COMPUTER ARCHITECTURE

Project 2 Report

ABSTRACT

Comparison between different cache configurations for different CPUs for all 5 SPEC benchmarks. Cost function and Evaluation function was defined to calculate the optimum cache configuration for all 5 SPEC benchmarks over 3 different CPUs.

Anmol Gautam , Soumyadeep Choudhury

AXG190014 SXC180056

Marks Distribution: 100/100

Table of Contents

Introduction Page 2-3

- Gem5
- Cache
- Types of Caches
- Associativity in Caches
- Cycles Per Instruction (CPI)
- Cost Function (CF)

Installing gem5 Page 4

Challenge and Issues Page 5-6

Modifications and Changes Page 7-9

Config.ini Page 10

Generated Stats File Page 10-11

Cost Function Page 12

Evaluation Function Page 13

Results Page 14-32

Conclusion Page 33

References Page 33

Appendix Page 34-40

Introduction

Gem5

Gem5 simulator is a platform for simulating computer system architecture. It incorporates system level architecture and processor microarchitecture.

Key Features:

- Gem5 gives four interpretation-based CPUs which shares a common high-level ISA architecture.
- A completely coordinated GPU mode.
- Integrated NoMali GPU model removes the need for software rendering.
- Implemented event-driven memory system captures the impact of current and emerging memories.
- A model that plays back elastic traces.
- Homogenous and Heterogeneous multicore model.
- Full system capacity for Alpha, ARM, SPARC and x86
- Various frameworks can be launched inside a solitary reenactment process.
- Gem5's items are masterminded in OS-obvious power and clock areas, empowering a scope of tests in power-and vitality proficiency.
- Gem5 can be effectively ran as a thread inside system event kernel, which synchronizes events and timeline.

Cache:

In processing, a cache is an equipment or programming segment that stores information with the goal that future solicitations for that information can be served quicker; the information put away in a cache may be the aftereffect of a prior calculation or a duplicate of information put away somewhere else. A cache hit happens when the mentioned information can be found in a cache, while a cache miss happens when it can't. Cache hits are served by perusing information from the store, which is quicker than recomputing an outcome or perusing from a more slow information store; consequently, the more demands that can be served from the cache, the quicker the framework performs.

To be practical and to empower proficient utilization of information, caches must be generally little. By and by, caches have substantiated themselves in numerous regions of figuring, in light of the fact that run of the mill PC applications gets to information with a high level of region of reference. Such access designs show worldly territory, where information is mentioned that has been as of late mentioned as of now, and spatial locality, where information is mentioned that is put away physically near the information that has just been mentioned.

Types of Caches:

- **Level 1** - It is a kind of memory wherein information is put away and acknowledged that are quickly put away in CPU. Most normally utilized register is collector, program counter, an address register and so forth.
- **Level 2** - It is the quickest memory which has quicker access time where information is incidentally put away for quicker access.
- **Level 3** - It is a memory on which PC works as of now. It is little in the estimate and once control is off information never again remains in this memory.

Associativity in Caches:

- **Direct Mapped Cache** - In a direct-mapped cache structure, the cache is composed into various sets with a solitary cache line for every set. In light of the location of the memory block, it can just involve a solitary cache line. The cache can be surrounded as a (n*1) segment network.
- **Fully Associative Cache** - In a Fully associative cache, the store is composed of a solitary cache set with different store lines. A memory block can involve any of the cache lines. The store associate can be encircled as (1*m) push grid.
- **Set Associative Cache** - The Set associative cache can be envisioned as a (n*m) grid. The cache is separated into 'n' sets and each set contains 'm' store lines. A memory block is first mapped onto a set and afterward set into any cache line of the set.

Cycles per Instruction (CPI):

In computer architecture, cycles per guidance (otherwise known as clock cycles per guidance, tickers per guidance, or CPI) is one part of a processor's presentation: the normal number of clock cycles per guidance for a program or program section. It is the multiplicative backward of directions per cycle.

Cost Function (CF):

A cost function is an element of information costs and yield amount whose worth is the expense of making that yield given those info costs, regularly applied using the cost bend by organizations to limit cost and expand creation productivity.

Installing Gem5

We opted to use our local machine for this project. Main reason being that we thought, that on a local machine each individual benchmark would take less time than on the server. We inferred that to be correct based on the run time of bzip2 and sjeng provided in the project description and the time we got on our local machine while testing our setup for the first time.

Our local machine specifications are as follow → Core i5 (7th generation), overclockable up to 3.22 Ghz, 20 GB of DRAM.

Challenges and Issues

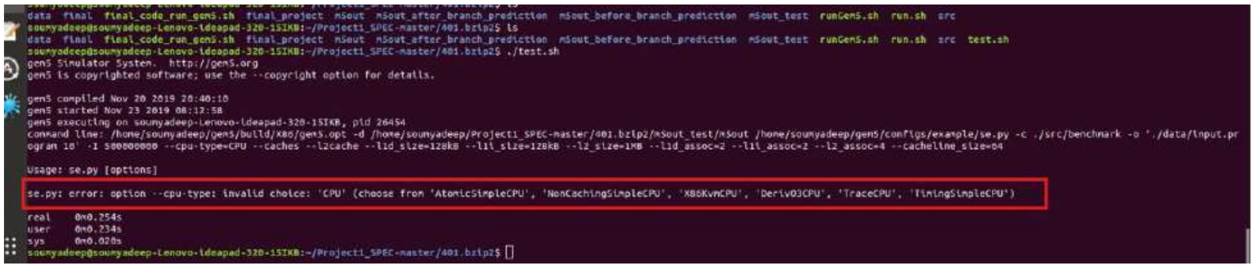
Mentioned below are the challenges and Issues faced during project 1. New challenges and New Issues describes the ones we faced during project 2. We are mentioning the old challenges as they were not specific to the project 1 but rather generic issues that can be faced on any system.

New Challenges:

- For each benchmark we experimented over a large range of cache configurations. Therefore, to expedite this process we dedicated all 8 cores of our machine to the experiments using the following command.
    - taskset -cuda 0,1,2,3,4,5,6,7 --parallel -j9 ./final_run_gem5.sh

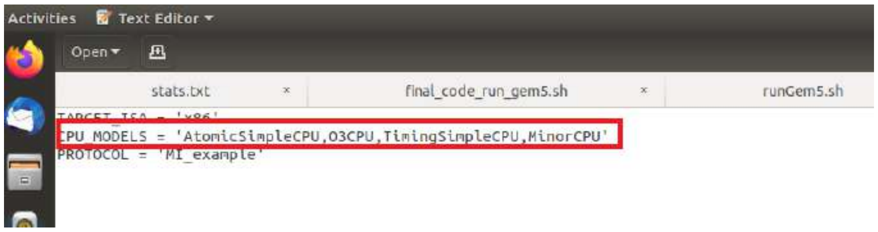
New Issues:

- DerivO3CPU and MinorCPU was not simulated in gem5 so while running the SPEC benchmarks we couldn't find the object files of these executable CPUs for our experiment.



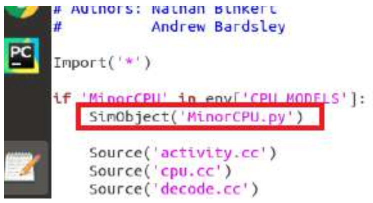
```
data  final  final_code_run_gem5.sh  final_project  m5out  m5out_after_branch_prediction  m5out_before_branch_prediction  m5out_test  runGem5.sh  run.sh  src
souryadeep@souryadeep-Lenovo-ideapad-320-15IKB:~/Project1_SPEC-master/401.bzip2$ ls
data  final  final_code_run_gem5.sh  final_project  m5out  m5out_after_branch_prediction  m5out_before_branch_prediction  m5out_test  runGem5.sh  run.sh  src  test.sh
souryadeep@souryadeep-Lenovo-ideapad-320-15IKB:~/Project1_SPEC-master/401.bzip2$ ./test.sh
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Nov 20 2019 20:40:10
gem5 started Nov 23 2019 08:12:58
gem5 executing on souryadeep-Lenovo-ideapad-320-15IKB, pid 26454
command line: /home/souryadeep/gem5/build/X86/gem5.opt -d /home/souryadeep/Project1_SPEC-master/401.bzip2/m5out_test/m5out /home/souryadeep/gem5/configs/example/se.py -c ./src/benchmark -o './data/input.pr
ogram 10' -I 500000000 --cpu-type=CPU --caches --l2cache --l1d_size=128kB --l1i_size=128kB --l2_size=1MB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=4 --cacheline_size=64

Usage: se.py [options]

se.py: error: option --cpu-type: invalid choice: 'CPU' (choose from 'AtomicSimpleCPU', 'NonCachingSimpleCPU', 'X86KvmCPU', 'DerivO3CPU', 'TraceCPU', 'TimingSimpleCPU')

real    0m0.254s
user    0m0.234s
sys     0m0.020s
souryadeep@souryadeep-Lenovo-ideapad-320-15IKB:~/Project1_SPEC-master/401.bzip2$
```

- To fix the above issues we made changes in gem5 in the following files.
    - gem5/build_opts/X86



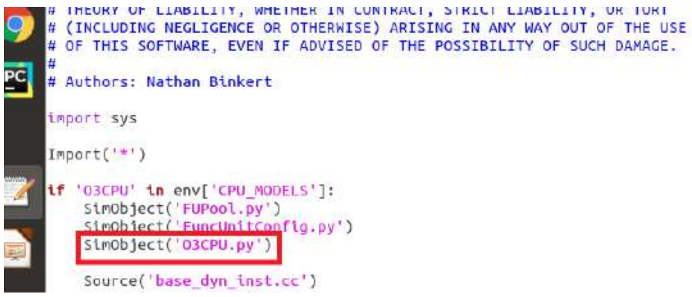
```
TARGET_ISA = 'x86'
CPU_MODELS = 'AtomicSimpleCPU,O3CPU,TimingSimpleCPU,MinorCPU'
PROTOCOL = 'MI_example'
```

    - gem5/src/cpu/minor/SConscript



```
# Authors: Nathan Binkert
#          Andrew Bardsley

Import('*')

if 'MinorCPU' in env['CPU_MODELS']:
    SimObject('MinorCPU.py')

    Source('activity.cc')
    Source('cpu.cc')
    Source('decode.cc')
```

    - gem5/src/cpu/o3/SConscript



```
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
# Authors: Nathan Binkert

import sys

Import('*')

if 'O3CPU' in env['CPU_MODELS']:
    SimObject('FUPool.py')
    SimObject('FuncUnitConfig.py')
    SimObject('O3CPU.py')

    Source('base_dyn_inst.cc')
```

Old Challenges:

- To build on gem5, we had to install dependencies for which we referred to the official documentation of gem5 where required softwares and libraries are mentioned.
- Simply installing dependencies as mentioned in manual pages, on our local system, did not work
    - This happened because in our system we are using deep learning cuda network with python 3.6 libraries for Machine Learning.
    - We are using cuda10.1, Python 3.6, cudnn 2.1., cross compiler is upgraded to latest version and kernel version had been upgraded to > 4.3.3. Official manual page of gem5 suggests using kernel version between 3.1.3 to 4.1.1.
    - Ubuntu version is 18.04.5 LTS. Libraries suggested in gem5 manual page are not directly compatible for this Ubuntu version.
- To solve the above-mentioned issues, we created two instances of the kernel and installed the following files:
    - Softwares : SCons1, zlib1, m41, protobuf1, pydot1
    - Dependencies: libfdt1, dnet1, iostream3, libelf1, PLY1, x11ksyms1, fputils1
- Cross compiler issue was solved by aliasing UEFI mode in boot system so that simulator can run on CPU kernel instead of GPU (default for cuda network).

Old Issues:

- Faced an issue of <boost/bind.hpp> while building gem5. This was because of kernel version issue. This was solved by reducing kernel version.
- Issue faced of parsing Input arguments in benchmark files. These took string values for which appropriate changes were made in shell file.
- Last thread was implicitly exiting every time. The reason for this was that the number of channels and time steps (lbm) were not synchronized.
- Maximum number of threads that can run parallelly in gem5 was prespecified at a less value. Because of that gem5 was not able to execute all instructions if the total number of instructions were given a high value. Solved by increasing the maximum thread count.
- Panic condition fd<0 occurred which led the failure of opening the .src/benchmark file because DRAM memory capacity usage exceeded/did not match the address range that was assigned.
- CPU type in shell was given as Atomic but name convention of executable created by python libraries in gem5/build/86 is different.

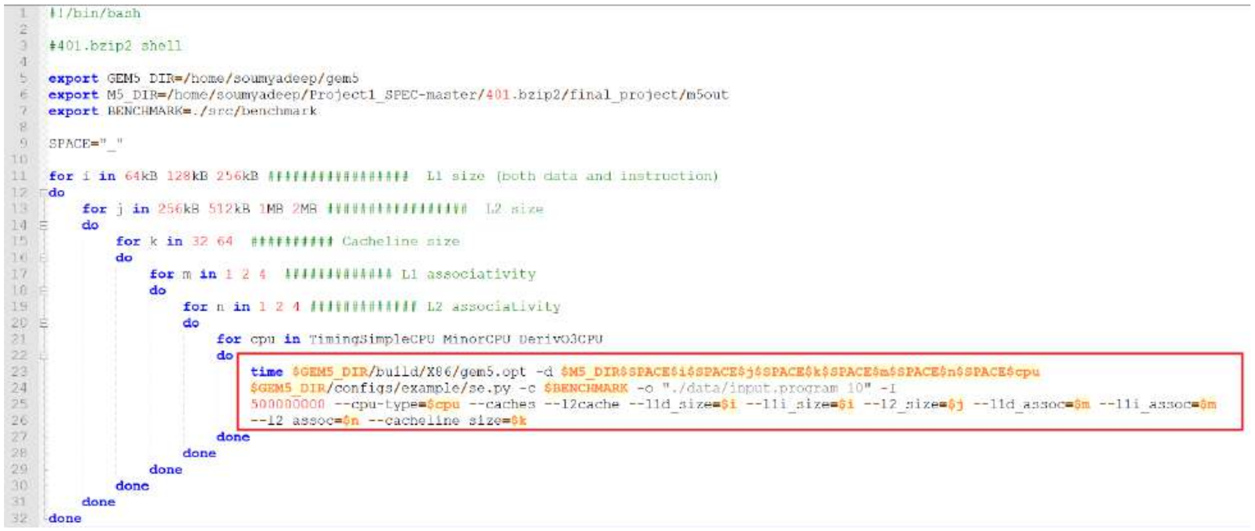
Modifications and Changes

We made changes in shell as per the project guidelines to run all the benchmarks for their respective cache sizes and associativity and CPU type.

To automate the process we have written a shell code along with the changes as mentioned above to perform the said experiments over different combinations of cache sizes, associativity and CPU type.

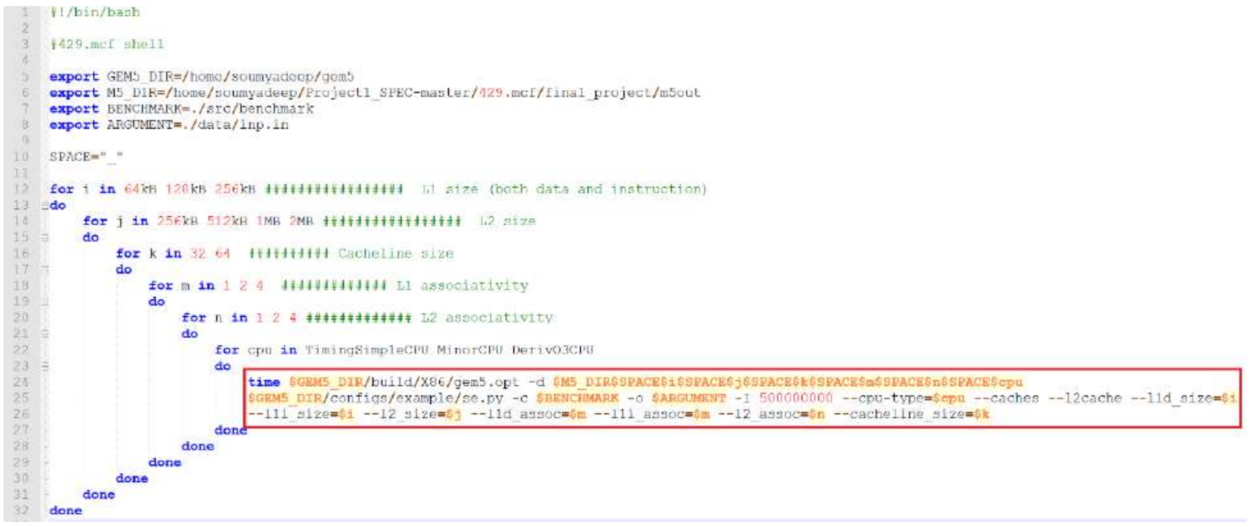
All these changes are given below.

