304-32:~/Project_1/simplesim-3.0} ./sim-cache -cache:dl1 dl1:512:64:2:f -cache:il1 il1:512:64:2:f -cache:il2 dl2 -cache:dl2 ul2:16384:64:1:f -tlb:itlb none -tlb:dtlb none benchmarks/anagram.alpha benchmarks/words < benchmarks/anagram.in > OUT

*sim: ** simulation statistics ***

sim_num_insn 25593186 # total number of instructions executed

sim_num_refs 9031728 # total number of loads and stores executed

sim_elapsed_time 3 # total simulation time in seconds

sim_inst_rate 8531062.0000 # simulation speed (in insts/sec)

il1.accesses 25593186 # total number of accesses

il1.hits 25592691 # total number of hits

il1.misses 495 # total number of misses

il1.replacements 16 # total number of replacements

il1.writebacks 0 # total number of writebacks

il1.invalidations 0 # total number of invalidations

il1.miss_rate 0.0000 # miss rate (i.e., misses/ref)

il1.repl_rate 0.0000 # replacement rate (i.e., repls/ref)

il1.wb_rate 0.0000 # writeback rate (i.e., wrbks/ref)

il1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref) *dl1.accesses 11153897 # total number of accesses*

dl1.hits 11099651 # total number of hits

dl1.misses 54246 # total number of misses

dl1.replacements 53222 # total number of replacements

dl1.writebacks 37897 # total number of writebacks

dl1.invalidations 0 # total number of invalidations

dl1.miss_rate 0.0049 # miss rate (i.e., misses/ref)

dl1.repl_rate 0.0048 # replacement rate (i.e., repls/ref)

dl1.wb_rate 0.0034 # writeback rate (i.e., wrbks/ref)

dl1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)

ul2.accesses 92638 # total number of accesses

ul2.hits 63100 # total number of hits

ul2.misses 29538 # total number of misses

ul2.replacements 13154 # total number of replacements

ul2.writebacks 12741 # total number of writebacks

ul2.invalidations 0 # total number of invalidations

ul2.miss_rate 0.3189 # miss rate (i.e., misses/ref)

ul2.repl_rate 0.1420 # replacement rate (i.e., repls/ref)

ul2.wb_rate 0.1375 # writeback rate (i.e., wrbks/ref)

ul2.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)

ld_text_base 0x0120000000 # program text (code) segment base

ld_text_size 106496 # program text (code) size in bytes

ld_data_base 0x0140000000 # program initialized data segment base

ld_data_size 71264 # program init'ed `.data' and uninit'ed `.bss' size in bytes

ld_stack_base 0x011ff9b000 # program stack segment base (highest address in stack)

ld_stack_size 16384 # program initial stack size *ld_prog_entry 0x01200059c0 # program entry point (initial PC)*

ld_environ_base 0x011ff97000 # program environment base address address

ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian

mem.page_count 182 # total number of pages allocated

mem.page_mem 1456k # total size of memory pages allocated

mem.ptab_misses 454294 # total first level page table misses

mem.ptab_accesses 73719151 # total page table accesses

mem.ptab_miss_rate 0.0062 # first level page table miss rate

CPI = 1.05684953

GO BENCHMARKS

{cs6304-32:~/Project_1/simplesim-3.0} ./sim-cache -cache:dl1 dl1:512:64:2:f -cache:il1 il1:512:64:2:f -cache:il2 dl2 -cache:dl2 ul2:16384:64:1:f -tlb:itlb none -tlb:dtlb none benchmarks/go.alpha 50 9 benchmarks/2stone9.in > OUT

*sim: ** simulation statistics ***

sim_num_insn 545812708 # total number of instructions executed

sim_num_refs 211690635 # total number of loads and stores executed

sim_elapsed_time 57 # total simulation time in seconds

sim_inst_rate 9575661.5439 # simulation speed (in insts/sec)

il1.accesses 545812708 # total number of accesses

il1.hits 545098009 # total number of hits

il1.misses 714699 # total number of misses

il1.replacements 713675 # total number of replacements

il1.writebacks 0 # total number of writebacks

il1.invalidations	*0 # total number of invalidations*
il1.miss_rate	*0.0013 # miss rate (i.e., misses/ref)*
il1.repl_rate	*0.0013 # replacement rate (i.e., repls/ref)*
il1.wb_rate	*0.0000 # writeback rate (i.e., wrbks/ref)*
il1.inv_rate	*0.0000 # invalidation rate (i.e., invs/ref)*
dl1.accesses	*213788508 # total number of accesses*
dl1.hits	*213579212 # total number of hits*
dl1.misses	*209296 # total number of misses*
dl1.replacements	*208272 # total number of replacements*
dl1.writebacks	*95533 # total number of writebacks*
dl1.invalidations	*0 # total number of invalidations*
dl1.miss_rate	*0.0010 # miss rate (i.e., misses/ref)*
dl1.repl_rate	*0.0010 # replacement rate (i.e., repls/ref)*
dl1.wb_rate	*0.0004 # writeback rate (i.e., wrbks/ref)*
dl1.inv_rate	*0.0000 # invalidation rate (i.e., invs/ref)*
ul2.accesses	*1019528 # total number of accesses*
ul2.hits	*927360 # total number of hits*
ul2.misses	*92168 # total number of misses*
ul2.replacements	*75784 # total number of replacements*
ul2.writebacks	*25726 # total number of writebacks*
ul2.invalidations	*0 # total number of invalidations*
ul2.miss_rate	*0.0904 # miss rate (i.e., misses/ref)*
ul2.repl_rate	*0.0743 # replacement rate (i.e., repls/ref)*
ul2.wb_rate	*0.0252 # writeback rate (i.e., wrbks/ref)*
ul2.inv_rate	*0.0000 # invalidation rate (i.e., invs/ref)*
ld_text_base	*0x0120000000 # program text (code) segment base*
ld_text_size	*376832 # program text (code) size in bytes*

ld_data_base 0x0140000000 # program initialized data segment base

ld_data_size 612032 # program init'ed `.data' and uninit'ed `.bss' size in bytes

ld_stack_base 0x011ff9b000 # program stack segment base (highest address in stack)

ld_stack_size 16384 # program initial stack size

ld_prog_entry 0x0120007bb0 # program entry point (initial PC)

ld_environ_base 0x011ff97000 # program environment base address address

ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian

mem.page_count 246 # total number of pages allocated

mem.page_mem 1968k # total size of memory pages allocated

mem.ptab_misses 1656511 # total first level page table misses

mem.ptab_accesses 1520170656 # total page table accesses

mem.ptab_miss_rate 0.0011 # first level page table miss rate

CPI = 1.01521279

PART 3

This part, given the size of L1 and L2 caches, asks us to take various possible combinations and figure out the optimal out of them. To do that we consider CPIs of all possible cache configurations by changing the following:

1. Associativity: We consider Direct-mapped caches,2-way set associative caches,4-way set associative caches, 8-way set associative caches and fully associative caches for both L1 and L2.
2. Block Size: We consider cache Block sizes to be either 32 Bytes or 64 Bytes.
3. Replacement Policy: We consider the three replacement policies namely; FIFO, LRU and Random for each case.
4. We also consider cache splits to be either fully unified caches, fully separate instruction and data caches (both L1 and L2 are separate) or L1 separate and L2 unified cache.

We generate data sets for all combinations by changing the above mentioned parameters, and then analyze them for each benchmark to finally arrive at an optimal cache configuration

Formulae :

a) Separate L1 cache & Separate L2 Cache:

CPI = CPI_{ideal} + 5* (L1InsMissRate * (L1InsAccess/Total Ins) +

L1DataMissRate * (L1DataAccess/Total Ins)) + 40 * (L2InsMissRate * (L2 InsAccess/Total Ins) + L2data MissRate*(L2 Data Access/Total Ins))

b) Separate L1 Cache & Unified L2 cache:

CPI = CPI $_{ideal}$ + 5* (L1InsMissRate * (L1 Ins Access/Total Ins) + L1DataMissRate * (L1 Data access/total Ins)) + 40 * (L2MissRate * (L2 access/Total Ins))

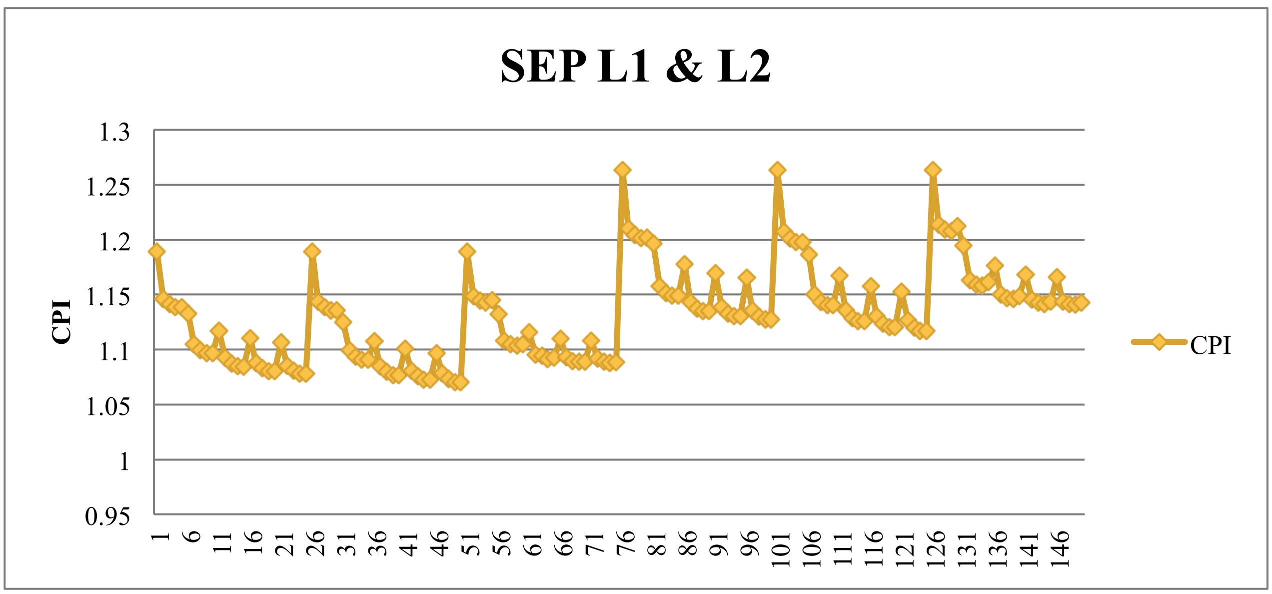
c) Unified L1 Cache & Unified L2 Cache:

CPI = CPI = CPI ideal + 5* (L1MissRate * (L1 Access/Total Ins)) + 40 * (L2MissRate * (L2 access/Total Ins)

The plots for each of the benchmarks on each cache split are given below:

GCC BENCHMARKS

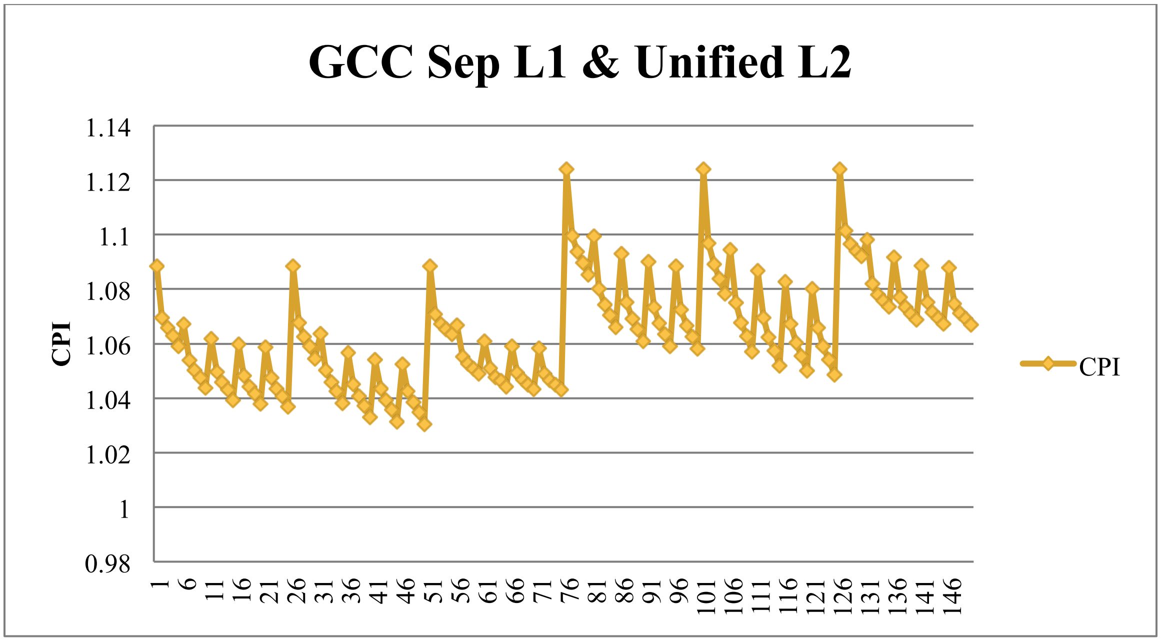
SEPARATE L1 AND L2 CACHE



OPTIMAL CONFIGURATION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	FULL	FULL	1.070019525

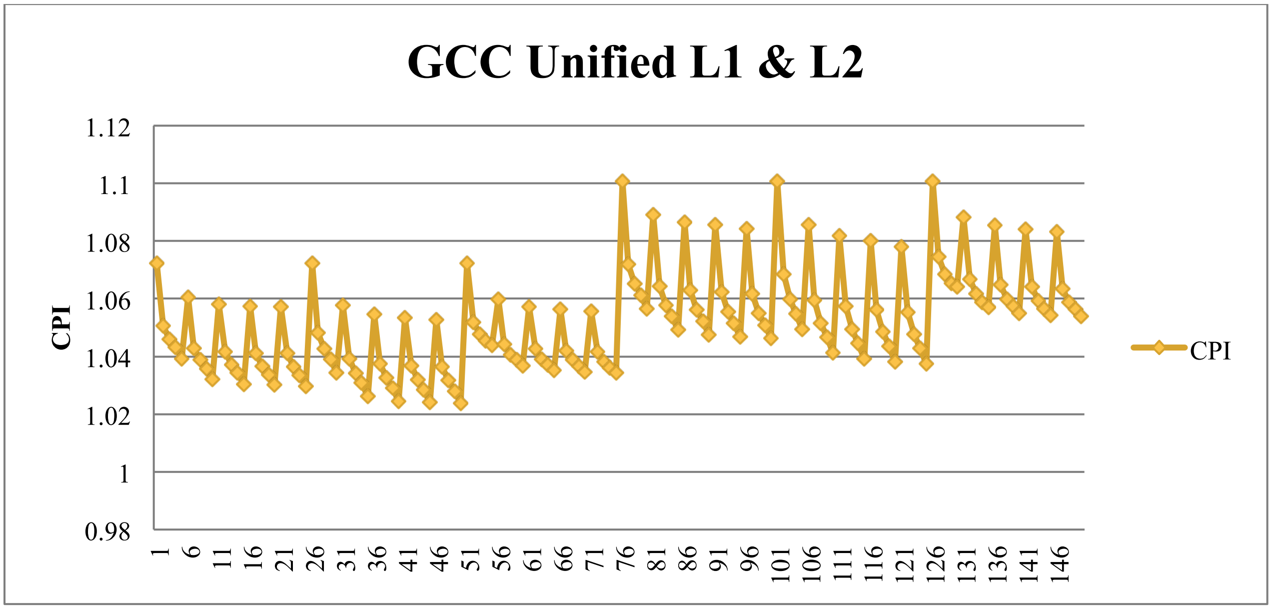
SEPARATE L1 AND UNIFIED L2 CACHE



OPTIMAL CONFIGURATION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	FULL	FULL	1.030414411

UNIFIED L1 AND L2 CACHE

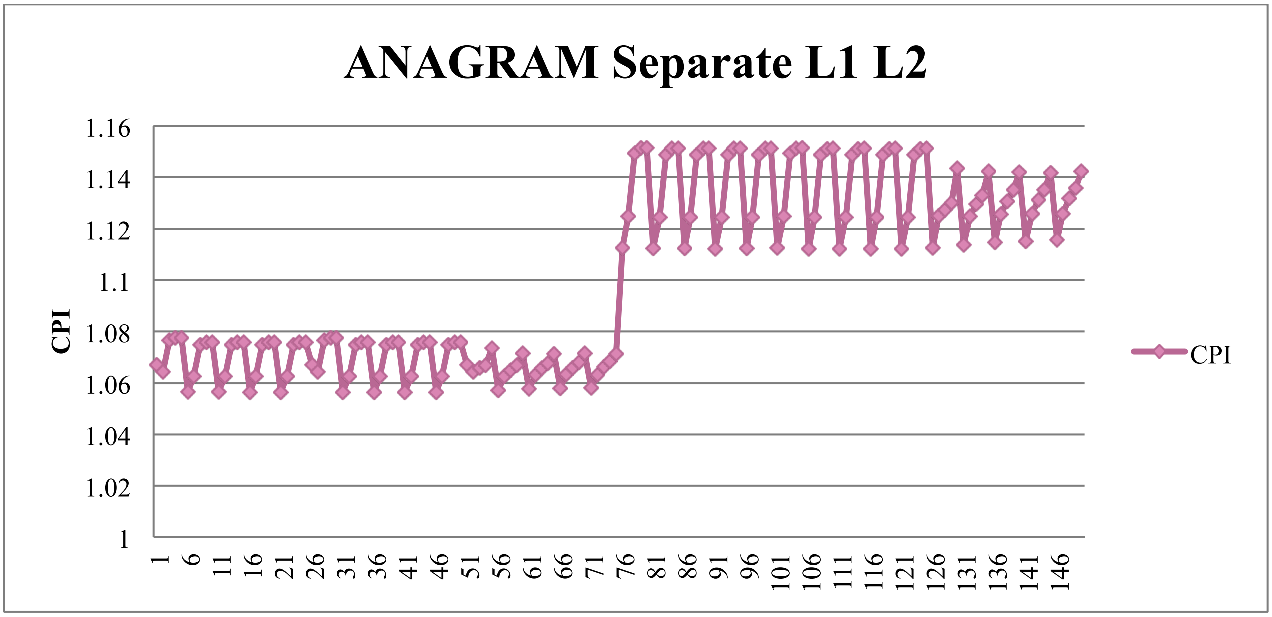


OPTIMAL CONFIGURATION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	FULL	FULL	1.02370471