- **401.bzip2**



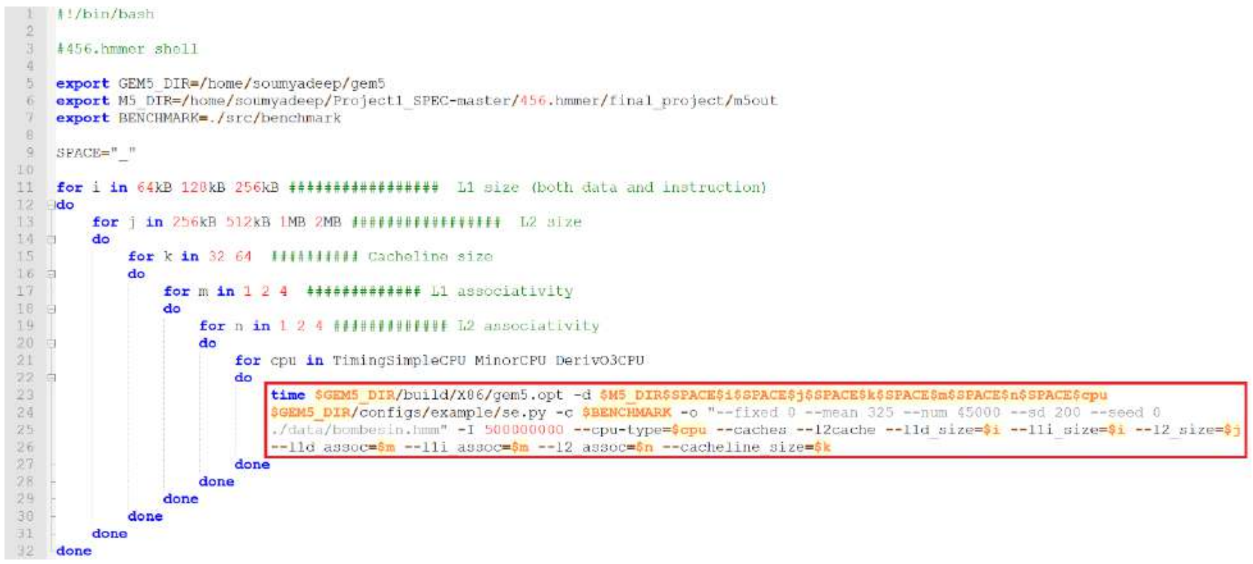
```
#!/bin/bash

#401.bzip2 shell

export GEM5_DIR=/home/soumyadeep/gem5
export M5_DIR=/home/soumyadeep/Project1_SPEC-master/401.bzip2/final_project/m5out
export BENCHMARK=./src/benchmark

SPACE="_"

for i in 64kB 128kB 256kB ################  L1 size (both data and instruction)
do
    for j in 256kB 512kB 1MB 2MB ################  L2 size
    do
        for k in 32 64  ########## Cacheline size
        do
            for m in 1 2 4  ############# L1 associativity
            do
                for n in 1 2 4 ############# L2 associativity
                do
                    for cpu in TimingSimpleCPU MinorCPU DerivO3CPU
                    do
                     time $GEM5_DIR/build/X86/gem5.opt -d $M5_DIR$SPACE$i$SPACE$j$SPACE$k$SPACE$m$SPACE$n$SPACE$cpu
                     $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o "./data/input.program 10" -I
                     500000000 --cpu-type=$cpu --caches --l2cache --l1d_size=$i --l1i_size=$i --l2_size=$j --l1d_assoc=$m --l1i_assoc=$m
                     --l2_assoc=$n --cacheline_size=$k
                    done
                done
            done
        done
    done
done
```

- **429.mcf**



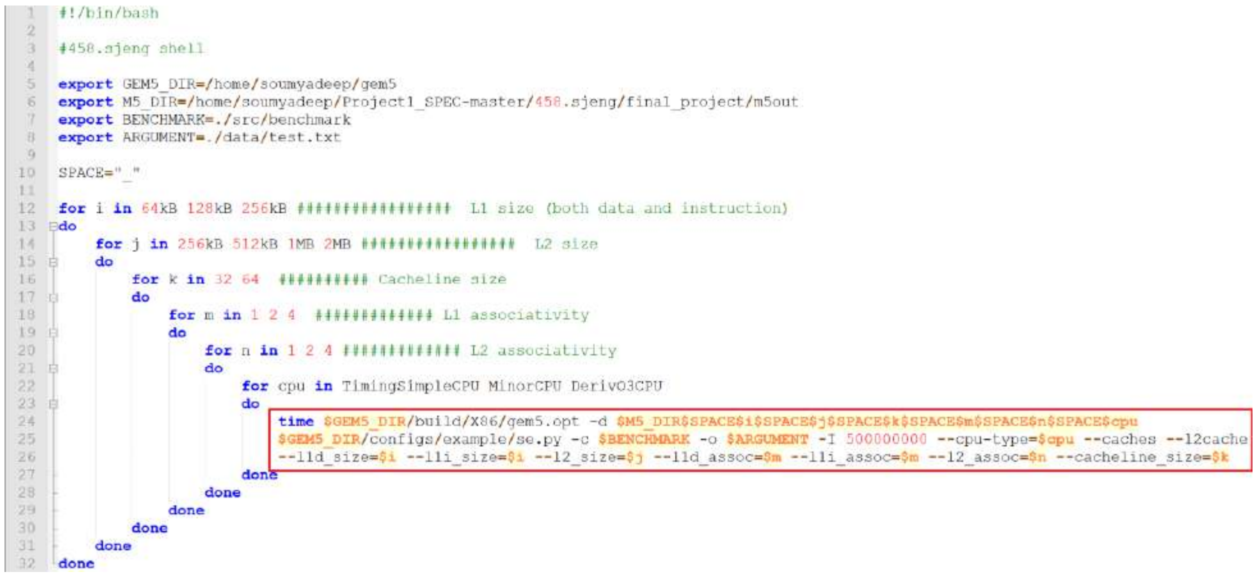
```
#!/bin/bash

#429.mcf shell

export GEM5_DIR=/home/soumyadeep/gem5
export M5_DIR=/home/soumyadeep/Project1_SPEC-master/429.mcf/final_project/m5out
export BENCHMARK=./src/benchmark
export ARGUMENT=./data/inp.in

SPACE="_"

for i in 64kB 128kB 256kB ################  L1 size (both data and instruction)
do
    for j in 256kB 512kB 1MB 2MB ################  L2 size
    do
        for k in 32 64  ########## Cacheline size
        do
            for m in 1 2 4  ############# L1 associativity
            do
                for n in 1 2 4 ############# L2 associativity
                do
                    for cpu in TimingSimpleCPU MinorCPU DerivO3CPU
                    do
                     time $GEM5_DIR/build/X86/gem5.opt -d $M5_DIR$SPACE$i$SPACE$j$SPACE$k$SPACE$m$SPACE$n$SPACE$cpu
                     $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 500000000 --cpu-type=$cpu --caches --l2cache --l1d_size=$i
                     --l1i_size=$i --l2_size=$j --l1d_assoc=$m --l1i_assoc=$m --l2_assoc=$n --cacheline_size=$k
                    done
                done
            done
        done
    done
done
```

- **456.hmmer**



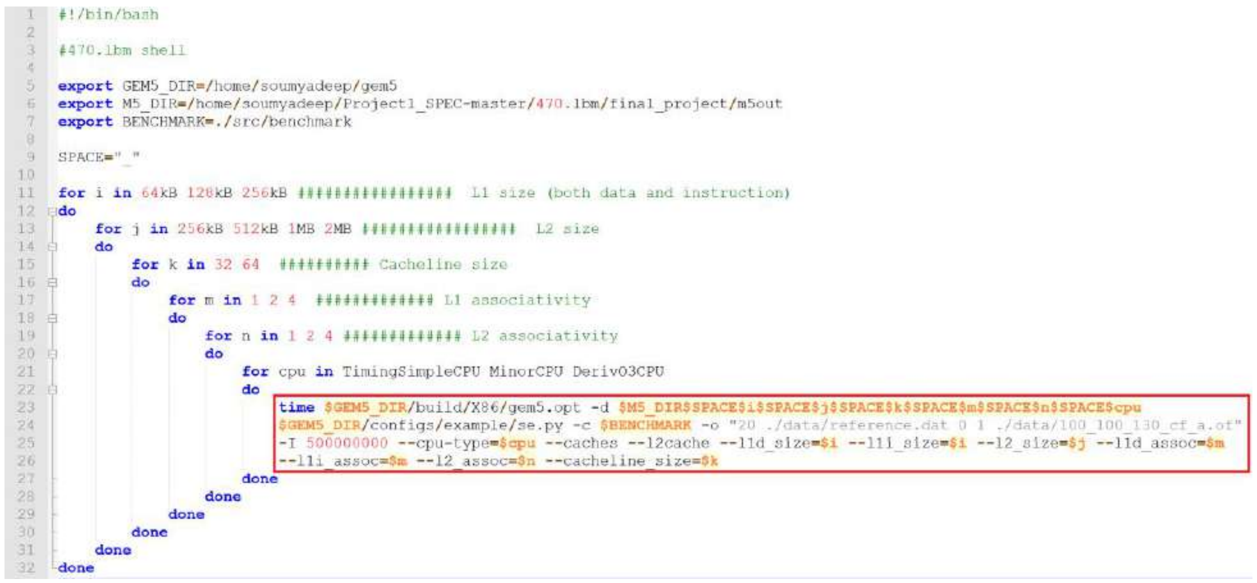
```
#!/bin/bash

#456.hmmer shell

export GEM5_DIR=/home/soumyadeep/gem5
export M5_DIR=/home/soumyadeep/Project1_SPEC-master/456.hmmer/final_project/m5out
export BENCHMARK=./src/benchmark

SPACE="_"

for i in 64kB 128kB 256kB ################  L1 size (both data and instruction)
do
    for j in 256kB 512kB 1MB 2MB ################  L2 size
    do
        for k in 32 64  ########## Cacheline size
        do
            for m in 1 2 4  ############# L1 associativity
            do
                for n in 1 2 4 ############# L2 associativity
                do
                    for cpu in TimingSimpleCPU MinorCPU DerivO3CPU
                    do
                        time $GEM5_DIR/build/X86/gem5.opt -d $M5_DIR$SPACE$i$SPACE$j$SPACE$k$SPACE$m$SPACE$n$SPACE$cpu
                        $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o "--fixed 0 --mean 325 --num 45000 --sd 200 --seed 0
                        ./data/bombesin.hmm" -I 500000000 --cpu-type=$cpu --caches --l2cache --l1d_size=$i --l1i_size=$i --l2_size=$j
                        --l1d_assoc=$m --l1i_assoc=$m --l2_assoc=$n --cacheline_size=$k
                    done
                done
            done
        done
    done
done
```

- **458.sjeng**



```
#!/bin/bash

#458.sjeng shell

export GEM5_DIR=/home/soumyadeep/gem5
export M5_DIR=/home/soumyadeep/Project1_SPEC-master/458.sjeng/final_project/m5out
export BENCHMARK=./src/benchmark
export ARGUMENT=./data/test.txt

SPACE="_"

for i in 64kB 128kB 256kB ################  L1 size (both data and instruction)
do
    for j in 256kB 512kB 1MB 2MB ################  L2 size
    do
        for k in 32 64  ########## Cacheline size
        do
            for m in 1 2 4  ############# L1 associativity
            do
                for n in 1 2 4 ############# L2 associativity
                do
                    for cpu in TimingSimpleCPU MinorCPU DerivO3CPU
                    do
                        time $GEM5_DIR/build/X86/gem5.opt -d $M5_DIR$SPACE$i$SPACE$j$SPACE$k$SPACE$m$SPACE$n$SPACE$cpu
                        $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 500000000 --cpu-type=$cpu --caches --l2cache
                        --l1d_size=$i --l1i_size=$i --l2_size=$j --l1d_assoc=$m --l1i_assoc=$m --l2_assoc=$n --cacheline_size=$k
                    done
                done
            done
        done
    done
done
```

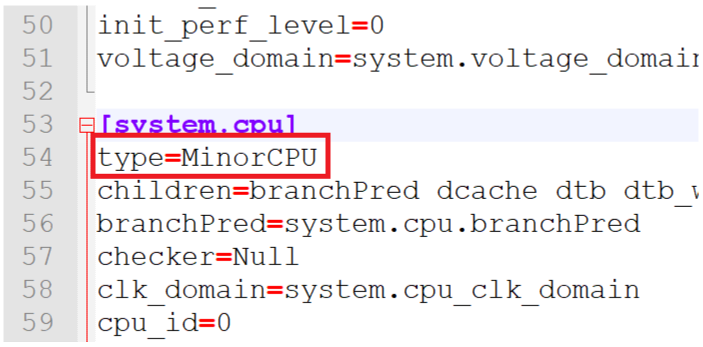
- **470.lbm**



```
#!/bin/bash

#470.lbm shell

export GEM5_DIR=/home/soumyadeep/gem5
export M5_DIR=/home/soumyadeep/Project1_SPEC-master/470.lbm/final_project/m5out
export BENCHMARK=./src/benchmark

SPACE="_"

for i in 64kB 128kB 256kB ################  L1 size (both data and instruction)
do
    for j in 256kB 512kB 1MB 2MB ################  L2 size
    do
        for k in 32 64  ########## Cacheline size
        do
            for m in 1 2 4  ############# L1 associativity
            do
                for n in 1 2 4 ############# L2 associativity
                do
                    for cpu in TimingSimpleCPU MinorCPU DerivO3CPU
                    do
                        time $GEM5_DIR/build/X86/gem5.opt -d $M5_DIR$SPACE$i$SPACE$j$SPACE$k$SPACE$m$SPACE$n$SPACE$cpu
                        $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o "20 ./data/reference.dat 0 1 ./data/100_100_130_cf_a.of"
                        -I 500000000 --cpu-type=$cpu --caches --l2cache --l1d_size=$i --l1i_size=$i --l2_size=$j --l1d_assoc=$m
                        --l1i_assoc=$m --l2_assoc=$n --cacheline_size=$k
                    done
                done
            done
        done
    done
done
```

Config.ini

Whenever a CPU is running for configuration of benchmarks, config.ini file reflects the name of the CPU used. Results of config.ini for MinorCPU is given below. The same changes shows on all config.ini files of all benchmarks.



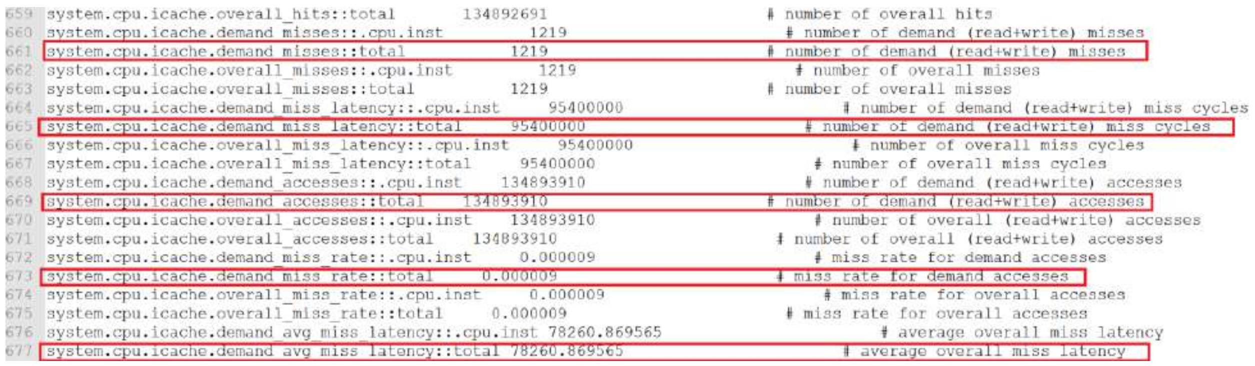
```
init_perf_level=0
voltage_domain=system.voltage_domai

[system.cpu]
type=MinorCPU
children=branchPred dcache dtb dtb_
branchPred=system.cpu.branchPred
checker=Null
clk_domain=system.cpu_clk_domain
cpu_id=0
```

Generated Stats File

Stats are generated for L1(Instruction), L1(Data) and L2 caches example of which is given below.

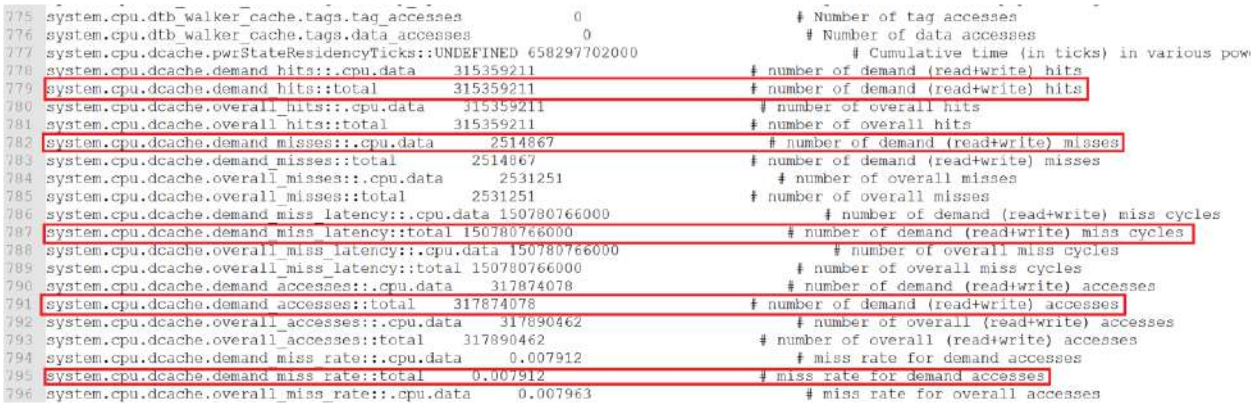
As we are using latest version of gem5, L1 data is produced in stat file as L1 data and instruction cache.

- L1 Instruction Cache Parameters



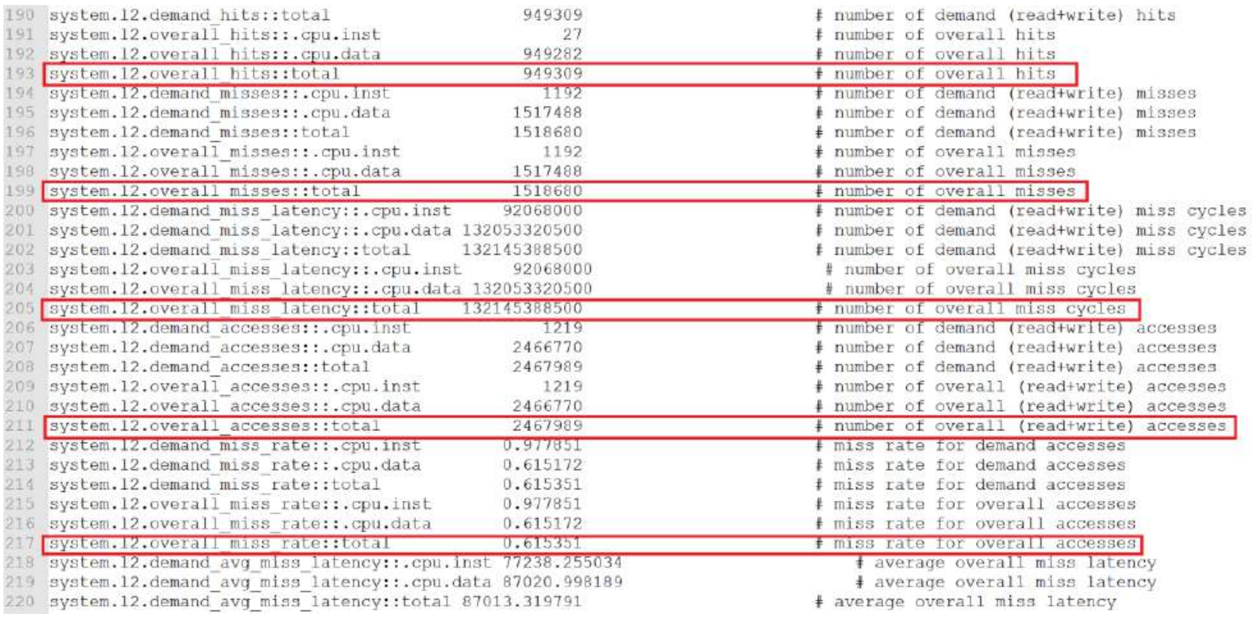
```
system.cpu.icache.overall_hits::total       134892691                  # number of overall hits
system.cpu.icache.demand_misses::.cpu.inst           1219                # number of demand (read+write) misses
system.cpu.icache.demand_misses::total           1219                  # number of demand (read+write) misses
system.cpu.icache.overall_misses::.cpu.inst           1219                # number of overall misses
system.cpu.icache.overall_misses::total           1219                 # number of overall misses
system.cpu.icache.demand_miss_latency::.cpu.inst      95400000                # number of demand (read+write) miss cycles
system.cpu.icache.demand_miss_latency::total      95400000              # number of demand (read+write) miss cycles
system.cpu.icache.overall_miss_latency::.cpu.inst      95400000             # number of overall miss cycles
system.cpu.icache.overall_miss_latency::total      95400000           # number of overall miss cycles
system.cpu.icache.demand_accesses::.cpu.inst     134893910             # number of demand (read+write) accesses
system.cpu.icache.demand_accesses::total     134893910               # number of demand (read+write) accesses
system.cpu.icache.overall_accesses::.cpu.inst     134893910            # number of overall (read+write) accesses
system.cpu.icache.overall_accesses::total     134893910                # number of overall (read+write) accesses
system.cpu.icache.demand_miss_rate::.cpu.inst      0.000009            # miss rate for demand accesses
system.cpu.icache.demand_miss_rate::total      0.000009              # miss rate for demand accesses
system.cpu.icache.overall_miss_rate::.cpu.inst      0.000009           # miss rate for overall accesses
system.cpu.icache.overall_miss_rate::total      0.000009             # miss rate for overall accesses
system.cpu.icache.demand_avg_miss_latency::.cpu.inst 78260.869565             # average overall miss latency
system.cpu.icache.demand_avg_miss_latency::total 78260.869565              # average overall miss latency
```

- L1 Data Cache Parameters



```
system.cpu.dtb_walker_cache.tags.tag_accesses          0                   # Number of tag accesses
system.cpu.dtb_walker_cache.tags.data_accesses          0                   # Number of data accesses
system.cpu.dcache.pwrStateResidencyTicks::UNDEFINED 658297702000                 # Cumulative time (in ticks) in various pow
system.cpu.dcache.demand_hits::.cpu.data     315359211                # number of demand (read+write) hits
system.cpu.dcache.demand_hits::total         315359211                # number of demand (read+write) hits
system.cpu.dcache.overall_hits::.cpu.data     315359211                # number of overall hits
system.cpu.dcache.overall_hits::total         315359211                # number of overall hits
system.cpu.dcache.demand_misses::.cpu.data       2514867                # number of demand (read+write) misses
system.cpu.dcache.demand_misses::total         2514867                # number of demand (read+write) misses
system.cpu.dcache.overall_misses::.cpu.data       2531251                # number of overall misses
system.cpu.dcache.overall_misses::total         2531251                # number of overall misses
system.cpu.dcache.demand_miss_latency::.cpu.data 150780766000                # number of demand (read+write) miss cycles
system.cpu.dcache.demand_miss_latency::total 150780766000                # number of demand (read+write) miss cycles
system.cpu.dcache.overall_miss_latency::.cpu.data 150780766000                # number of overall miss cycles
system.cpu.dcache.overall_miss_latency::total 150780766000                # number of overall miss cycles
system.cpu.dcache.demand_accesses::.cpu.data     317874078                # number of demand (read+write) accesses
system.cpu.dcache.demand_accesses::total     317874078                # number of demand (read+write) accesses
system.cpu.dcache.overall_accesses::.cpu.data     317890462                # number of overall (read+write) accesses
system.cpu.dcache.overall_accesses::total     317890462                # number of overall (read+write) accesses
system.cpu.dcache.demand_miss_rate::.cpu.data     0.007912                # miss rate for demand accesses
system.cpu.dcache.demand_miss_rate::total     0.007912                # miss rate for demand accesses
system.cpu.dcache.overall_miss_rate::.cpu.data     0.007963                # miss rate for overall accesses
```

- L2 Cache Parameters



```
system.l2.demand_hits::total                   949309                       # number of demand (read+write) hits
system.l2.overall_hits::.cpu.inst                  27                       # number of overall hits
system.l2.overall_hits::.cpu.data              949282                       # number of overall hits
system.l2.overall_hits::total                  949309                       # number of overall hits
system.l2.demand_misses::.cpu.inst               1192                       # number of demand (read+write) misses
system.l2.demand_misses::.cpu.data            1517488                       # number of demand (read+write) misses
system.l2.demand_misses::total                1518680                       # number of demand (read+write) misses
system.l2.overall_misses::.cpu.inst              1192                       # number of overall misses
system.l2.overall_misses::.cpu.data           1517488                       # number of overall misses
system.l2.overall_misses::total               1518680                       # number of overall misses
system.l2.demand_miss_latency::.cpu.inst     92068000                       # number of demand (read+write) miss cycles
system.l2.demand_miss_latency::.cpu.data 132053320500                       # number of demand (read+write) miss cycles
system.l2.demand_miss_latency::total     132145388500                       # number of demand (read+write) miss cycles
system.l2.overall_miss_latency::.cpu.inst     92068000                       # number of overall miss cycles
system.l2.overall_miss_latency::.cpu.data 132053320500                       # number of overall miss cycles
system.l2.overall_miss_latency::total     132145388500                       # number of overall miss cycles
system.l2.demand_accesses::.cpu.inst             1219                       # number of demand (read+write) accesses
system.l2.demand_accesses::.cpu.data          2466770                       # number of demand (read+write) accesses
system.l2.demand_accesses::total              2467989                       # number of demand (read+write) accesses
system.l2.overall_accesses::.cpu.inst            1219                       # number of overall (read+write) accesses
system.l2.overall_accesses::.cpu.data         2466770                       # number of overall (read+write) accesses
system.l2.overall_accesses::total             2467989                       # number of overall (read+write) accesses
system.l2.demand_miss_rate::.cpu.inst        0.977851                       # miss rate for demand accesses
system.l2.demand_miss_rate::.cpu.data        0.615172                       # miss rate for demand accesses
system.l2.demand_miss_rate::total            0.615351                       # miss rate for demand accesses
system.l2.overall_miss_rate::.cpu.inst       0.977851                       # miss rate for overall accesses
system.l2.overall_miss_rate::.cpu.data       0.615172                       # miss rate for overall accesses
system.l2.overall_miss_rate::total           0.615351                       # miss rate for overall accesses
system.l2.demand_avg_miss_latency::.cpu.inst 77238.255034                       # average overall miss latency
system.l2.demand_avg_miss_latency::.cpu.data 87020.998189                       # average overall miss latency
system.l2.demand_avg_miss_latency::total 87013.319791                       # average overall miss latency
```

Cost Function

For our experiments, we defined a cost function depending on the size, associativity and performance of a particular cache configuration of a particular CPU type.

Assumptions:

- Base price of 1kB of L1 cache is equal to 16 times to that of L2 cache.
    - From the performed experiments we observed that with the increase in L1 cache size CPI is decreasing.
    - Area overhead of L1 is much lesser than L2 therefore it is much more expensive.
    - To build over this assumption we took references from [1] & [2] research articles.
- Base price of associativity of L1 cache is 4 times to that of L2 cache.
    - To increase associativity, we need extra hardware. Since L1 is much smaller in size, the hardware precision required for L1 will lead to more price of L1 than L2.
    - To build over this assumption we took references from [3] & [4] research articles.
- **To further ease our calculations for the experiments**, we have assumed the following prices:
    - Base price of L2 (1kB) = \$1
    - Hence, Base price of L1 (1kB) = \$16
    - Base price of associativity of L2 = \$0.5
    - Hence, Base price of associativity of L1 = \$2
    - Base price of each cache-line block = \$0.25
    - Fixed Cost (Fabrication and other development) = \$20

These values are chosen hypothetically based on the assumptions for the ease of calculations. Real market values might differ.

Based on our assumptions mentioned above, we define our cost function as given below.

$$\boxed{\mathbf{CF = FC + N_{MC}\{\ [(BP_{L1} * L1_{size}) + (BP_{L1\text{-}Assoc} * L1_{Assoc})] + [(BP_{L2} * L2_{size}) + (BP_{L2\text{-}Assoc} * L2_{Assoc})] + [BP_{CL} * CL_{size}]\ \}}}$$

Where,

- CF - > Cost Function
- FC -> Fixed Cost
- N_{MC} -> Number of Cores (default = 1)
- BP_{L1} -> Base Price of L1 Cache
- $L1_{size}$-> L1 Cache Size
- $BP_{L1\text{-}Assoc}$ -> Base price of associativity of L1
- $L1_{Assoc}$ -> L1 Cache Associativity
- BP_{L2} -> Base price of L2 Cache
- $L2_{size}$ -> L2 Cache Size
- $BP_{L2\text{-}Assoc}$ -> Base price of associativity of L1
- $L2_{Assoc}$-> L1 Cache Associativity
- BP_{CL} -> Base price of Cache-Line
- CL_{size} -> Cache-Line size

Evaluating Function

In this project, we define an Evaluate Function to determine the optimal cache configuration depending on the CPI and the above defined Cost Function.

We performed the evaluate functions on all the different CPU's and the benchmarks to show an optimal cache configuration choice for each case.

To define the Evaluation Function, we define the following methodology to generate the respective equation

- Calculate the sum of all the Cost Values for all the cache configurations
- Find the average Cost Value for all the Cost Values w.r.t. to total number of experiments
- Calculate the absolute mean difference for all the Cost Values for all cache configurations
- Use Local Minima to find the set of minimum Cost Values from the whole set
- Among the minimum Cost Values, Find the optimal cache configuration respective to the minimum CPI among them

  From the above methodology, we represent the Evaluate Equation as follows –

$$\boxed{\textbf{EF} = \min_{CPI}\{\lim_{CF\to Local\ Minima}[\mathbf{Abs}(\mathbf{Avg}(\sum \mathbf{CFi}) - \mathbf{CFi})]\}}$$

Where,

- EF -> Evaluation Function
- CF -> Cost Function (Defined earlier)
- $\text{Lim}_{\text{CF->Local Minima}}$ -> Minimum cost values from all the cache configurations
- Min_{CPI} -> Minimum CPI in the local minima
- Summation is done over total number of combinations

Extra Notes for Evaluation Function :

- We chose to define our Evaluation function in such a way that it finds the configurations with least price then the configuration having least CPI among the least expensive ones.
- The reason for the above is that we assumed that a customer might be more focused towards most effective ones with least price.
- Other evaluation could be for a customer who is more focused towards the most effective configuration overall or within a particular price range.

$$\boxed{\text{EF} = \min_{CPI}\{\int_{P1}^{P2}[\text{Abs}(\text{Avg}(\sum \text{CFi}) - \text{CFi})]\}}$$

  Where, P1 & P2 are the price range.

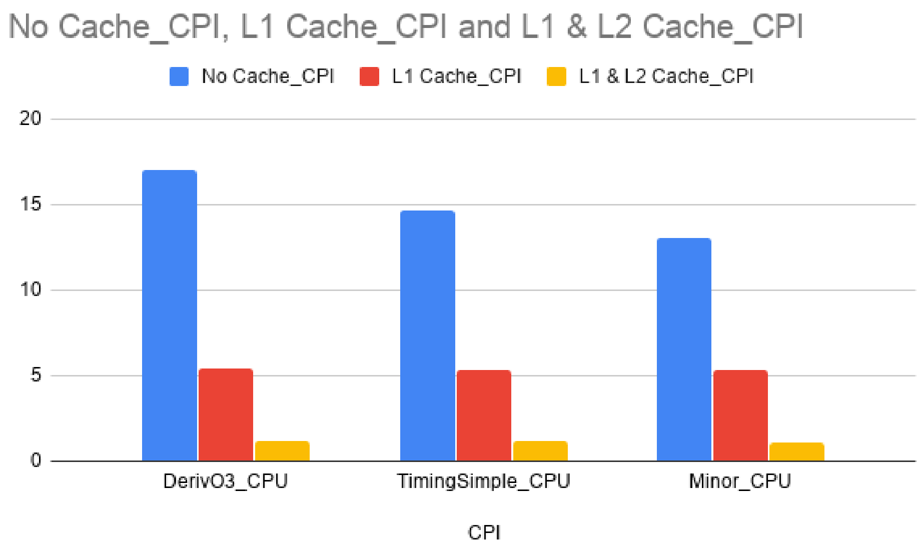
Results

In this section we will describe all the experiments performed over various cache configurations and different CPU types for all the SPEC benchmarks using CPI, Cost Function and Evaluating Function. Moreover, we will explain the tradeoffs for various cache configurations.