ANAGRAM BENCHMARKS

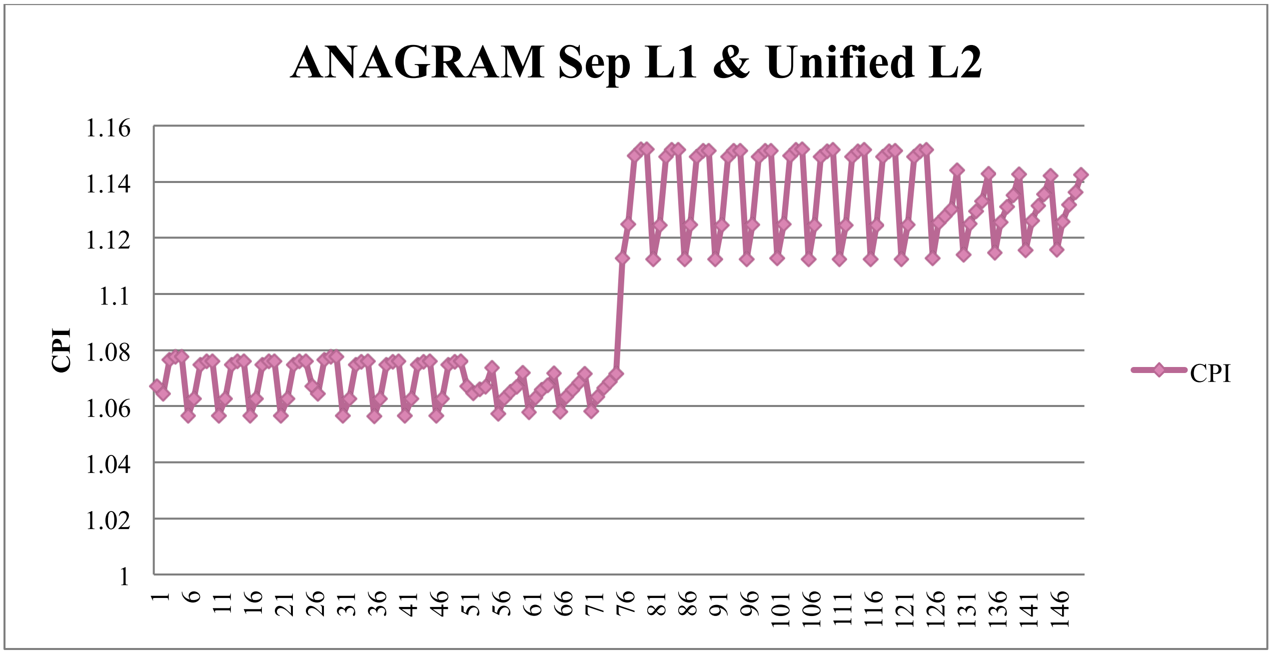
SEPARATE L1 AND L2 CACHE



OPTIMAL CONFIGURATION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	8	1	1.05649295

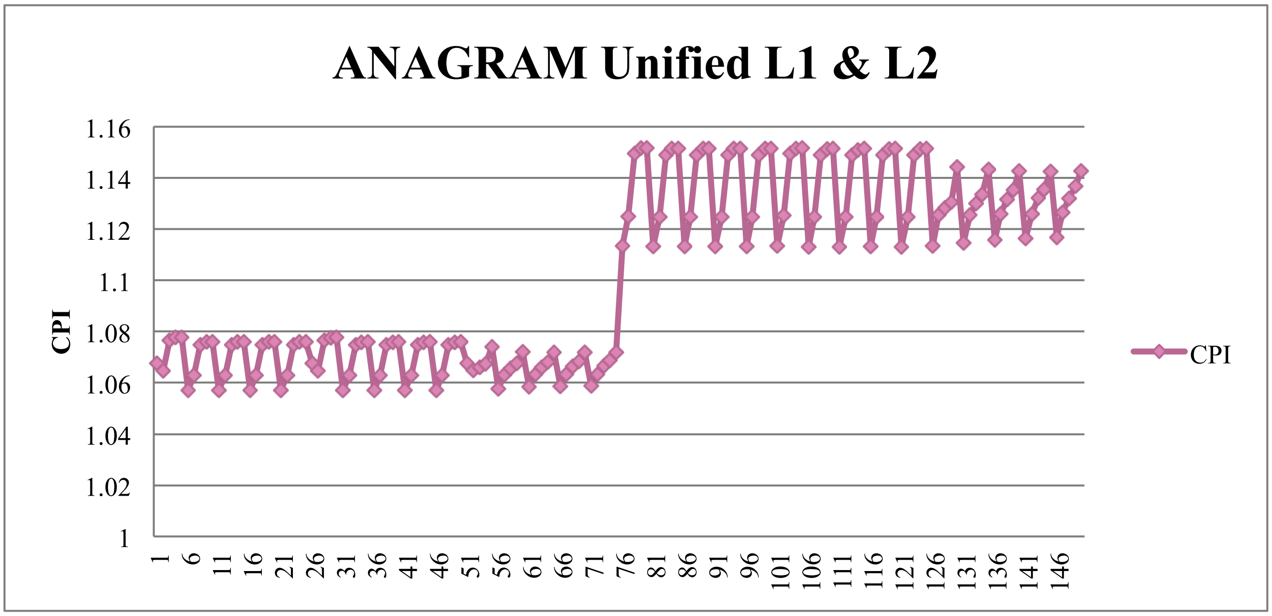
SEPARATE L1 AND UNIFIED L2 CACHE



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	4	1	1.056531875

UNIFIED L1 AND L2 CACHE

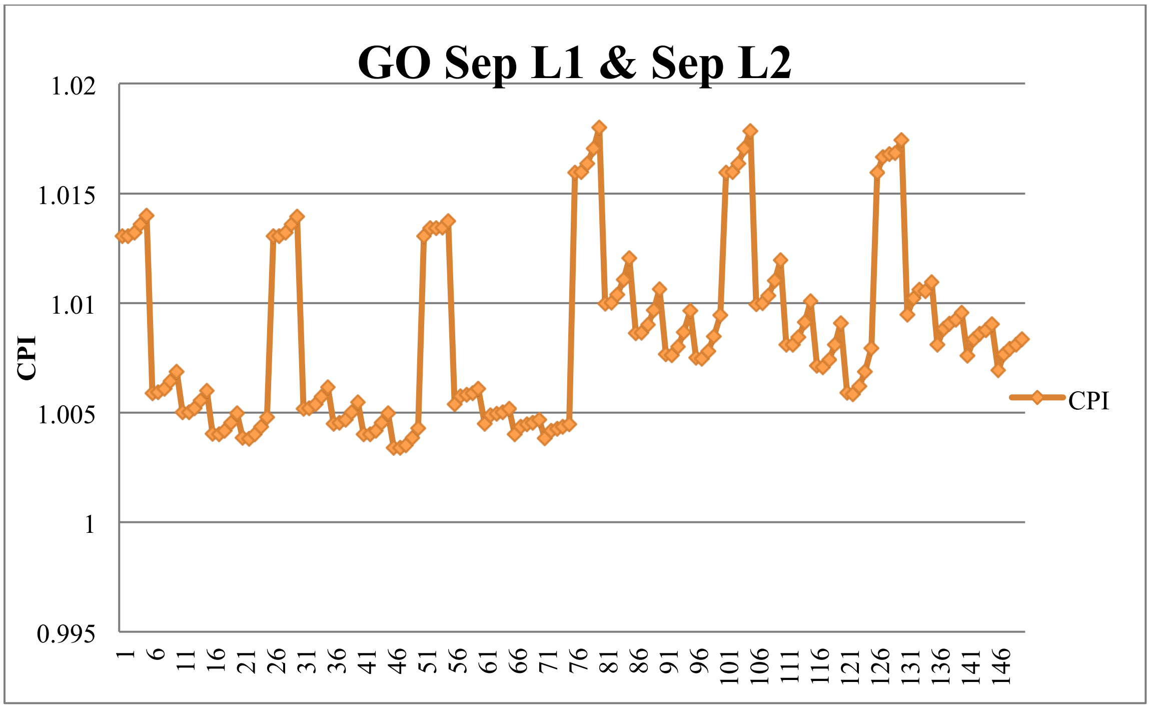


OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	4	1	1.057093017

GO BENCHMARKS

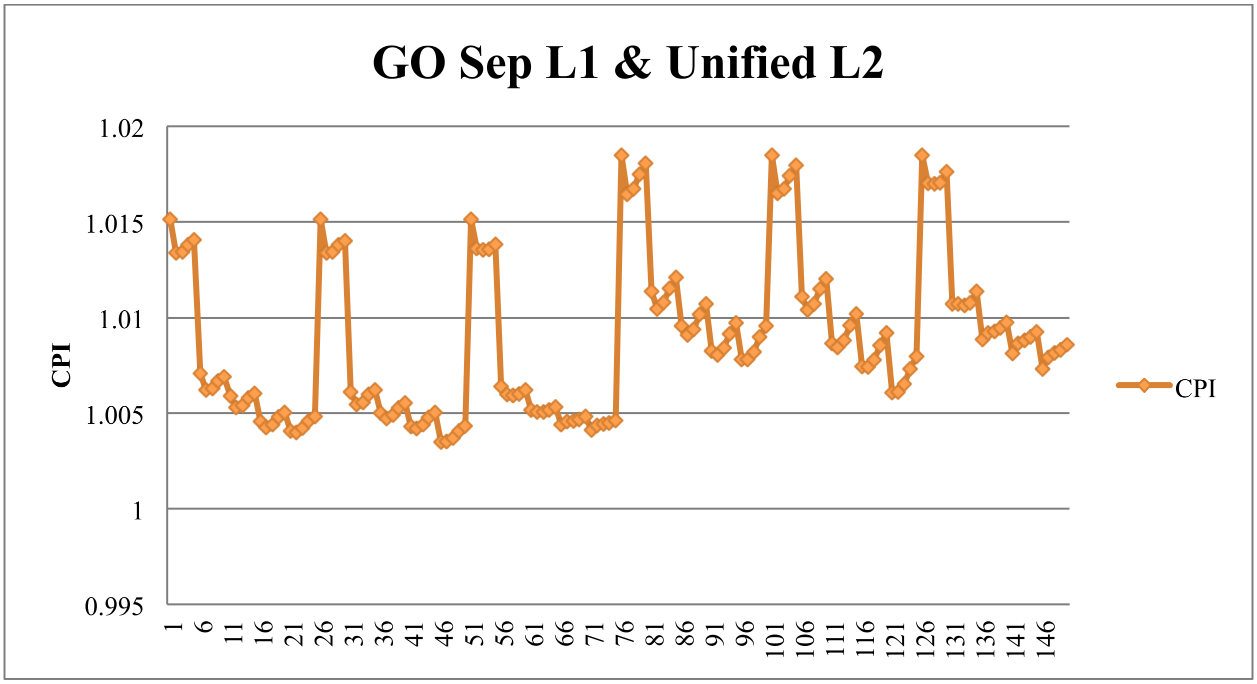
SEPARATE L1 AND L2 CACHE



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	Full	2	1.00337697

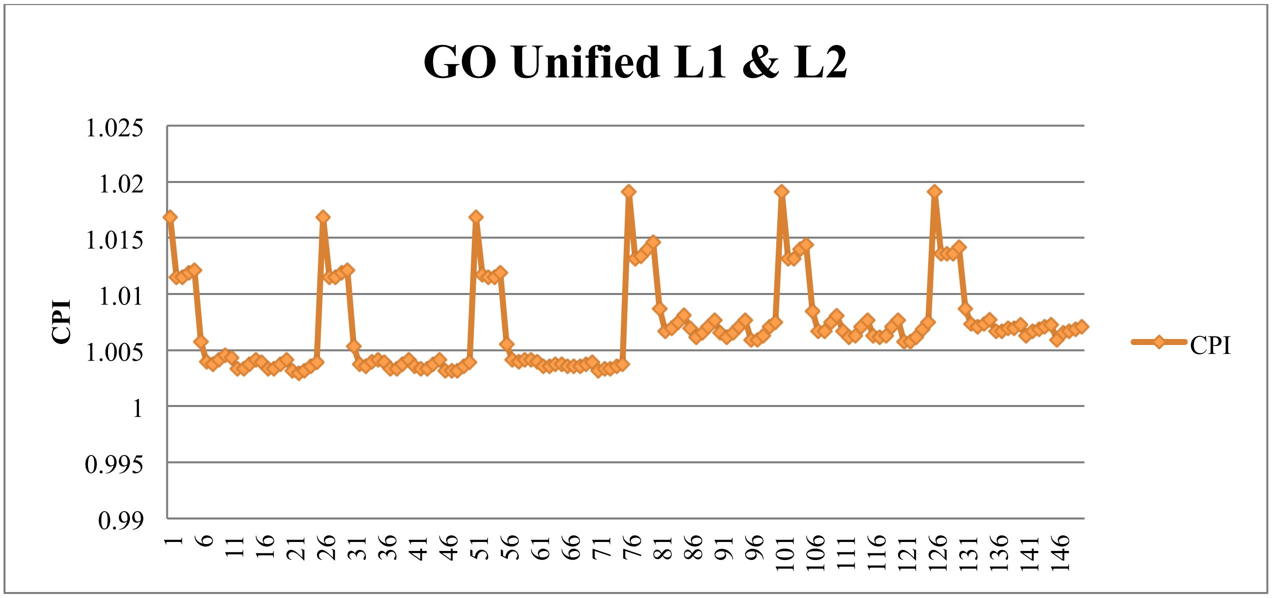
SEPARATE L1 AND UNIFEID L2 CACHE



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
32	F	Full	1	1.003491394

UNIFEID L1 AND L2 CACHE



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOCIATIVITY(L1)	ASSOCIATIVITY(L2)	CPI
64	F	Full	2	1.00294521

FINAL OPTIMAL CONFIGURATIONS(comparing above values):

GCC BENCHMARKS: Unified and Fully-associative L1 and L2 cache, of block size 32 bytes, with FIFO replacement policy.

ANAGRAM BENCHMARKS: Separate L1 8-way set associative cache and L2 direct mapped cache, of block size 32bytes and FIFO replacement policy.

GO BENCHMARKS: Unified Fully associative L1 cache and unified 2-way associative L2 cache of block size 64 bytes and FIFO replacement policy.

PART 4

DEFINING COST FUNCTION

Cost of a cache is an important factor while designing a memory hierarchy. Though lower CPI is a desired criterion, cost plays a vital role in determining the optimal CPI. A tradeoff of CPI should be made to acquire lower cost which is the ultimate aim of manufacturers. The following are the parameters considered while determining the cost function:

- Block Size
- Cache size
- Associativity
- Cache splitting
- Replacement policy

Block Size: Block size is assumed to affect the cost because, throughout our tests, we have assumed the miss penalty to be constant irrespective of the block size. But as the block size increases, to keep the miss penalty constant we would need additional hardware as more bytes of data need to be moved in the same time.

Cache Size: This accounts for the major percentage of cost. As the size of the cost increases the cost also increases. We assume a that the size of the cache accounts for 60% of the total 100% cost. L1 caches are of higher cost than L2 caches because they are faster and operate at the CPU clock cycle. L2 caches are larger than L1 caches but are slower than L2 caches. But comparatively cost of L1 caches is higher than L2 caches.

For Block Size of 64 bytes:

L1 – 35

L2 – 25

For Block Size of 32 Bytes:

L1 – 25

L2 – 15

Associativity : The next important factor which plays a vital role in determining the cost of a cache is associativity. As the associativity increases the cost also increases because of hardware complexity. Lets assume a weightage of 20% for associativity

1 – 1.25

2 – 2.5

4 – 5

8 – 10

Fully – 20

Cache Splitting: There are three main types of cache hierarchy

L1 separate- L2 separate

L1 separate-L2 unified

L1 unified-L2 unified

Separate caches for instruction and data require more hardware which eventually increases the cost. L1 caches are expensive than L2 because they are much faster and operate at CPU clock cycles.

L1 separate – 3.75

L2 separate –3.25

L1 unified – 2.5

L2 unified – 2

Replacement Policy:

There are three types of replacement policies namely, LRU , FIFO and Random. The hardware required for LRU is much complex and costlier than the other two replacement policies with random policy having the lowest cost. An overall weightage of 8.5 % is assumed.

LRU – 2.95

FIFO –2.85

Random –2.7

Thus the cost function can be defined as follows

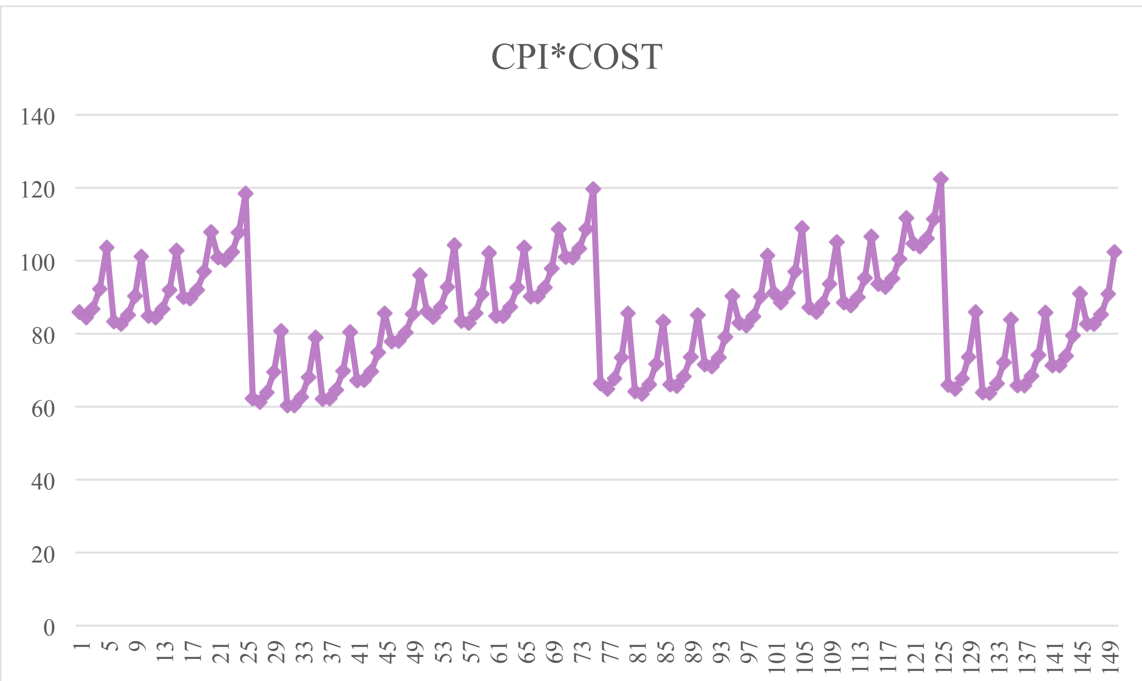
Cost = Weight(CacheSize for Specific block size L1) + Weight(CacheSize for Specific block size L2) + Weight(Cache Splitting L1) + Weight(Cache Splitting L2) + Weight(Associativity L1) + Weight(Associativity L2) + Weight(Replacement Policy)

PART 5

The various costs for each benchmark on various cache splits have been multiplied with the CPI, to give graphs of COST*CPI for different cache splits in different benchmarks. This is done so that the optimal configuration can be found out from the data set, after considering the costs, generated for each combination using the cost function in part 4.