Note : *Total combinations that were taken for each SPEC benchmark and CPU can be found in the appendix. Due to large number of combinations our overall graph contains experiment number as varying cache sizes, as mentioned in appendix, as x-axis and CPI on the y-axis. Another graph shows the best choice for each CPU and Benchmark which has x-axis as experiment number and on y-axis the Cost. Also the data for the best configuration will be mentioned below. For all the other values kindly refer to data set calculated and provided alongside this report in a folder called 'dataset'. Name of each data set is provided along with the configuration below. As there are over 3000 total datasets, they are not put entirely in the report.*

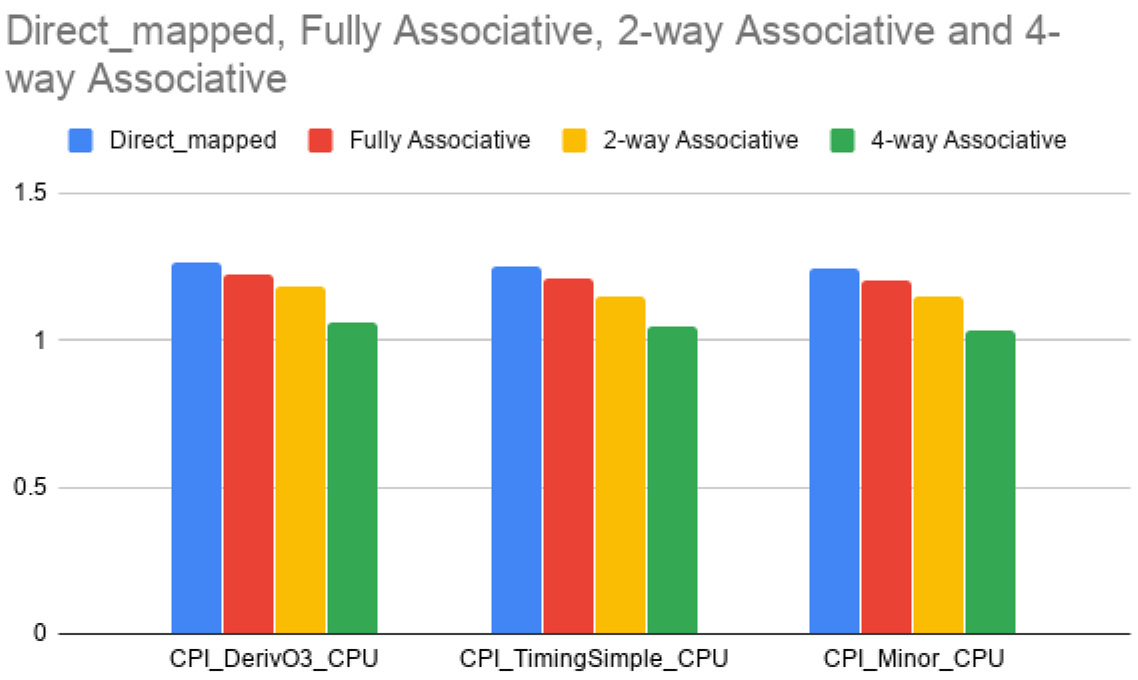
1. In this section, we represent the experiments and tradeoff performed and analyzed over no cache, only L1 cache and both L1 & L2 cache. We observed that a CPI of a system increases exponentially without the use of cache memory due to the increase in more DRAM and Physical Memory Access and takes a lot of time to execute a set of instructions.

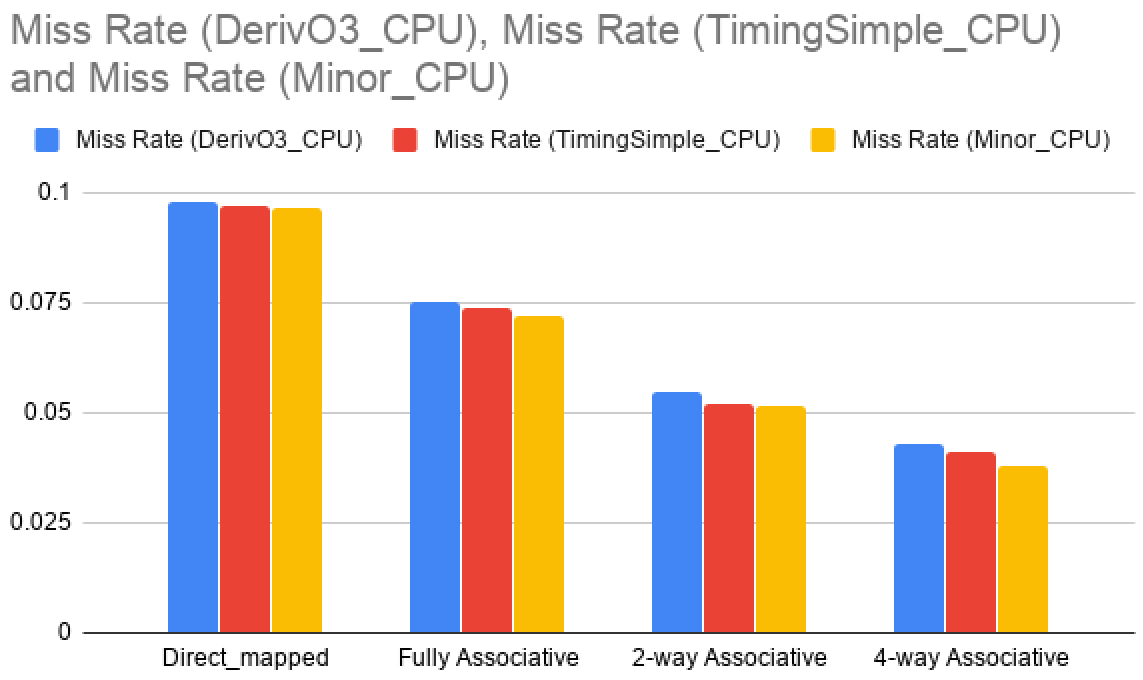
$\text{CPI}_{\text{CPU-TYPE}}$	No Cache	L1 Cache	L1 & L2 Caches
$\text{CPI}_{\text{DerivO3CPU}}$	16.9968	5.4422	1.2089
$\text{CPI}_{\text{TimingSimpleCPU}}$	14.697	5.3086	1.1375
$\text{CPI}_{\text{MinorCPU}}$	13.0724	5.2749	1.1264



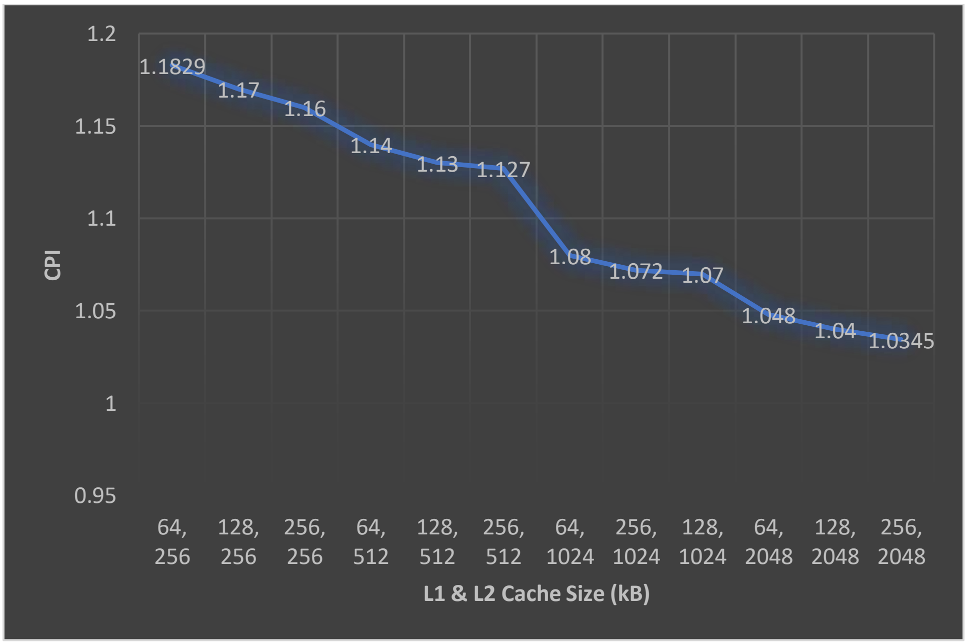
2. In this section, we represent the experiments and tradeoff performed and analyzed over various associativity and direct-mapped cache. We observed that a direct mapped cache is too slow with LRU replacement policy (default config) as it takes more cycles to fetch data from the physical memory and execute a set of instructions. It increases the miss rate, latency and CPI of the system compared to the caches with better associativity. With increase in associativity, the miss rate decreases, and the CPI improves for the respective system.

Experiment Type (Average of L1&L2)	**Direct Mapped**	**Fully Associative**	**2-way Associative**	**4-way Associative**
$CPI_{DerivO3CPU}$	1.2641	1.2202	1.1829	1.0621
$CPI_{TimingSimpleCPU}$	1.2508	1.2113	1.1508	1.0478
$CPI_{MinorCPU}$	1.2455	1.2015	1.1471	1.03345
$Missrate_{DerivO3CPU}$	0.09778	0.0752	0.0546	0.04279
$Missrate_{TimingSimpleCPU}$	0.09715	0.0736	0.0522	0.04112
$Missrate_{MinorCPU}$	0.09645	0.072	0.0516	0.038

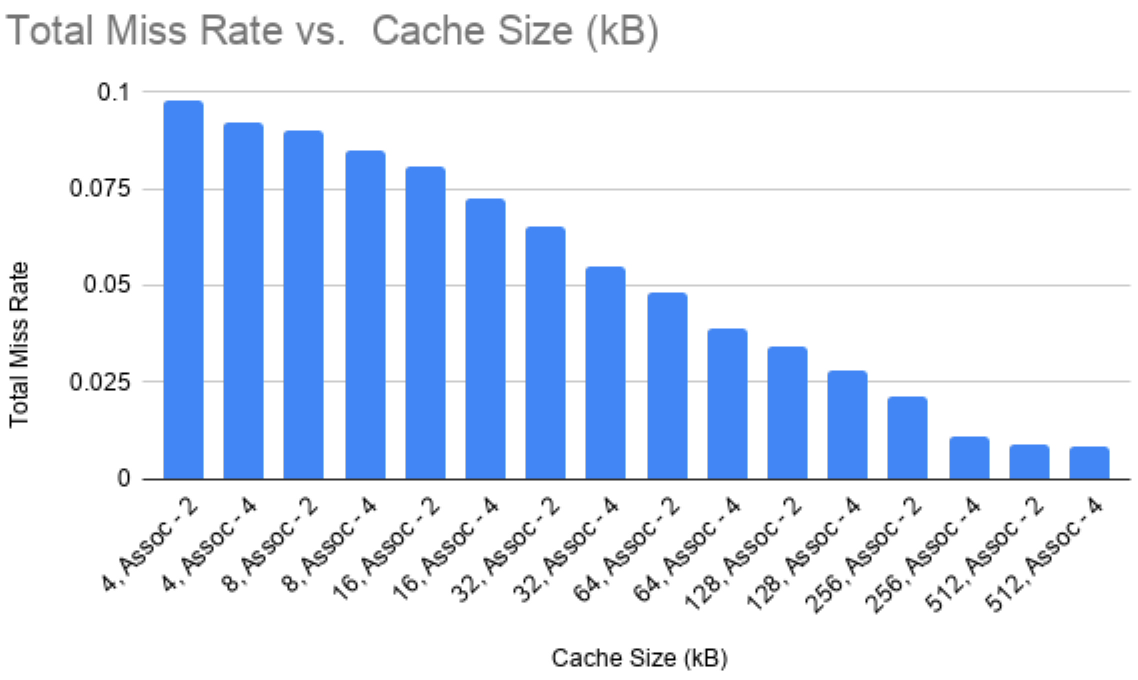




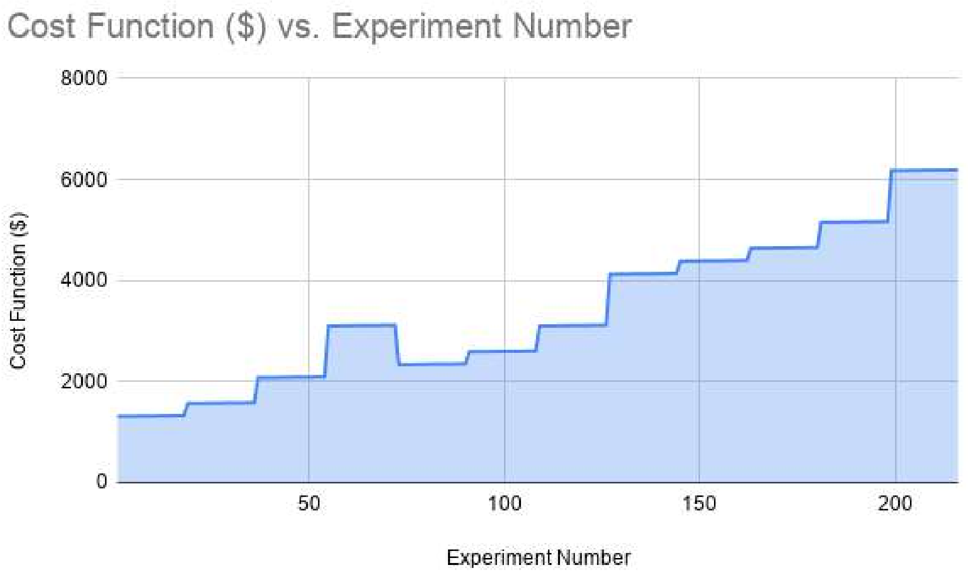
3. In this section, the graph shows the variation when L1 and L2 are varied and cache line size and associativity are fixed.



4. In this section, the graph describes the decrease in total miss rate with the increase in cache size and degree of associativity.

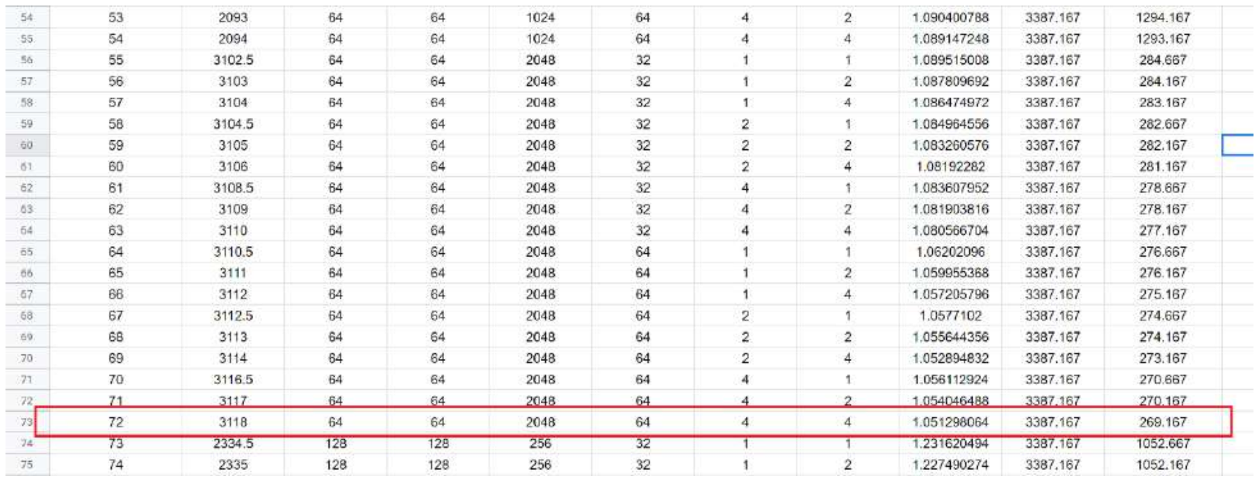


5. In this section, the graph shows the change in cost function with the changes in cache configurations. With increase in cache sizes, cost values are increasing.

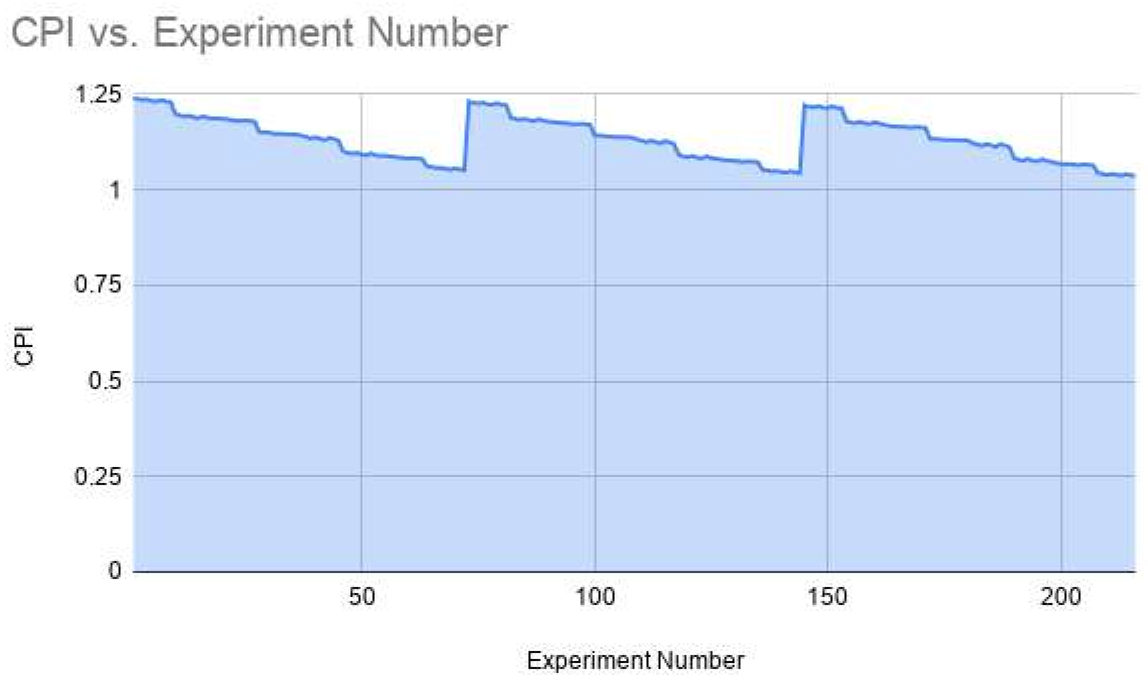


6. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 401.zip2 and cpu-type – Minor. Highlighted part shows the best choice. For full data-set look at “401zip2_MinorCPU” in dataset folder.