GCC BENCHMARKS:

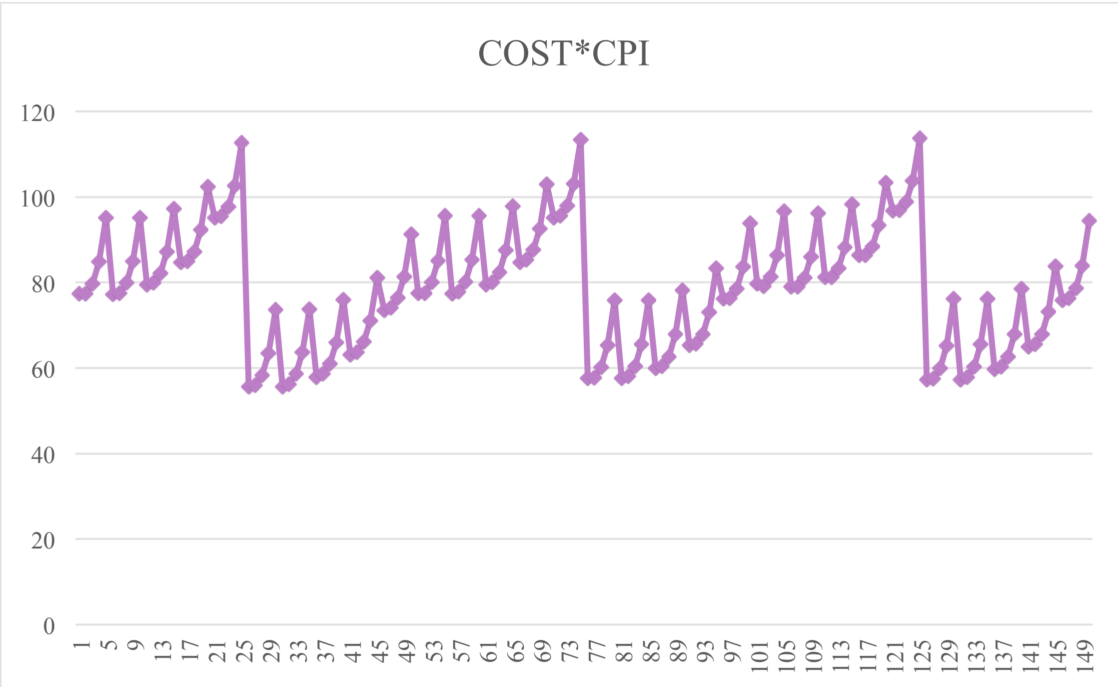
SEPARATE L1 AND L2 CACHES:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	2	2	***1.099160515***	***60.28895423***

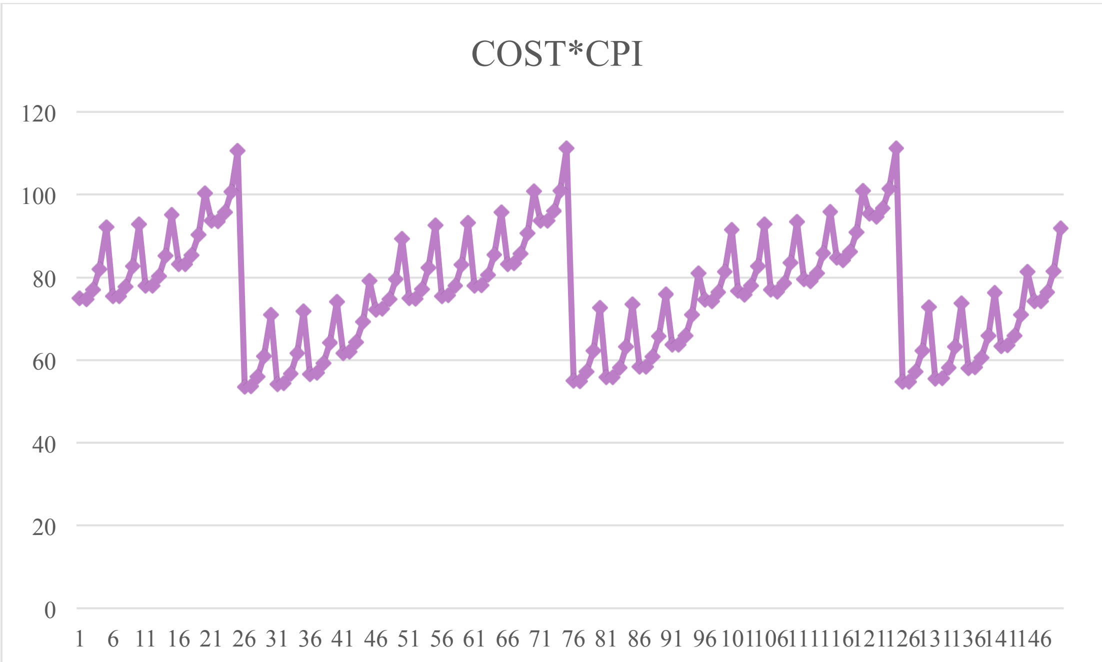
SEPARATE L1 AND UNIFIED L2 CACHE:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	1	1	***1.088327483***	***55.61353436***

UNIFEID L1 AND UNIFIED L2 CACHE:

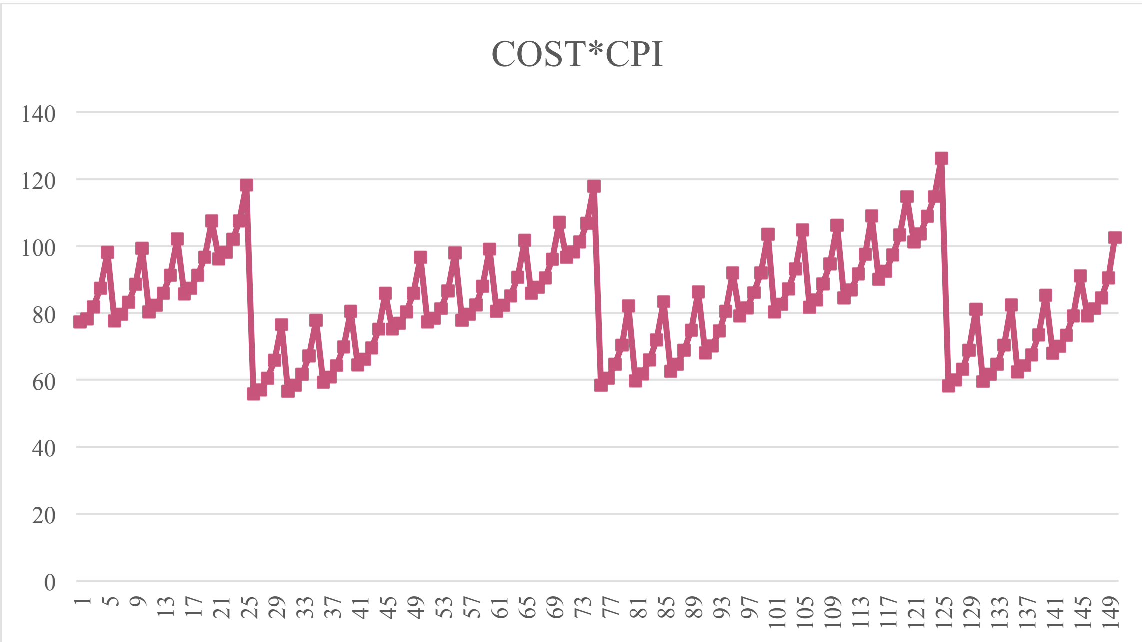


OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	1	1	***1.072390172***	***53.45865009***

ANAGRAM BENCHMARKS:

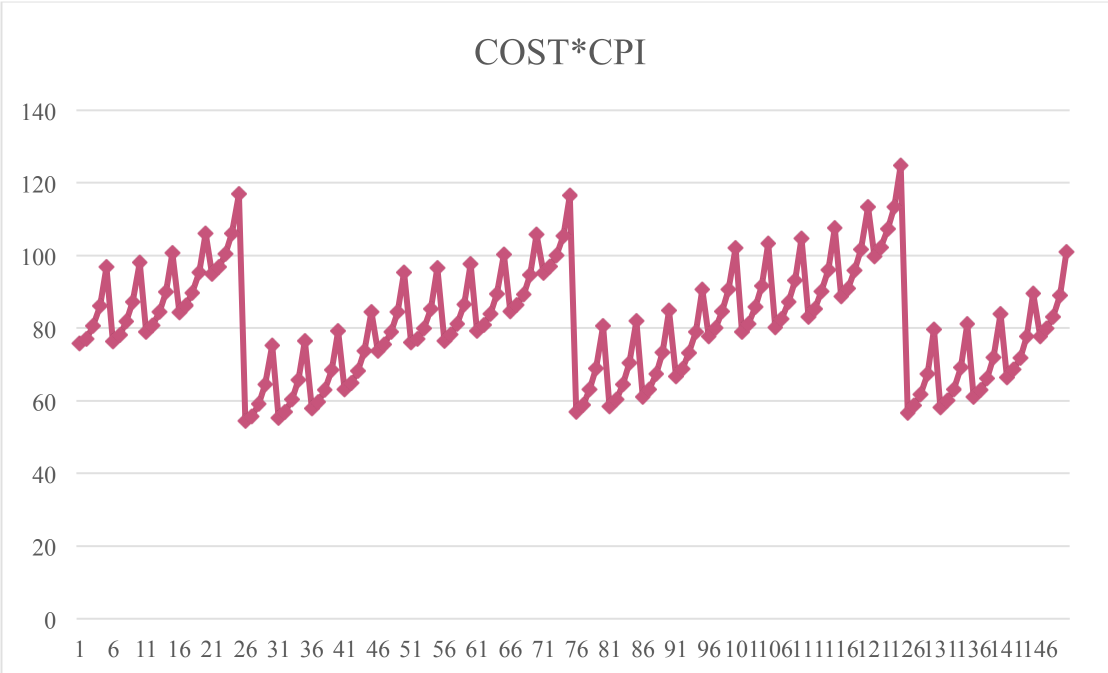
SEPARATE L1 AND L2 CACHES:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	1	1	***1.067023514***	***55.85868094***

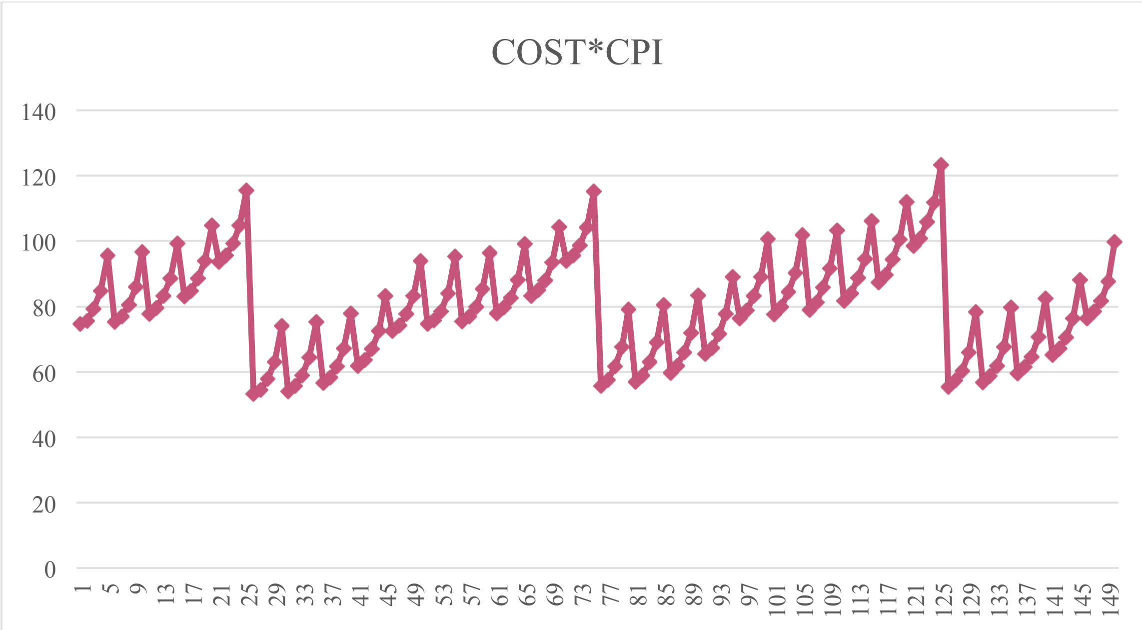
SEPARATE L1 AND UNIFEID L2:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	1	1	***1.067062466***	***54.52689199***

UNIFIED L1 AND UNIFIED L2 CACHES:

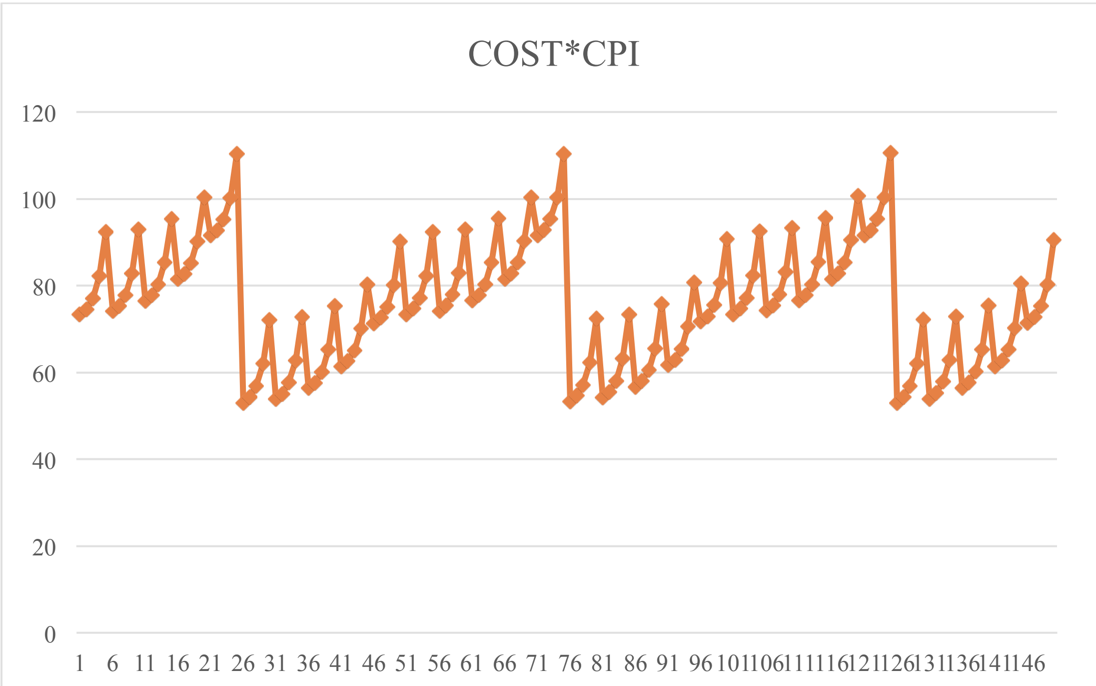


OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	1	1	***1.067626056***	***53.22115889***

GO BENCHMARKS:

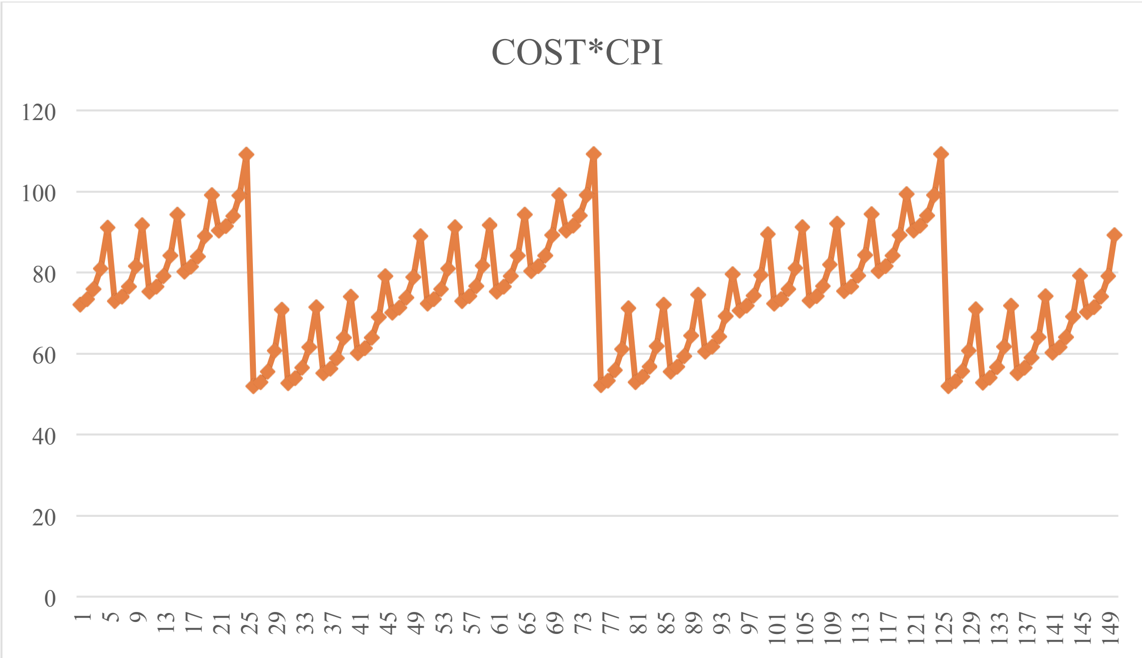
SEPARATE L1 AND L2 CACHES:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	R	1	1	***1.015948189***	***53.03249549***

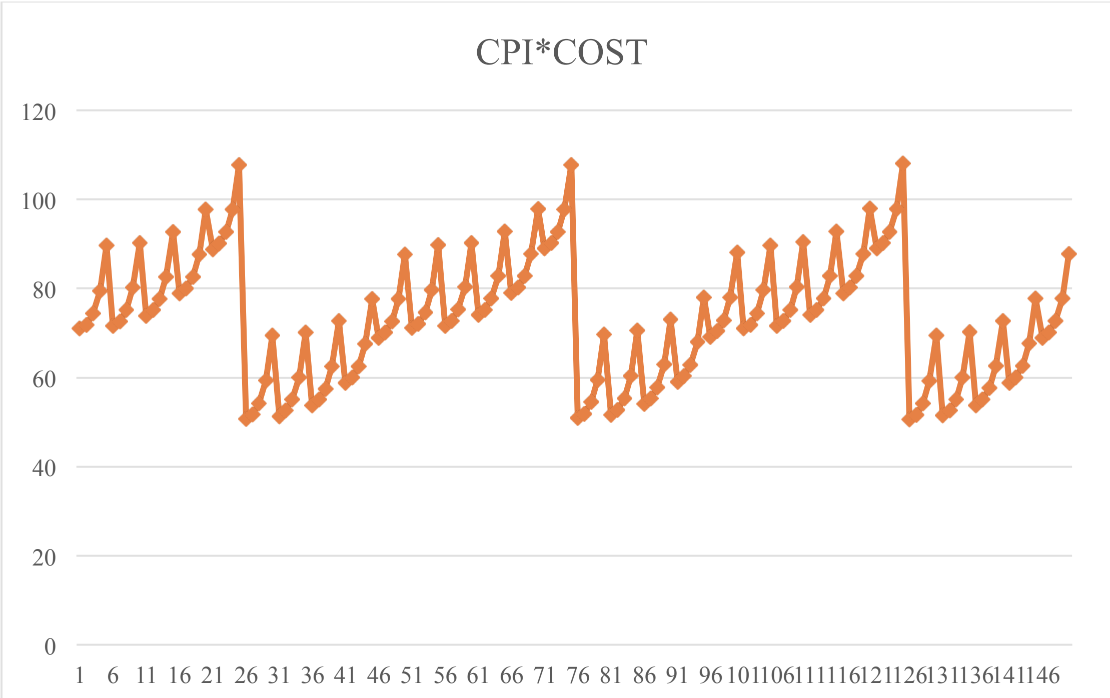
L1 SEPARATE AND L2 UNIFIED:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	F	1	1	***1.01515293***	***51.87431474***

L1 UNIFEID AND L2 UNIFIED CACHE:



OPTIMAL SOLUTION:

BLOCK SIZE	REPLACEMENT	ASSOC L1	ASSOC L2	CPI	COST*CPI
32	R	1	1	***1.019103246***	***50.64943134***

FINAL OPTIMAL SOLUTIONS FOR DIFFERENT BENCHMARKS:

GCC BENCHMARKS: L1 DIRECT-MAPPED,L2 DIRECT MAPPED, BLOCK SIZE 32 BYTES AND REPLACEMENT POLICY IS FIFO AND FULLY UNIFIED CACHES. CPI*COST = 53.45865

ANAGRAM BENCHMARKS: L1 DIRECT-MAPPED,L2 DIRECT MAPPED, BLOCK SIZE 32 BYTES AND REPLACEMENT POLICY IS FIFO AND FULLY UNIFIED CACHES. CPI*COST = 53.22116

GO BENCHMARKS: L1 DIRECT-MAPPED,L2 DIRECT MAPPED, BLOCK SIZE 32 BYTES AND REPLACEMENT POLICY IS FIFO AND FULLY UNIFEID CACHES. CPI*COST = 50.64943

FINAL OPTIMAL SOLUTION ALL BENCHMARKS COMBINED:

L1 DIRECT-MAPPED,L2 DIRECT MAPPED, BLOCK SIZE 32 BYTES AND REPLACEMENT POLICY IS FIFO AND FULLY UNIFEID CACHES. CPI*COST = 50.64943. (GO BENCHMARKS).