Experiment Number	Cost Function	IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Avg Cost	Mean diff

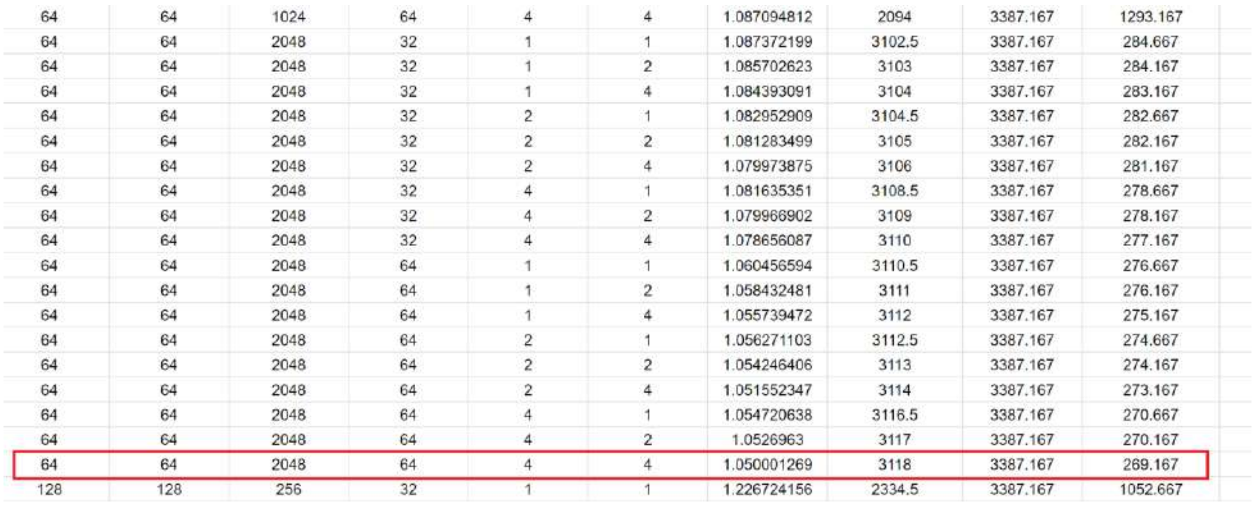


54	53	2093	64	64	1024	64	4	2	1.090400788	3387.167	1294.167
55	54	2094	64	64	1024	64	4	4	1.089147248	3387.167	1293.167
56	55	3102.5	64	64	2048	32	1	1	1.089515008	3387.167	284.667
57	56	3103	64	64	2048	32	1	2	1.087809692	3387.167	284.167
58	57	3104	64	64	2048	32	1	4	1.086474972	3387.167	283.167
59	58	3104.5	64	64	2048	32	2	1	1.084964556	3387.167	282.667
60	59	3105	64	64	2048	32	2	2	1.083260576	3387.167	282.167
61	60	3106	64	64	2048	32	2	4	1.08192282	3387.167	281.167
62	61	3108.5	64	64	2048	32	4	1	1.083607952	3387.167	278.667
63	62	3109	64	64	2048	32	4	2	1.081903816	3387.167	278.167
64	63	3110	64	64	2048	32	4	4	1.080566704	3387.167	277.167
65	64	3110.5	64	64	2048	64	1	1	1.06202096	3387.167	276.667
66	65	3111	64	64	2048	64	1	2	1.059955368	3387.167	276.167
67	66	3112	64	64	2048	64	1	4	1.057205796	3387.167	275.167
68	67	3112.5	64	64	2048	64	2	1	1.0577102	3387.167	274.667
69	68	3113	64	64	2048	64	2	2	1.055644356	3387.167	274.167
70	69	3114	64	64	2048	64	2	4	1.052894832	3387.167	273.167
71	70	3116.5	64	64	2048	64	4	1	1.056112924	3387.167	270.667
72	71	3117	64	64	2048	64	4	2	1.054046488	3387.167	270.167
73	72	3118	64	64	2048	64	4	4	1.051298064	3387.167	269.167
74	73	2334.5	128	128	256	32	1	1	1.231620494	3387.167	1052.667
75	74	2335	128	128	256	32	1	2	1.227490274	3387.167	1052.167

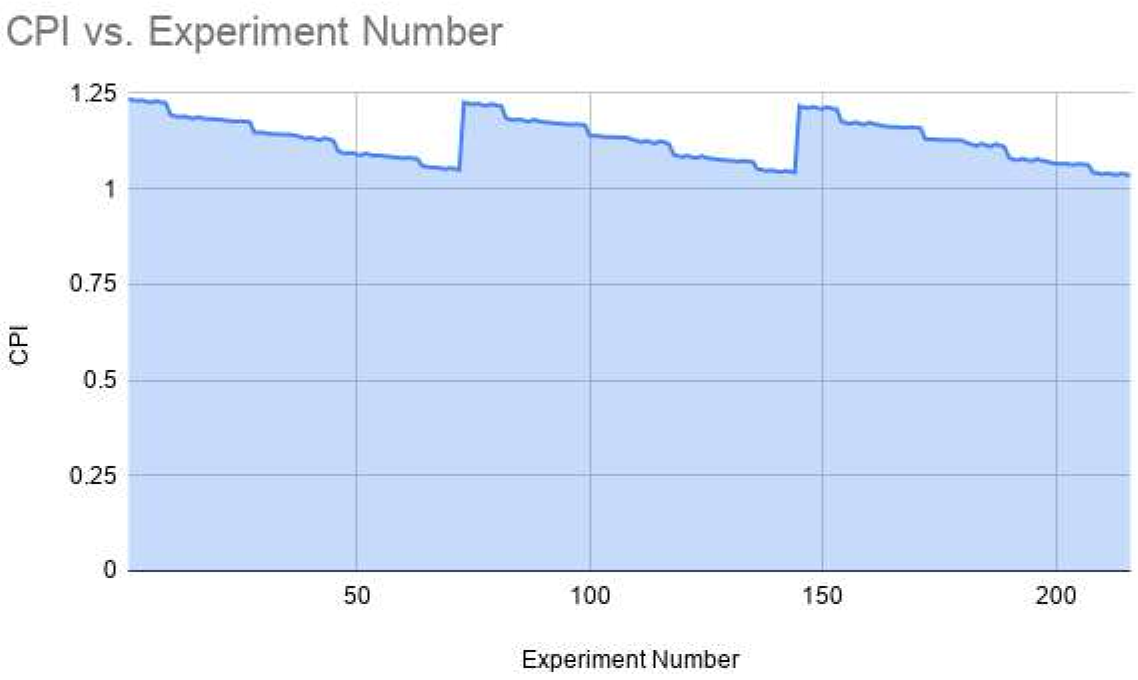


7. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 401.zip2 and cpu-type – TimingSimple. Highlighted part shows the best choice. For full data-set look at “401zip2_TimingSimpleCPU” in dataset folder.

IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff

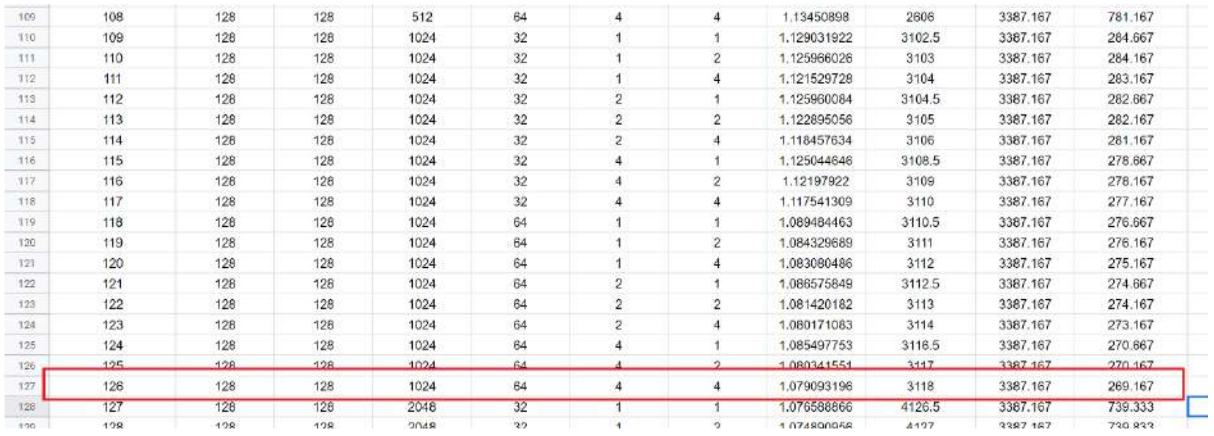


64	64	1024	64	4	4	1.087094812	2094	3387.167	1293.167
64	64	2048	32	1	1	1.087372199	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.085702623	3103	3387.167	284.167
64	64	2048	32	1	4	1.084393091	3104	3387.167	283.167
64	64	2048	32	2	1	1.082952909	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.081283499	3105	3387.167	282.167
64	64	2048	32	2	4	1.079973875	3106	3387.167	281.167
64	64	2048	32	4	1	1.081635351	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.079966902	3109	3387.167	278.167
64	64	2048	32	4	4	1.078656087	3110	3387.167	277.167
64	64	2048	64	1	1	1.060456594	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.058432481	3111	3387.167	276.167
64	64	2048	64	1	4	1.055739472	3112	3387.167	275.167
64	64	2048	64	2	1	1.056271103	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.054246406	3113	3387.167	274.167
64	64	2048	64	2	4	1.051552347	3114	3387.167	273.167
64	64	2048	64	4	1	1.054720638	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.0526963	3117	3387.167	270.167
64	64	2048	64	4	4	1.050001269	3118	3387.167	269.167
128	128	256	32	1	1	1.226724156	2334.5	3387.167	1052.667

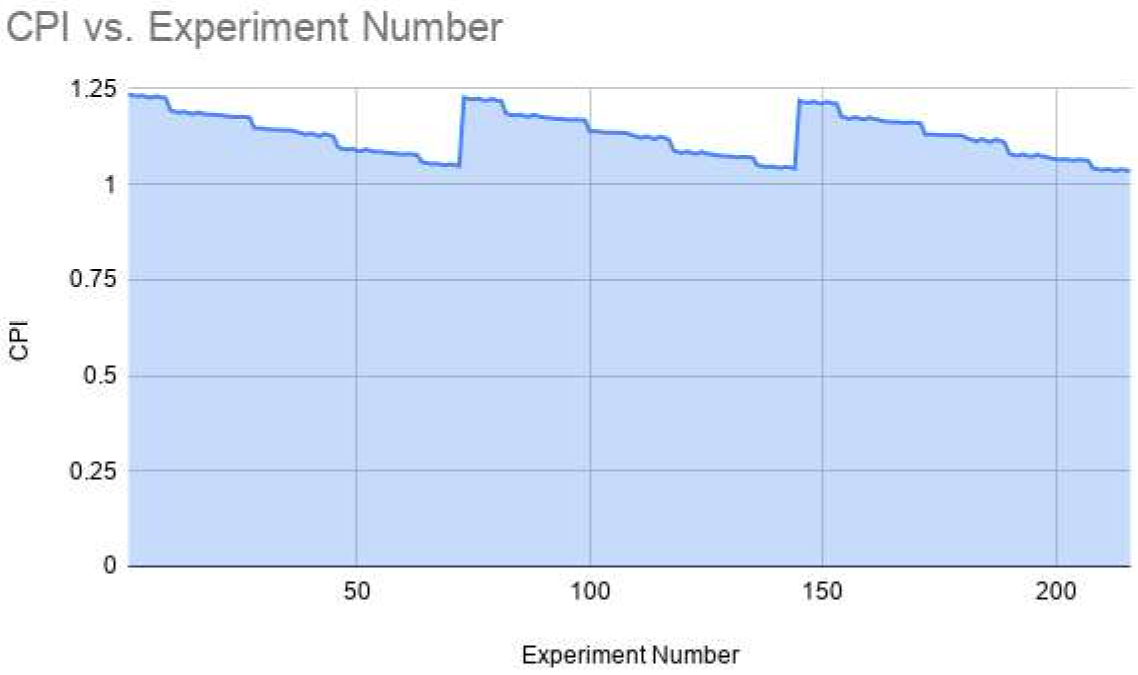


8. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 401.zip2 and cpu-type – DerivO3. Highlighted part shows the best choice. For full data-set look at “401zip2_DerivO3CPU” in dataset folder.

Experiment Number	IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff

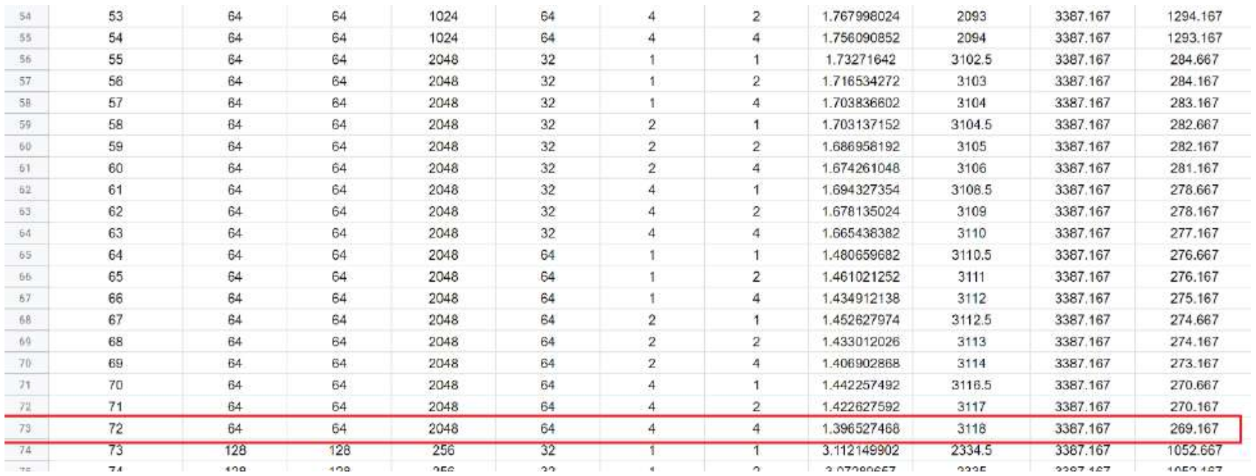


109	108	128	128	512	64	4	4	1.13450898	2606	3387.167	781.167
110	109	128	128	1024	32	1	1	1.129031922	3102.5	3387.167	284.667
111	110	128	128	1024	32	1	2	1.125966026	3103	3387.167	284.167
112	111	128	128	1024	32	1	4	1.121529728	3104	3387.167	283.167
113	112	128	128	1024	32	2	1	1.125960084	3104.5	3387.167	282.667
114	113	128	128	1024	32	2	2	1.122895056	3105	3387.167	282.167
115	114	128	128	1024	32	2	4	1.118457634	3106	3387.167	281.167
116	115	128	128	1024	32	4	1	1.125044646	3108.5	3387.167	278.667
117	116	128	128	1024	32	4	2	1.12197922	3109	3387.167	278.167
118	117	128	128	1024	32	4	4	1.117541309	3110	3387.167	277.167
119	118	128	128	1024	64	1	1	1.089484463	3110.5	3387.167	276.667
120	119	128	128	1024	64	1	2	1.084329689	3111	3387.167	276.167
121	120	128	128	1024	64	1	4	1.083080486	3112	3387.167	275.167
122	121	128	128	1024	64	2	1	1.086575849	3112.5	3387.167	274.667
123	122	128	128	1024	64	2	2	1.081420182	3113	3387.167	274.167
124	123	128	128	1024	64	2	4	1.080171083	3114	3387.167	273.167
125	124	128	128	1024	64	4	1	1.085497753	3116.5	3387.167	270.667
126	125	128	128	1024	64	4	2	1.080341551	3117	3387.167	270.167
127	126	128	128	1024	64	4	4	1.079093196	3118	3387.167	269.167
128	127	128	128	2048	32	1	1	1.076588866	4126.5	3387.167	739.333
129	128	128	128	2048	32	1	2	1.074890956	4127	3387.167	739.833

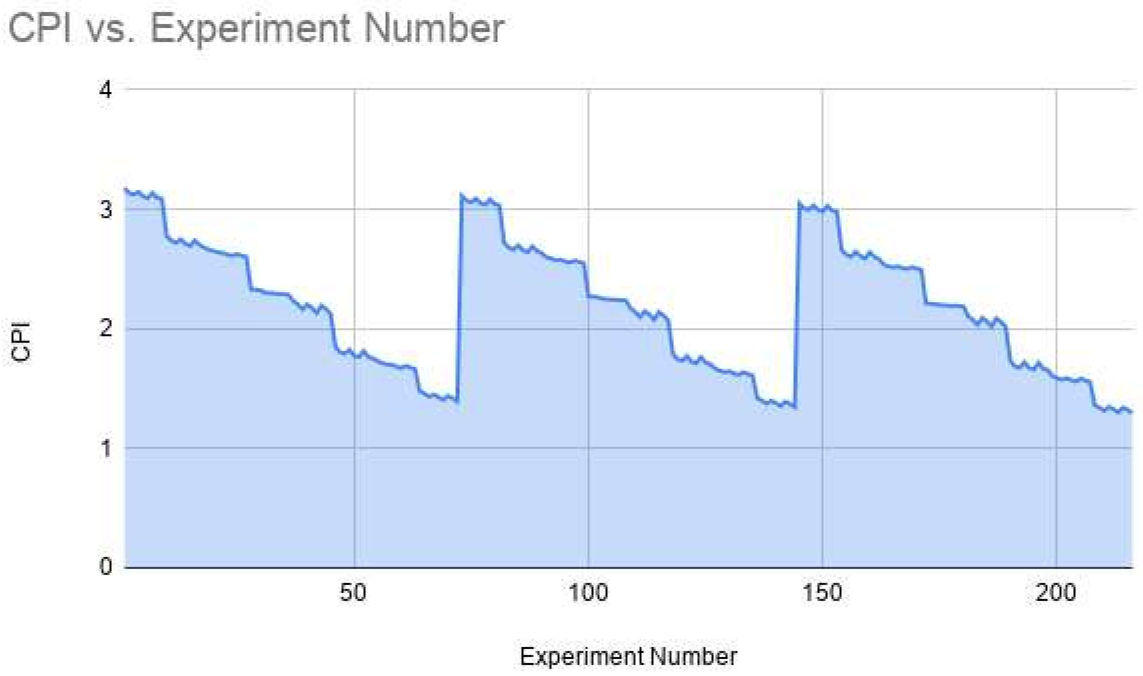


9. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 429.mcf and cpu-type – Minor. Highlighted part shows the best choice. For full data-set look at “429mcf_MinorCPU” in dataset folder.

Experiment Number	IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff

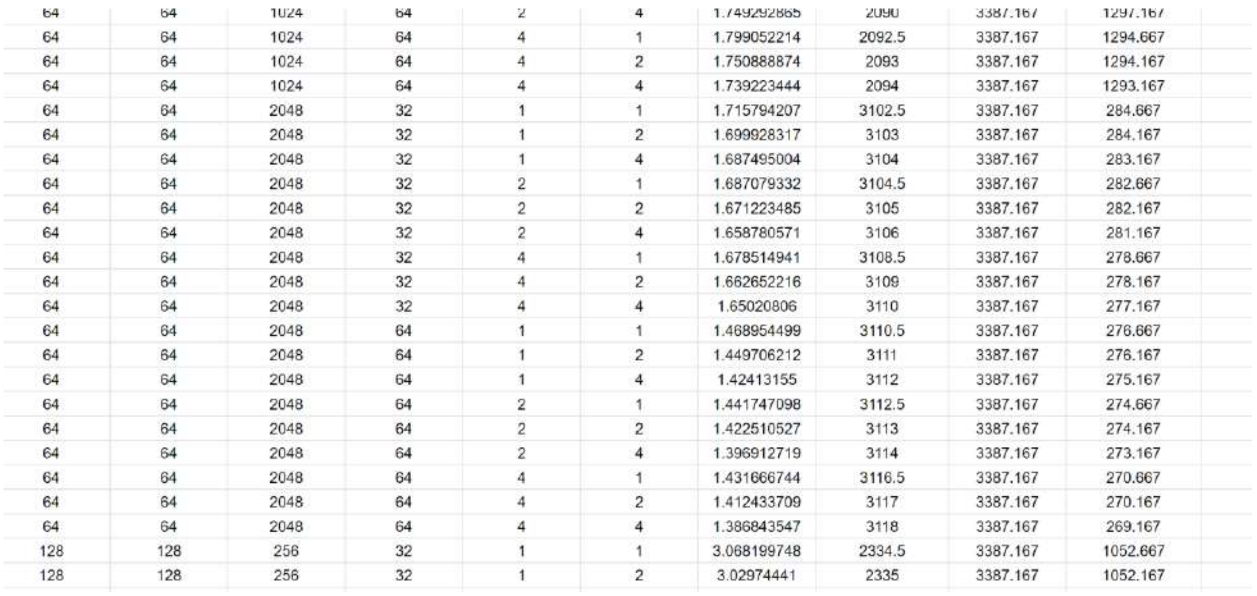


53	64	64	1024	64	4	2	1.767998024	2093	3387.167	1294.167
54	64	64	1024	64	4	4	1.756090852	2094	3387.167	1293.167
55	64	64	2048	32	1	1	1.73271642	3102.5	3387.167	284.667
56	64	64	2048	32	1	2	1.716534272	3103	3387.167	284.167
57	64	64	2048	32	1	4	1.703836602	3104	3387.167	283.167
58	64	64	2048	32	2	1	1.703137152	3104.5	3387.167	282.667
59	64	64	2048	32	2	2	1.686958192	3105	3387.167	282.167
60	64	64	2048	32	2	4	1.674261048	3106	3387.167	281.167
61	64	64	2048	32	4	1	1.694327354	3108.5	3387.167	278.667
62	64	64	2048	32	4	2	1.678135024	3109	3387.167	278.167
63	64	64	2048	32	4	4	1.665438382	3110	3387.167	277.167
64	64	64	2048	64	1	1	1.480659682	3110.5	3387.167	276.667
65	64	64	2048	64	1	2	1.461021252	3111	3387.167	276.167
66	64	64	2048	64	1	4	1.434912138	3112	3387.167	275.167
67	64	64	2048	64	2	1	1.452627974	3112.5	3387.167	274.667
68	64	64	2048	64	2	2	1.433012026	3113	3387.167	274.167
69	64	64	2048	64	2	4	1.406902868	3114	3387.167	273.167
70	64	64	2048	64	4	1	1.442257492	3116.5	3387.167	270.667
71	64	64	2048	64	4	2	1.422627592	3117	3387.167	270.167
72	64	64	2048	64	4	4	1.396527468	3118	3387.167	269.167
73	128	128	256	32	1	1	3.112149902	2334.5	3387.167	1052.667
74	128	128	256	32	1	2	[illegible]	2335	3387.167	1052.167

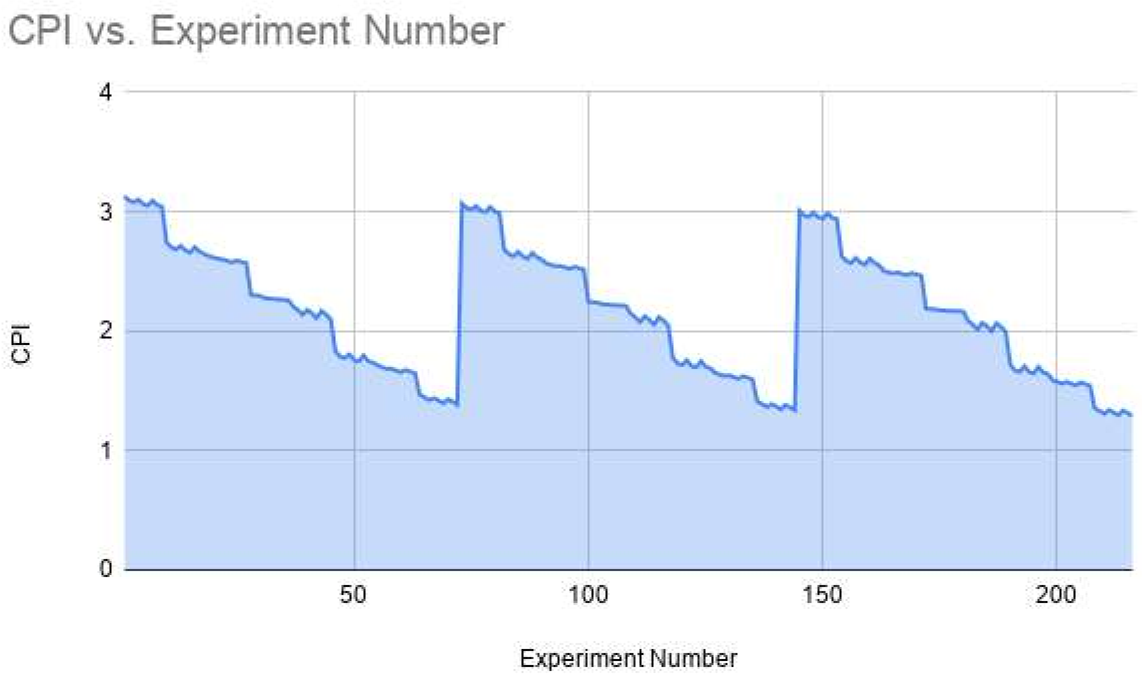


10. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 429.mcf and cpu-type – TimingSimple. Highlighted part shows the best choice. For full data-set look at “429mcf_TimingSimpleCPU” in dataset folder.

IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff

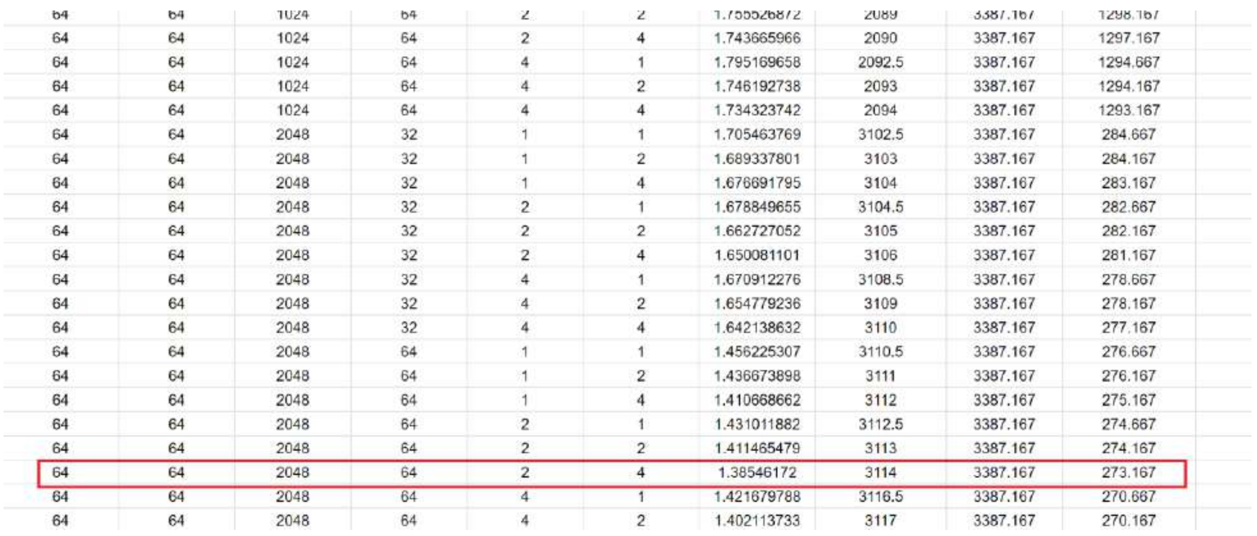


64	64	1024	64	2	4	1.749292865	2090	3387.167	1297.167
64	64	1024	64	4	1	1.799052214	2092.5	3387.167	1294.667
64	64	1024	64	4	2	1.750888874	2093	3387.167	1294.167
64	64	1024	64	4	4	1.739223444	2094	3387.167	1293.167
64	64	2048	32	1	1	1.715794207	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.699928317	3103	3387.167	284.167
64	64	2048	32	1	4	1.687495004	3104	3387.167	283.167
64	64	2048	32	2	1	1.687079332	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.671223485	3105	3387.167	282.167
64	64	2048	32	2	4	1.658780571	3106	3387.167	281.167
64	64	2048	32	4	1	1.678514941	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.662652216	3109	3387.167	278.167
64	64	2048	32	4	4	1.65020806	3110	3387.167	277.167
64	64	2048	64	1	1	1.468954499	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.449706212	3111	3387.167	276.167
64	64	2048	64	1	4	1.42413155	3112	3387.167	275.167
64	64	2048	64	2	1	1.441747098	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.422510527	3113	3387.167	274.167
64	64	2048	64	2	4	1.396912719	3114	3387.167	273.167
64	64	2048	64	4	1	1.431666744	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.412433709	3117	3387.167	270.167
64	64	2048	64	4	4	1.386843547	3118	3387.167	269.167
128	128	256	32	1	1	3.068199748	2334.5	3387.167	1052.667
128	128	256	32	1	2	3.02974441	2335	3387.167	1052.167

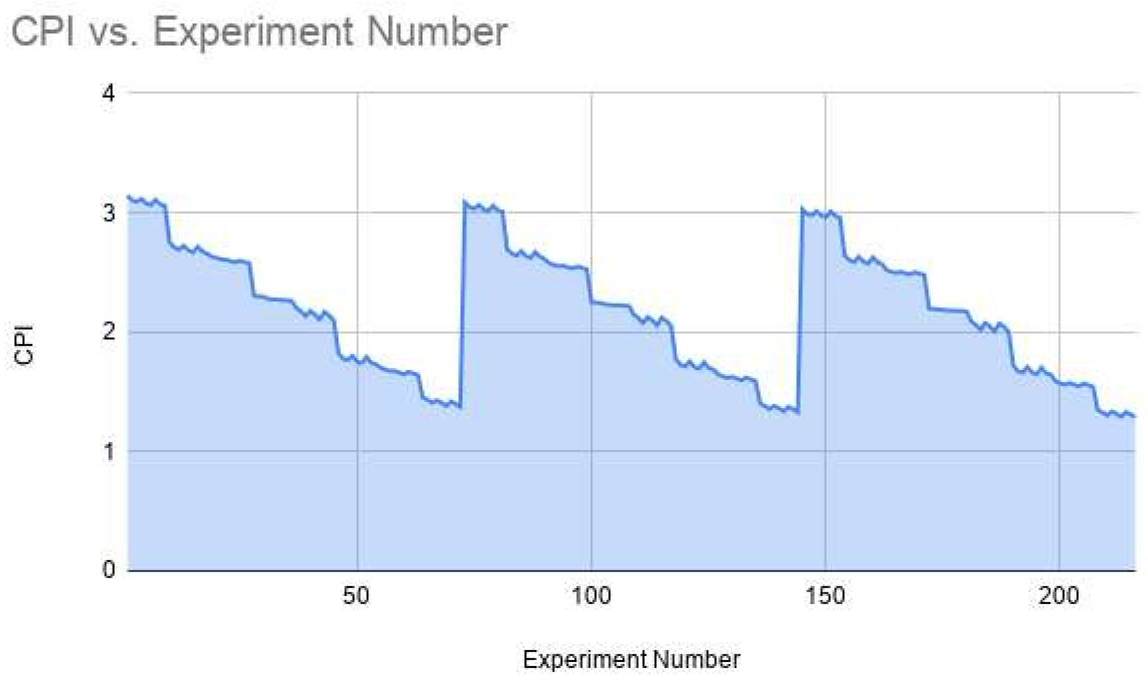


11. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 429.mcf and cpu-type – DerivO3. Highlighted part shows the best choice. For full data-set look at "429mcf_DerivO3CPU" in dataset folder.