OPTIMAL CPI IS 1.019103246

APPENDIX

SCRIPTS USED:

The Datasets for all the computations were generated using 9 shell scripts. One for each of fully unified, fully separate, and L2 unified on GCC,GO and ANAGRAM benchmarks. Since, the scripts are quite similar with minimal changes, we add 3 scripts below used for Fully unified L1 and L2 cache simulation across the 3 Benchmarks.

UNIFIED_GCC.SH

```
#!/bin/bash


assoc_128_64=( 1 2 4 8 2048 )

#echo ${assoc_128_64[4]}

assoc_128_32=( 1 2 4 8 4096 )

assoc_1m_64=( 1 2 4 8 16384 )

assoc_1m_32=( 1 2 4 8 32768 )

sets_128_64=( 2048 1024 512 256 1 )

sets_128_32=( 4096 2048 1024 512 1 )

sets_1m_64=( 16384 8192 4096 2048 1 )

sets_1m_32=( 32768 16384 8192 4096 1 )

blk_size=( 64 32 )

replacement=( 'f' 'l' 'r' )
```

```
#res="./sim-cache -cache:dl1
dl1:"${sets_128_64[0]}":"${blk_size[0]}":"${assoc_128_64[0]}":"${replacement[0
]}" -cache:il1
il1:"${sets_128_64[0]}":"${blk_size[0]}":"${assoc_128_64[0]}":"${replacement[0
]}" -cache:dl2
dl2:"${sets_1m_64[0]}":"${blk_size[0]}":"${assoc_1m_64[0]}":"${replacement[0]
}" -cache:il2
il2:"${sets_1m_64[0]}":"${blk_size[0]}":"${assoc_1m_64[0]}":"${replacement[0]
}" -tlb:itlb none -tlb:dtlb none -redir:sim ../my_tmp1 benchmarks/cc1.alpha -O
benchmarks/1stmt.i"

#eval $res


#For each value in blk_size, when blk_size is 64 for each value in replacement and for each value in assoc_128_64 run for each value of assoc_1m_64 and sets_1m_64

#The code below will handle only separate data and instruction cache


for blk in "${blk_size[@]}"

do

        #echo $blk

        for repl in "${replacement[@]}"

        do

                #echo $repl

                if [ $blk -eq 64 ]

                then

                        set_counter1=0

                        for assoc in "${assoc_128_64[@]}"
```

```
do
        #echo $assoc
        #echo "set_counter1="$set_counter1
        #echo ${sets_128_64[$set_counter1]}
        #((set_counter1++))
        set_counter2=0
        for assoc2 in "${assoc_1m_64[@]}"
        do
                #echo $assoc2  --------> Add the command for 64
bit here
                #echo "set_counter2="$set_counter2
                #echo ${sets_1m_64[$set_counter2]}
                res="./sim-cache -cache:dl1
ul1:"${sets_128_64[$set_counter1]}":"$blk":"$assoc":"$repl" -cache:dl2
ul2:"${sets_1m_64[$set_counter2]}":"$blk":"$assoc2":"$repl" -tlb:itlb none -
tlb:dtlb none -redir:sim ../my_tmp1 benchmarks/cc1.alpha -O benchmarks/1stmt.i"
                eval $res
                #cat ../my_tmp1 >> ../LOG_GCC_SEP
                echo "L1 Unified cache size 128KB Block-
Size="$blk" Associativity="$assoc" No.of.Sets="${sets_128_64[$set_counter1]}"
Replacement Policy="$repl" and L2 Unified Cache size 1MB Block-Size="$blk"
Associativity="$assoc2" No.of.Sets="${sets_1m_64[$set_counter2]}" Replacement
Policy="$repl"." >> ../LOG_GCC_UNI
                grep "il1.hits" ../my_tmp1 >> ../LOG_GCC_UNI
                grep "il1.misses" ../my_tmp1 >> ../LOG_GCC_UNI
```

```
grep "il1.accesses" ../my_tmp1 >> ../LOG_GCC_UNI
grep "il1.miss_rate" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul1.hits" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul2.hits" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul1.misses" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul2.misses" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul1.accesses" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul2.accesses" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul1.miss_rate" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul2.miss_rate" ../my_tmp1 >> ../LOG_GCC_UNI
echo "" >> ../LOG_GCC_UNI

rm ../my_tmp1
((set_counter2++))
done
((set_counter1++))
done
```

```
else
    set_counter3=0
    for assoc in "${assoc_128_32[@]}"
    do
        #echo $assoc
        #echo "set_counter3="$set_counter3
        #echo ${sets_128_32[$set_counter3]}
        #((set_counter3++))
        set_counter4=0
        for assoc2 in "${assoc_1m_32[@]}"
        do
            #echo $assoc2 -----------> Add the command for 32
bit here
            #echo "set_counter4="$set_counter4
            #echo ${sets_1m_32[$set_counter4]}
            res1="./sim-cache -cache:dl1
ul1:"${sets_128_32[$set_counter3]}":"$blk":"$assoc":"$repl" -cache:dl2
ul2:"${sets_1m_32[$set_counter4]}":"$blk":"$assoc2":"$repl" -tlb:itlb none -
tlb:dtlb none -redir:sim ../my_tmp1 benchmarks/cc1.alpha -O benchmarks/1stmt.i"
            eval $res1
            echo "L1 Unified cache size 128KB Block-
Size="$blk" Associativity="$assoc" No.of.Sets="${sets_128_32[$set_counter3]}"
Replacement Policy="$repl" and L2 Unified Cache size 1MB Block-Size="$blk"
Associativity="$assoc2" No.of.Sets="${sets_1m_32[$set_counter4]}" Replacement
Policy="$repl"." >> ../LOG_GCC_UNI
```

```
#cat ../my_tmp1 >> ../LOG_GCC_SEP
grep "il1.hits" ../my_tmp1 >> ../LOG_GCC_UNI
grep "il1.misses" ../my_tmp1 >> ../LOG_GCC_UNI
grep "il1.access" ../my_tmp1 >> ../LOG_GCC_UNI
grep "il1.miss_rate" ../my_tmp1 >>
../LOG_GCC_UNI
grep "ul1.hits" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul2.hits" ../my_tmp1 >> ../LOG_GCC_UNI
grep "ul1.misses" ../my_tmp1 >>
../LOG_GCC_UNI
grep "ul2.misses" ../my_tmp1 >>
../LOG_GCC_UNI
grep "ul1.accesses" ../my_tmp1 >>
../LOG_GCC_UNI
grep "ul2.accesses" ../my_tmp1 >>
../LOG_GCC_UNI
grep "ul1.miss_rate" ../my_tmp1 >>
../LOG_GCC_UNI
grep "ul2.miss_rate" ../my_tmp1 >>
../LOG_GCC_UNI
echo "" >> ../LOG_GCC_UNI

rm ../my_tmp1
((set_counter4++))
done
```

```
                ((set_counter3++))
            done
        fi
    done
done
```

UNIFEID_GO.SH

```
#!/bin/bash


assoc_128_64=( 1 2 4 8 2048 )
#echo ${assoc_128_64[4]}
assoc_128_32=( 1 2 4 8 4096 )
assoc_1m_64=( 1 2 4 8 16384 )
assoc_1m_32=( 1 2 4 8 32768 )
sets_128_64=( 2048 1024 512 256 1 )
sets_128_32=( 4096 2048 1024 512 1 )
sets_1m_64=( 16384 8192 4096 2048 1 )
sets_1m_32=( 32768 16384 8192 4096 1 )
blk_size=( 64 32 )
replacement=( 'f' 'l' 'r' )
```

#For each value in blk_size, when blk_size is 64 for each value in replacement and for each value in assoc_128_64 run for each value of assoc_1m_64 and sets_1m_64