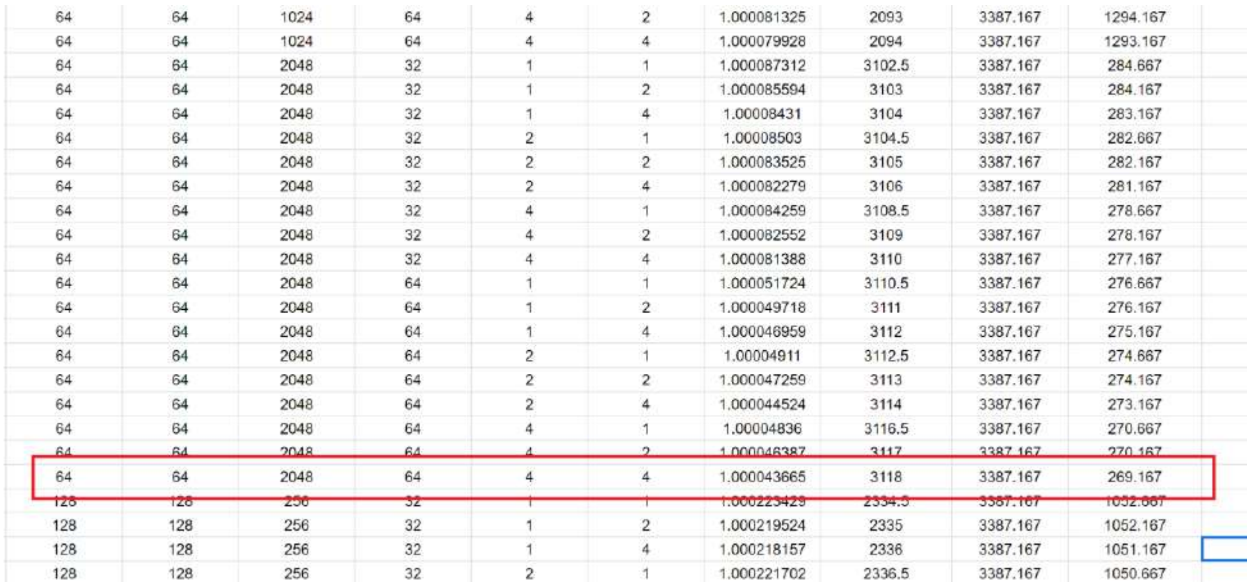
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff



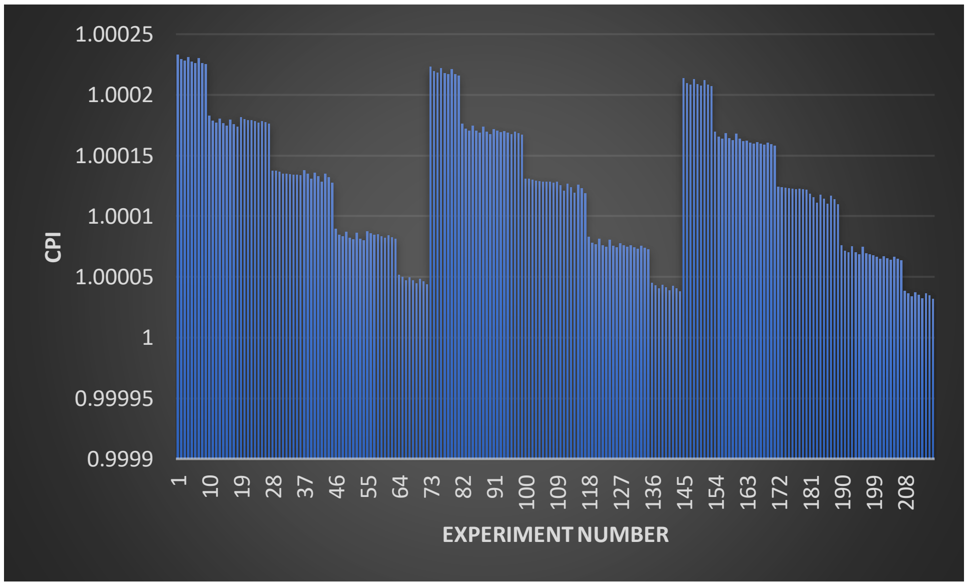
64	64	1024	64	2	2	1.755526872	2089	3387.167	1298.167
64	64	1024	64	2	4	1.743665966	2090	3387.167	1297.167
64	64	1024	64	4	1	1.795169658	2092.5	3387.167	1294.667
64	64	1024	64	4	2	1.746192738	2093	3387.167	1294.167
64	64	1024	64	4	4	1.734323742	2094	3387.167	1293.167
64	64	2048	32	1	1	1.705463769	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.689337801	3103	3387.167	284.167
64	64	2048	32	1	4	1.676691795	3104	3387.167	283.167
64	64	2048	32	2	1	1.678849655	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.662727052	3105	3387.167	282.167
64	64	2048	32	2	4	1.650081101	3106	3387.167	281.167
64	64	2048	32	4	1	1.670912276	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.654779236	3109	3387.167	278.167
64	64	2048	32	4	4	1.642138632	3110	3387.167	277.167
64	64	2048	64	1	1	1.456225307	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.436673898	3111	3387.167	276.167
64	64	2048	64	1	4	1.410668662	3112	3387.167	275.167
64	64	2048	64	2	1	1.431011882	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.411465479	3113	3387.167	274.167
64	64	2048	64	2	4	1.38546172	3114	3387.167	273.167
64	64	2048	64	4	1	1.421679788	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.402113733	3117	3387.167	270.167



12. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 456.hmmer and cpu-type – Minor. Highlighted part shows the best choice. For full data-set look at “456hmmer_MinorCPU” in dataset folder.

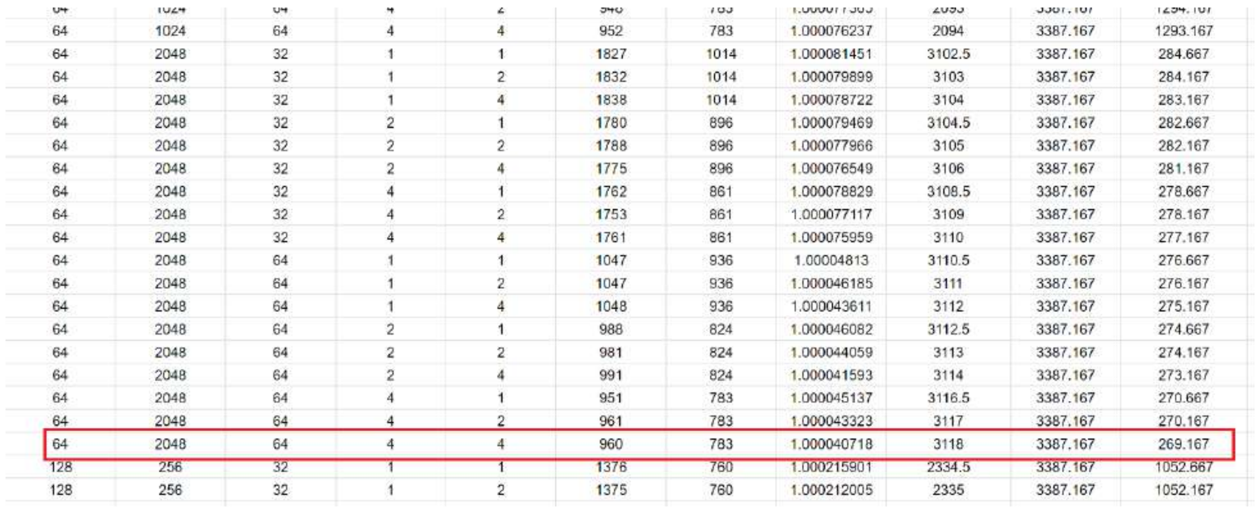


IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
64	64	1024	64	4	2	1.000081325	2093	3387.167	1294.167
64	64	1024	64	4	4	1.000079928	2094	3387.167	1293.167
64	64	2048	32	1	1	1.000087312	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.000085594	3103	3387.167	284.167
64	64	2048	32	1	4	1.00008431	3104	3387.167	283.167
64	64	2048	32	2	1	1.00008503	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.000083525	3105	3387.167	282.167
64	64	2048	32	2	4	1.000082279	3106	3387.167	281.167
64	64	2048	32	4	1	1.000084259	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.000082552	3109	3387.167	278.167
64	64	2048	32	4	4	1.000081388	3110	3387.167	277.167
64	64	2048	64	1	1	1.000051724	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.000049718	3111	3387.167	276.167
64	64	2048	64	1	4	1.000046959	3112	3387.167	275.167
64	64	2048	64	2	1	1.00004911	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.000047259	3113	3387.167	274.167
64	64	2048	64	2	4	1.000044524	3114	3387.167	273.167
64	64	2048	64	4	1	1.00004836	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.000046387	3117	3387.167	270.167
64	64	2048	64	4	4	1.000043665	3118	3387.167	269.167
128	128	256	32	1	1	1.000223429	2334.5	3387.167	1052.667
128	128	256	32	1	2	1.000219524	2335	3387.167	1052.167
128	128	256	32	1	4	1.000218157	2336	3387.167	1051.167
128	128	256	32	2	1	1.000221702	2336.5	3387.167	1050.667

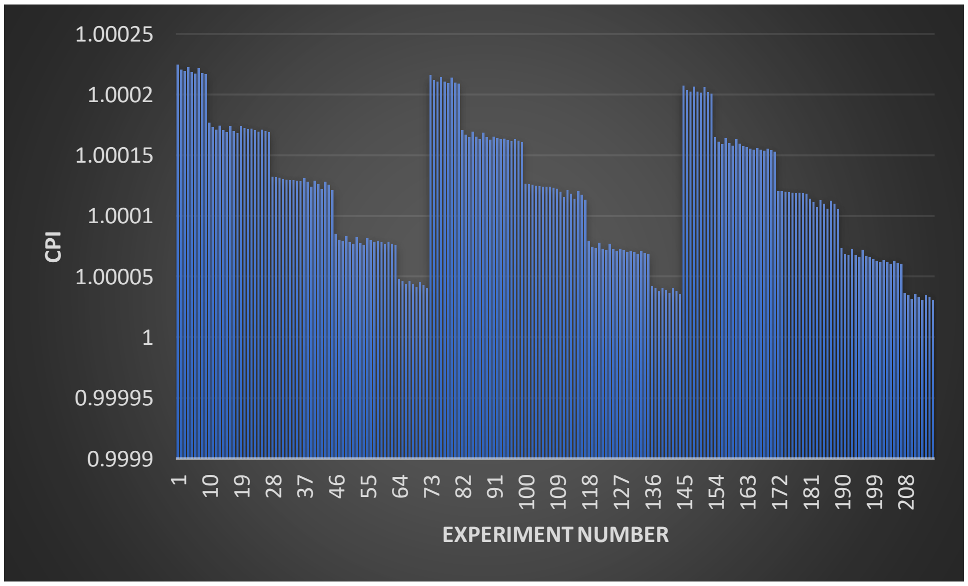


13. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 456.hmmer and cpu-type – TimingSimple. Highlighted part shows the best choice. For full data-set look at "456hmmer_TimingSimpleCPU" in dataset folder.

IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff

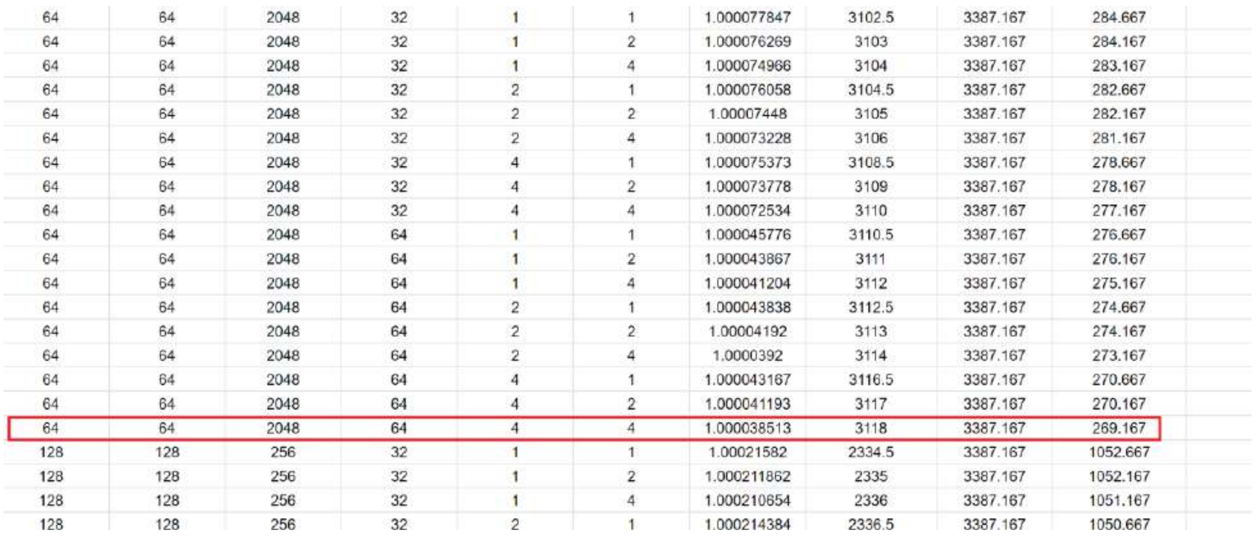


64	1024	64	4	2	948	783	[illegible]	2093	3387.167	1294.167
64	1024	64	4	4	952	783	1.000076237	2094	3387.167	1293.167
64	2048	32	1	1	1827	1014	1.000081451	3102.5	3387.167	284.667
64	2048	32	1	2	1832	1014	1.000079899	3103	3387.167	284.167
64	2048	32	1	4	1838	1014	1.000078722	3104	3387.167	283.167
64	2048	32	2	1	1780	896	1.000079469	3104.5	3387.167	282.667
64	2048	32	2	2	1788	896	1.000077966	3105	3387.167	282.167
64	2048	32	2	4	1775	896	1.000076549	3106	3387.167	281.167
64	2048	32	4	1	1762	861	1.000078829	3108.5	3387.167	278.667
64	2048	32	4	2	1753	861	1.000077117	3109	3387.167	278.167
64	2048	32	4	4	1761	861	1.000075959	3110	3387.167	277.167
64	2048	64	1	1	1047	936	1.00004813	3110.5	3387.167	276.667
64	2048	64	1	2	1047	936	1.000046185	3111	3387.167	276.167
64	2048	64	1	4	1048	936	1.000043611	3112	3387.167	275.167
64	2048	64	2	1	988	824	1.000046082	3112.5	3387.167	274.667
64	2048	64	2	2	981	824	1.000044059	3113	3387.167	274.167
64	2048	64	2	4	991	824	1.000041593	3114	3387.167	273.167
64	2048	64	4	1	951	783	1.000045137	3116.5	3387.167	270.667
64	2048	64	4	2	961	783	1.000043323	3117	3387.167	270.167
64	2048	64	4	4	960	783	1.000040718	3118	3387.167	269.167
128	256	32	1	1	1376	760	1.000215901	2334.5	3387.167	1052.667
128	256	32	1	2	1375	760	1.000212005	2335	3387.167	1052.167

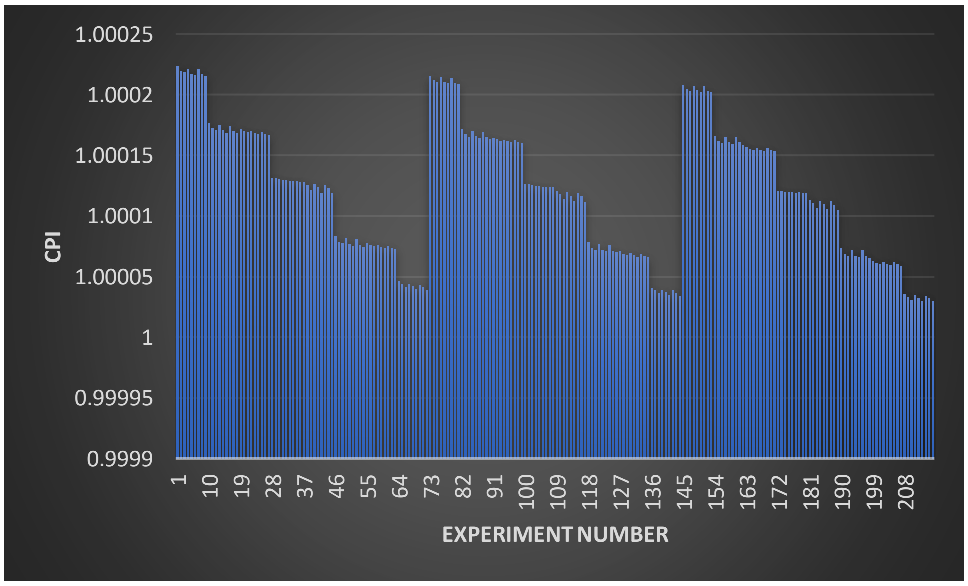


14. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 456.hmmer and cpu-type – DerivO3. Highlighted part shows the best choice. For full data-set look at “456hmmer_DerivO3CPU” in dataset folder.