#The code below will handle only separate data and instruction cache

```
for blk in "${blk_size[@]}"
do
    #echo $blk
    for repl in "${replacement[@]}"
    do
        #echo $repl
        if [ $blk -eq 64 ]
        then
            set_counter1=0
            for assoc in "${assoc_128_64[@]}"
            do
                #echo $assoc
                #echo "set_counter1="$set_counter1
                #echo ${sets_128_64[$set_counter1]}
                #((set_counter1++))
                set_counter2=0
                for assoc2 in "${assoc_1m_64[@]}"
                do
```

```
#echo $assoc2  --------> Add the command for 64 bit here

#echo "set_counter2="$set_counter2

#echo ${sets_1m_64[$set_counter2]}

res="./sim-cache -cache:dl1 ul1:"${sets_128_64[$set_counter1]}":"$blk":"$assoc":"$repl" -cache:dl2 ul2:"${sets_1m_64[$set_counter2]}":"$blk":"$assoc2":"$repl" -tlb:itlb none -tlb:dtlb none -redir:sim ../my_go_tmp benchmarks/go.alpha 50 9 benchmarks/2stone9.in > OUT"

eval $res

#cat ../my_go_tmp >> ../LOG_GO_SEP

echo "L1 Unified cache size 128KB Block-Size="$blk" Associativity="$assoc" No.of.Sets="${sets_128_64[$set_counter1]}" Replacement Policy="$repl" and L2 Unified Cache size 1MB Block-Size="$blk" Associativity="$assoc2" No.of.Sets="${sets_1m_64[$set_counter2]}" Replacement Policy="$repl"." >> ../LOG_GO_UNI

grep "il1.hits" ../my_go_tmp >> ../LOG_GO_UNI

grep "il1.misses" ../my_go_tmp >> ../LOG_GO_UNI

grep "il1.accesses" ../my_go_tmp >> ../LOG_GO_UNI

grep "il1.miss_rate" ../my_go_tmp >> ../LOG_GO_UNI

grep "ul1.hits" ../my_go_tmp >> ../LOG_GO_UNI

grep "ul2.hits" ../my_go_tmp >> ../LOG_GO_UNI

grep "ul1.misses" ../my_go_tmp >> ../LOG_GO_UNI
```

```
                    grep "ul2.misses" ../my_go_tmp >>
../LOG_GO_UNI
                    grep "ul1.accesses" ../my_go_tmp >>
../LOG_GO_UNI
                    grep "ul2.accesses" ../my_go_tmp >>
../LOG_GO_UNI
                    grep "ul1.miss_rate" ../my_go_tmp >>
../LOG_GO_UNI
                    grep "ul2.miss_rate" ../my_go_tmp >>
../LOG_GO_UNI
                    echo "" >> ../LOG_GO_UNI

                    rm ../my_go_tmp
                    ((set_counter2++))
                done
                ((set_counter1++))
            done
        else
            set_counter3=0
            for assoc in "${assoc_128_32[@]}"
            do
                #echo $assoc
                #echo "set_counter3="$set_counter3
                #echo ${sets_128_32[$set_counter3]}
```

```
#((set_counter3++))
set_counter4=0
for assoc2 in "${assoc_1m_32[@]}"
do
	#echo $assoc2 -----------> Add the command for 32
bit here
	#echo "set_counter4="$set_counter4
	#echo ${sets_1m_32[$set_counter4]}
	res1="./sim-cache -cache:dl1
ul1:"${sets_128_32[$set_counter3]}":"$blk":"$assoc":"$repl" -cache:dl2
ul2:"${sets_1m_32[$set_counter4]}":"$blk":"$assoc2":"$repl" -tlb:itlb none -
tlb:dtlb none -redir:sim ../my_go_tmp benchmarks/go.alpha 50 9
benchmarks/2stone9.in > OUT"
	eval $res1
	echo "L1 Unified cache size 128KB Block-
Size="$blk" Associativity="$assoc" No.of.Sets="${sets_128_32[$set_counter3]}"
Replacement Policy="$repl" and L2 Unified Cache size 1MB Block-Size="$blk"
Associativity="$assoc2" No.of.Sets="${sets_1m_32[$set_counter4]}" Replacement
Policy="$repl"." >> ../LOG_GO_UNI
	#cat ../my_go_tmp >> ../LOG_GO_SEP
	grep "il1.hits" ../my_go_tmp >> ../LOG_GO_UNI
	grep "il1.misses" ../my_go_tmp >>
../LOG_GO_UNI
	grep "il1.access" ../my_go_tmp >>
../LOG_GO_UNI
	grep "il1.miss_rate" ../my_go_tmp >>
../LOG_GO_UNI
```

```
                        grep "ul1.hits" ../my_go_tmp >> ../LOG_GO_UNI
                        grep "ul2.hits" ../my_go_tmp >> ../LOG_GO_UNI
                        grep "ul1.misses" ../my_go_tmp >>
../LOG_GO_UNI
                        grep "ul2.misses" ../my_go_tmp >>
../LOG_GO_UNI
                        grep "ul1.accesses" ../my_go_tmp >>
../LOG_GO_UNI
                        grep "ul2.accesses" ../my_go_tmp >>
../LOG_GO_UNI
                        grep "ul1.miss_rate" ../my_go_tmp >>
../LOG_GO_UNI
                        grep "ul2.miss_rate" ../my_go_tmp >>
../LOG_GO_UNI
                        echo "" >> ../LOG_GO_UNI

                        rm ../my_go_tmp
                        ((set_counter4++))
                    done
                    ((set_counter3++))
                done
            fi
    done
done
```

UNIFEID_ANAGRAM.SH

```
#!/bin/bash


assoc_128_64=( 1 2 4 8 2048 )

#echo ${assoc_128_64[4]}

assoc_128_32=( 1 2 4 8 4096 )

assoc_1m_64=( 1 2 4 8 16384 )

assoc_1m_32=( 1 2 4 8 32768 )

sets_128_64=( 2048 1024 512 256 1 )

sets_128_32=( 4096 2048 1024 512 1 )

sets_1m_64=( 16384 8192 4096 2048 1 )

sets_1m_32=( 32768 16384 8192 4096 1 )

blk_size=( 64 32 )

replacement=( 'f' 'l' 'r' )



#For each value in blk_size, when blk_size is 64 for each value in replacement and
for each value in assoc_128_64 run for each value of assoc_1m_64 and sets_1m_64

#The code below will handle only separate data and instruction cache


for blk in "${blk_size[@]}"

do

        #echo $blk
```

```
for repl in "${replacement[@]}"
do
	#echo $repl
	if [ $blk -eq 64 ]
	then
		set_counter1=0
		for assoc in "${assoc_128_64[@]}"
		do
			#echo $assoc
			#echo "set_counter1="$set_counter1
			#echo ${sets_128_64[$set_counter1]}
			#((set_counter1++))
			set_counter2=0
			for assoc2 in "${assoc_1m_64[@]}"
			do
				#echo $assoc2  --------> Add the command for 64 
bit here
				#echo "set_counter2="$set_counter2
				#echo ${sets_1m_64[$set_counter2]}
				res="./sim-cache -cache:dl1 
ul1:"${sets_128_64[$set_counter1]}":"$blk":"$assoc":"$repl" -cache:dl2 
ul2:"${sets_1m_64[$set_counter2]}":"$blk":"$assoc2":"$repl" -tlb:itlb none -
tlb:dtlb none -redir:sim ../my_anagram_tmp benchmarks/anagram.alpha 
benchmarks/words < benchmarks/anagram.in > OUT"
```

```
eval $res
#cat ../my_anagram_tmp >> ../LOG_ANAGRAM_SEP
echo "L1 Unified cache size 128KB Block-Size="$blk" Associativity="$assoc" No.of.Sets="${sets_128_64[$set_counter1]}" Replacement Policy="$repl" and L2 Unified Cache size 1MB Block-Size="$blk" Associativity="$assoc2" No.of.Sets="${sets_1m_64[$set_counter2]}" Replacement Policy="$repl"." >> ../LOG_ANAGRAM_UNI
grep "il1.hits" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "il1.misses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "il1.accesses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "il1.miss_rate" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "ul1.hits" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "ul2.hits" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "ul1.misses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "ul2.misses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "ul1.accesses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
grep "ul2.accesses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI
```

```
                    grep "ul1.miss_rate" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                    grep "ul2.miss_rate" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                    echo "" >> ../LOG_ANAGRAM_UNI

                    rm ../my_anagram_tmp
                    ((set_counter2++))
                done
                ((set_counter1++))
            done
        else
            set_counter3=0
            for assoc in "${assoc_128_32[@]}"
            do
                #echo $assoc
                #echo "set_counter3="$set_counter3
                #echo ${sets_128_32[$set_counter3]}
                #((set_counter3++))
                set_counter4=0
                for assoc2 in "${assoc_1m_32[@]}"
                do
```

#echo $assoc2 -----------> Add the command for 32 bit here

#echo "set_counter4="$set_counter4

#echo ${sets_1m_32[$set_counter4]}

res1="./sim-cache -cache:dl1 ul1:"${sets_128_32[$set_counter3]}":"$blk":"$assoc":"$repl" -cache:dl2 ul2:"${sets_1m_32[$set_counter4]}":"$blk":"$assoc2":"$repl" -tlb:itlb none -tlb:dtlb none -redir:sim ../my_anagram_tmp benchmarks/anagram.alpha benchmarks/words < benchmarks/anagram.in > OUT"

eval $res1

echo "L1 Unified cache size 128KB Block-Size="$blk" Associativity="$assoc" No.of.Sets="${sets_128_32[$set_counter3]}" Replacement Policy="$repl" and L2 Unified Cache size 1MB Block-Size="$blk" Associativity="$assoc2" No.of.Sets="${sets_1m_32[$set_counter4]}" Replacement Policy="$repl"." >> ../LOG_ANAGRAM_UNI

#cat ../my_anagram_tmp >> ../LOG_ANAGRAM_SEP

grep "il1.hits" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI

grep "il1.misses" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI

grep "il1.access" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI

grep "il1.miss_rate" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI

grep "ul1.hits" ../my_anagram_tmp >> ../LOG_ANAGRAM_UNI

```
                                        grep "ul2.hits" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        grep "ul1.misses" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        grep "ul2.misses" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        grep "ul1.accesses" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        grep "ul2.accesses" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        grep "ul1.miss_rate" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        grep "ul2.miss_rate" ../my_anagram_tmp >>
../LOG_ANAGRAM_UNI
                                        echo "" >> ../LOG_ANAGRAM_UNI

                                        rm ../my_anagram_tmp
                                        ((set_counter4++))
                              done
                              ((set_counter3++))
                    done
          fi
     done
done
```