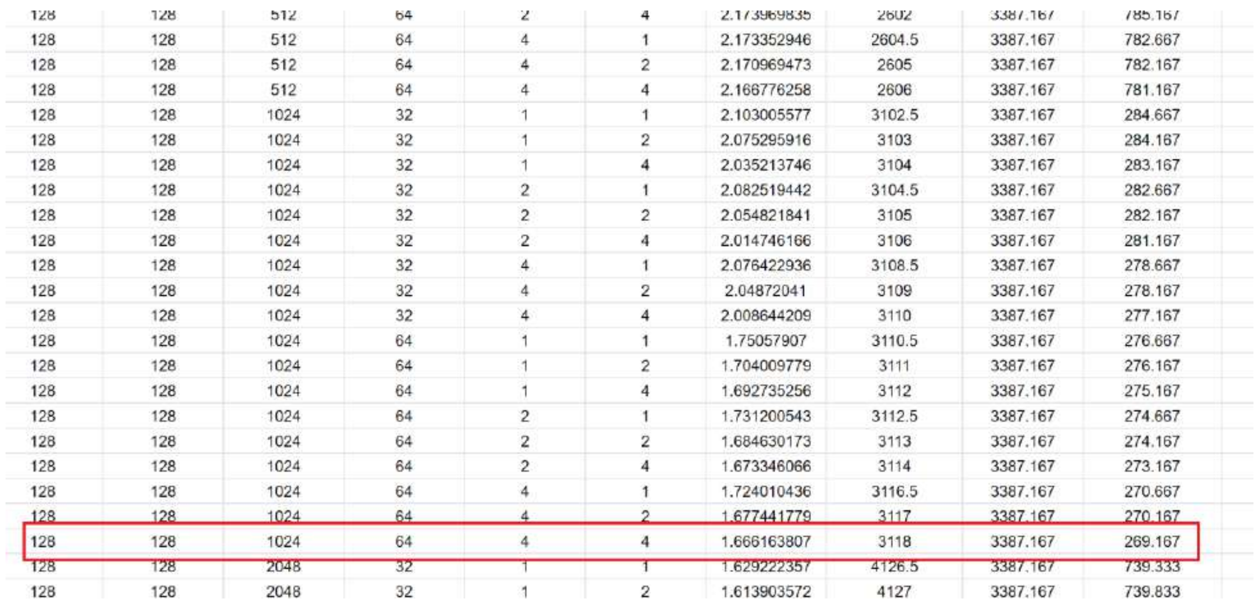
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff



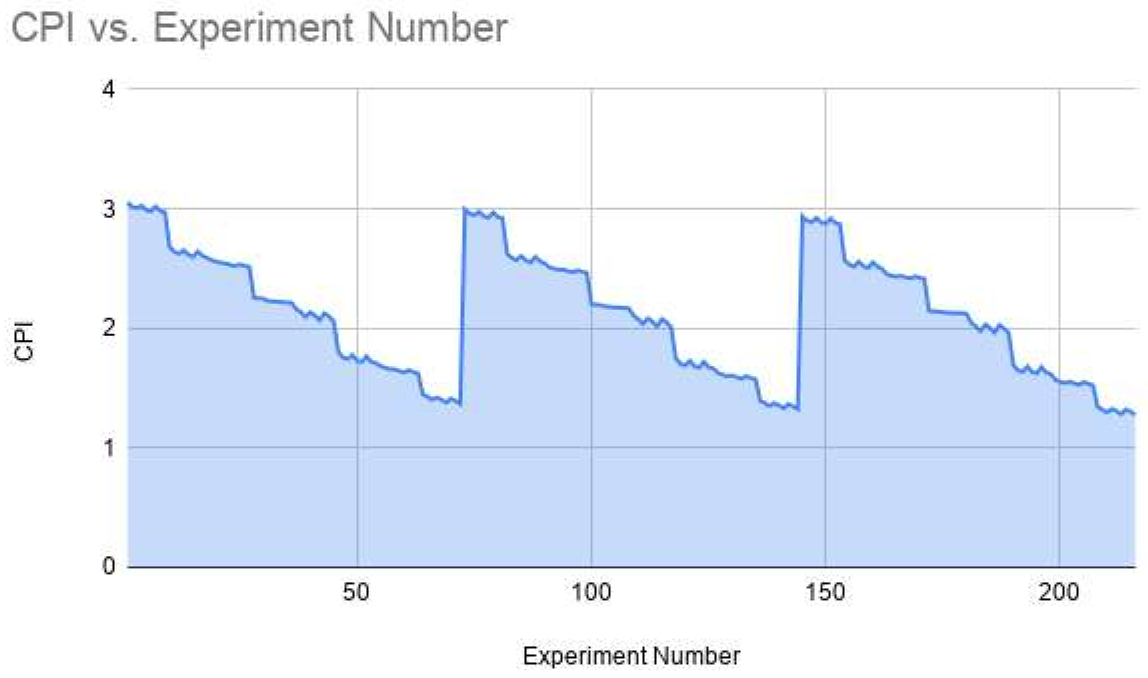
64	64	2048	32	1	1	1.000077847	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.000076269	3103	3387.167	284.167
64	64	2048	32	1	4	1.000074966	3104	3387.167	283.167
64	64	2048	32	2	1	1.000076058	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.00007448	3105	3387.167	282.167
64	64	2048	32	2	4	1.000073228	3106	3387.167	281.167
64	64	2048	32	4	1	1.000075373	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.000073778	3109	3387.167	278.167
64	64	2048	32	4	4	1.000072534	3110	3387.167	277.167
64	64	2048	64	1	1	1.000045776	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.000043867	3111	3387.167	276.167
64	64	2048	64	1	4	1.000041204	3112	3387.167	275.167
64	64	2048	64	2	1	1.000043838	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.00004192	3113	3387.167	274.167
64	64	2048	64	2	4	1.0000392	3114	3387.167	273.167
64	64	2048	64	4	1	1.000043167	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.000041193	3117	3387.167	270.167
64	64	2048	64	4	4	1.000038513	3118	3387.167	269.167
128	128	256	32	1	1	1.00021582	2334.5	3387.167	1052.667
128	128	256	32	1	2	1.000211862	2335	3387.167	1052.167
128	128	256	32	1	4	1.000210654	2336	3387.167	1051.167
128	128	256	32	2	1	1.000214384	2336.5	3387.167	1050.667



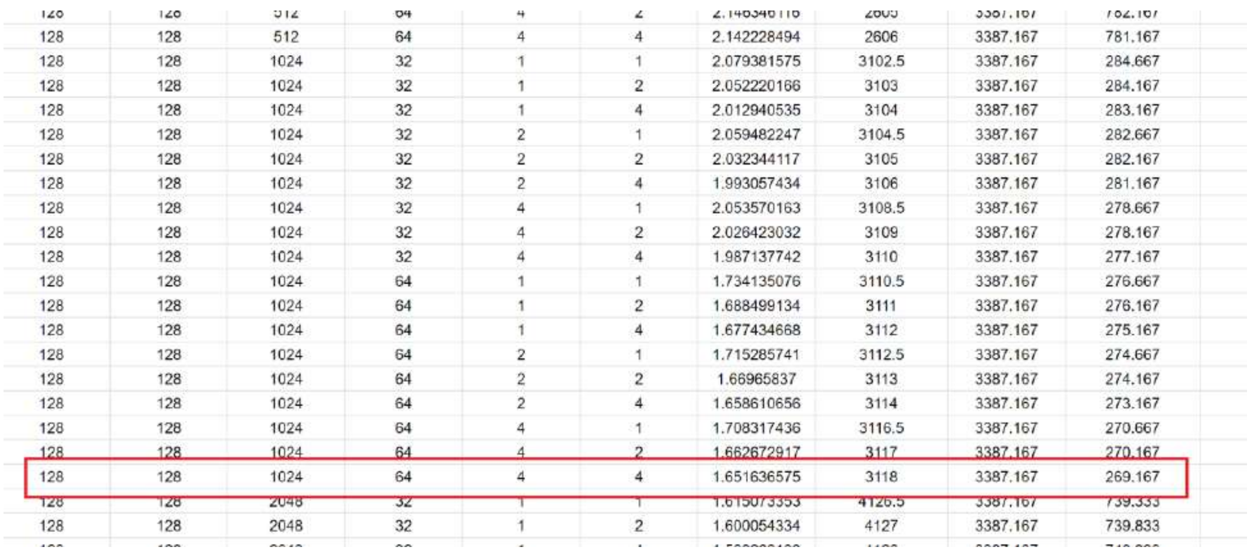
15. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 458.sjeng and cpu-type – Minor. Highlighted part shows the best choice. For full data-set look at “458sjeng_MinorCPU” in dataset folder.



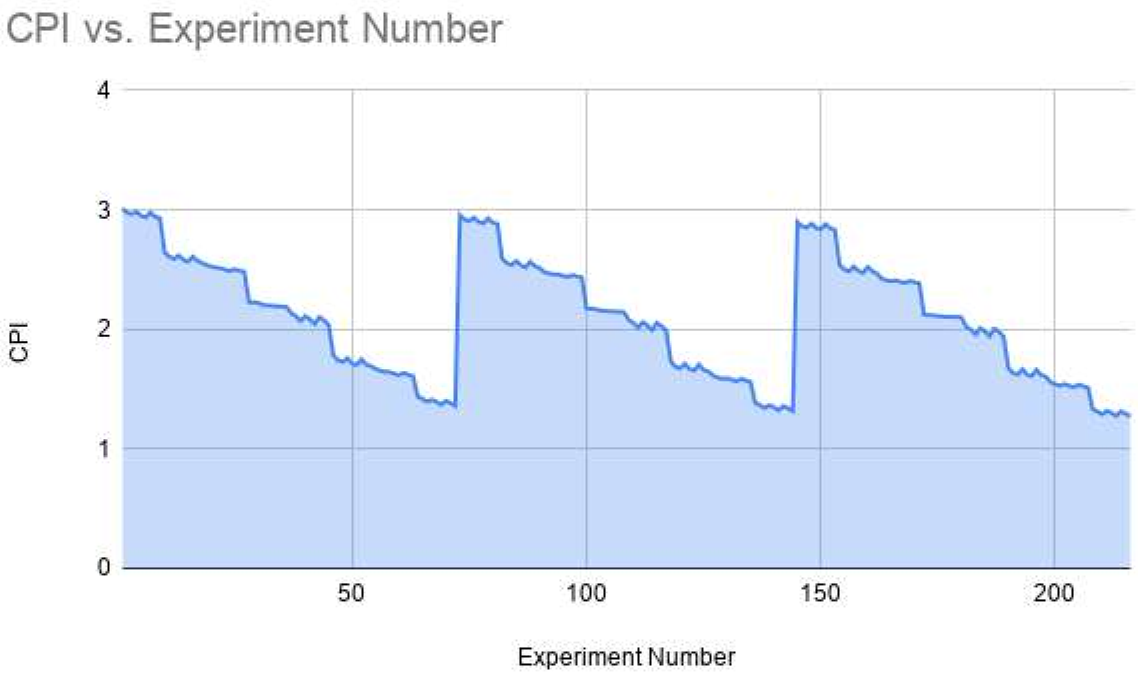
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
128	128	512	64	2	4	2.173969835	2602	3387.167	785.167
128	128	512	64	4	1	2.173352946	2604.5	3387.167	782.667
128	128	512	64	4	2	2.170969473	2605	3387.167	782.167
128	128	512	64	4	4	2.166776258	2606	3387.167	781.167
128	128	1024	32	1	1	2.103005577	3102.5	3387.167	284.667
128	128	1024	32	1	2	2.075295916	3103	3387.167	284.167
128	128	1024	32	1	4	2.035213746	3104	3387.167	283.167
128	128	1024	32	2	1	2.082519442	3104.5	3387.167	282.667
128	128	1024	32	2	2	2.054821841	3105	3387.167	282.167
128	128	1024	32	2	4	2.014746166	3106	3387.167	281.167
128	128	1024	32	4	1	2.076422936	3108.5	3387.167	278.667
128	128	1024	32	4	2	2.04872041	3109	3387.167	278.167
128	128	1024	32	4	4	2.008644209	3110	3387.167	277.167
128	128	1024	64	1	1	1.75057907	3110.5	3387.167	276.667
128	128	1024	64	1	2	1.704009779	3111	3387.167	276.167
128	128	1024	64	1	4	1.692735256	3112	3387.167	275.167
128	128	1024	64	2	1	1.731200543	3112.5	3387.167	274.667
128	128	1024	64	2	2	1.684630173	3113	3387.167	274.167
128	128	1024	64	2	4	1.673346066	3114	3387.167	273.167
128	128	1024	64	4	1	1.724010436	3116.5	3387.167	270.667
128	128	1024	64	4	2	1.677441779	3117	3387.167	270.167
128	128	1024	64	4	4	1.666163807	3118	3387.167	269.167
128	128	2048	32	1	1	1.629222357	4126.5	3387.167	739.333
128	128	2048	32	1	2	1.613903572	4127	3387.167	739.833



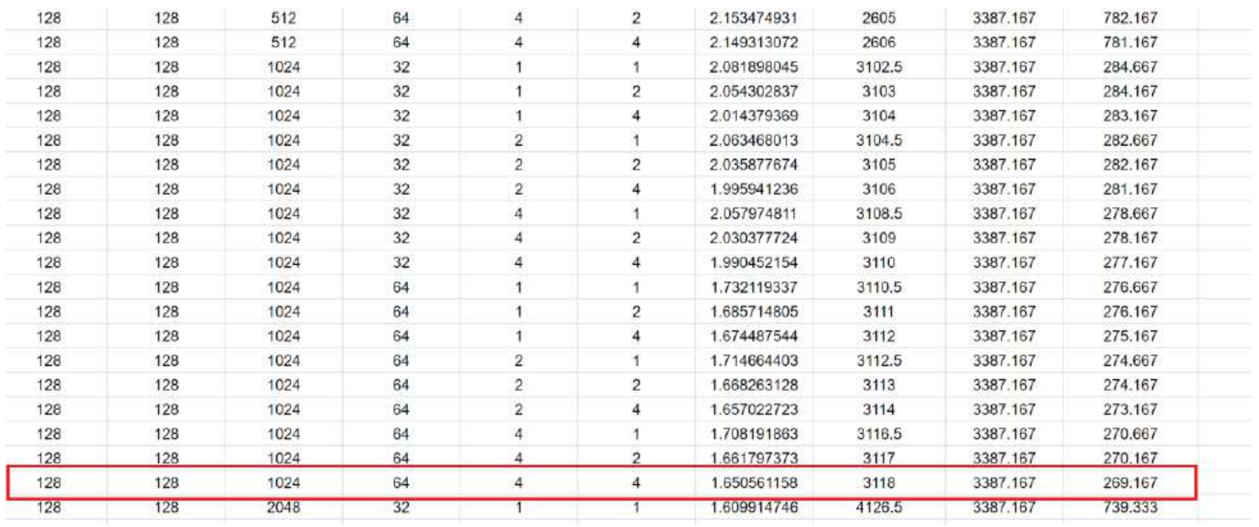
16. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 458.sjeng and cpu-type – TimingSimple. Highlighted part shows the best choice. For full data-set look at "458sjeng_TimingSimpleCPU" in dataset folder.



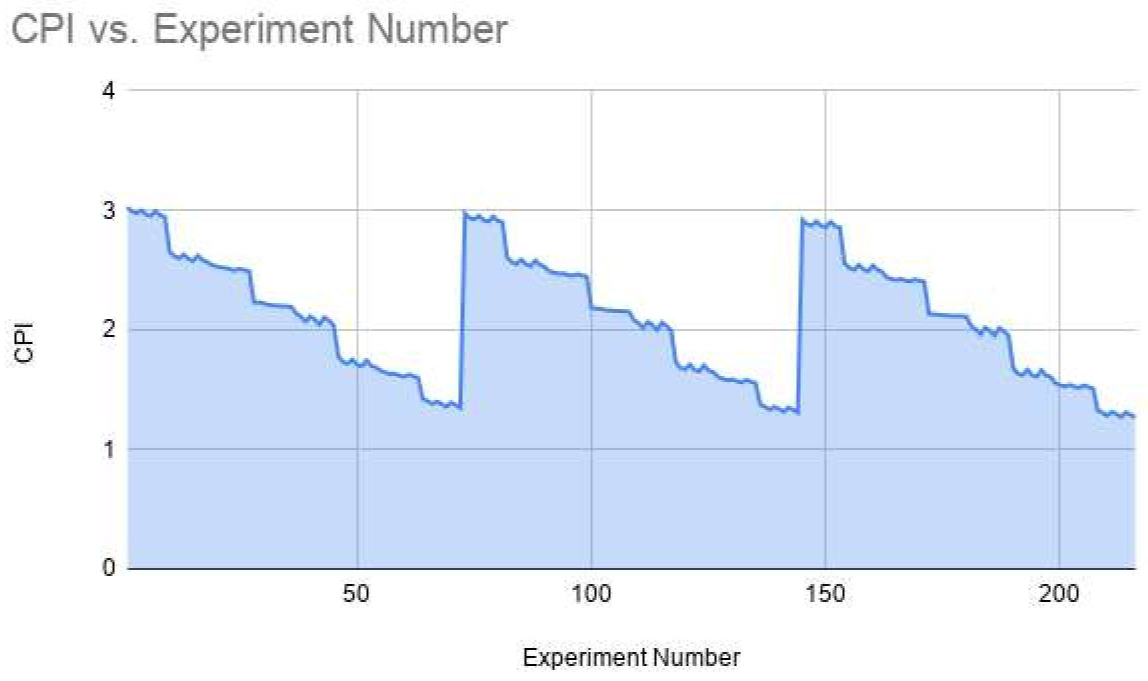
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
128	128	512	64	4	2	2.146346116	2605	3387.167	782.167
128	128	512	64	4	4	2.142228494	2606	3387.167	781.167
128	128	1024	32	1	1	2.079381575	3102.5	3387.167	284.667
128	128	1024	32	1	2	2.052220166	3103	3387.167	284.167
128	128	1024	32	1	4	2.012940535	3104	3387.167	283.167
128	128	1024	32	2	1	2.059482247	3104.5	3387.167	282.667
128	128	1024	32	2	2	2.032344117	3105	3387.167	282.167
128	128	1024	32	2	4	1.993057434	3106	3387.167	281.167
128	128	1024	32	4	1	2.053570163	3108.5	3387.167	278.667
128	128	1024	32	4	2	2.026423032	3109	3387.167	278.167
128	128	1024	32	4	4	1.987137742	3110	3387.167	277.167
128	128	1024	64	1	1	1.734135076	3110.5	3387.167	276.667
128	128	1024	64	1	2	1.688499134	3111	3387.167	276.167
128	128	1024	64	1	4	1.677434668	3112	3387.167	275.167
128	128	1024	64	2	1	1.715285741	3112.5	3387.167	274.667
128	128	1024	64	2	2	1.66965837	3113	3387.167	274.167
128	128	1024	64	2	4	1.658610656	3114	3387.167	273.167
128	128	1024	64	4	1	1.708317436	3116.5	3387.167	270.667
128	128	1024	64	4	2	1.662672917	3117	3387.167	270.167
128	128	1024	64	4	4	1.651636575	3118	3387.167	269.167
128	128	2048	32	1	1	1.615073353	4126.5	3387.167	739.333
128	128	2048	32	1	2	1.600054334	4127	3387.167	739.833
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]



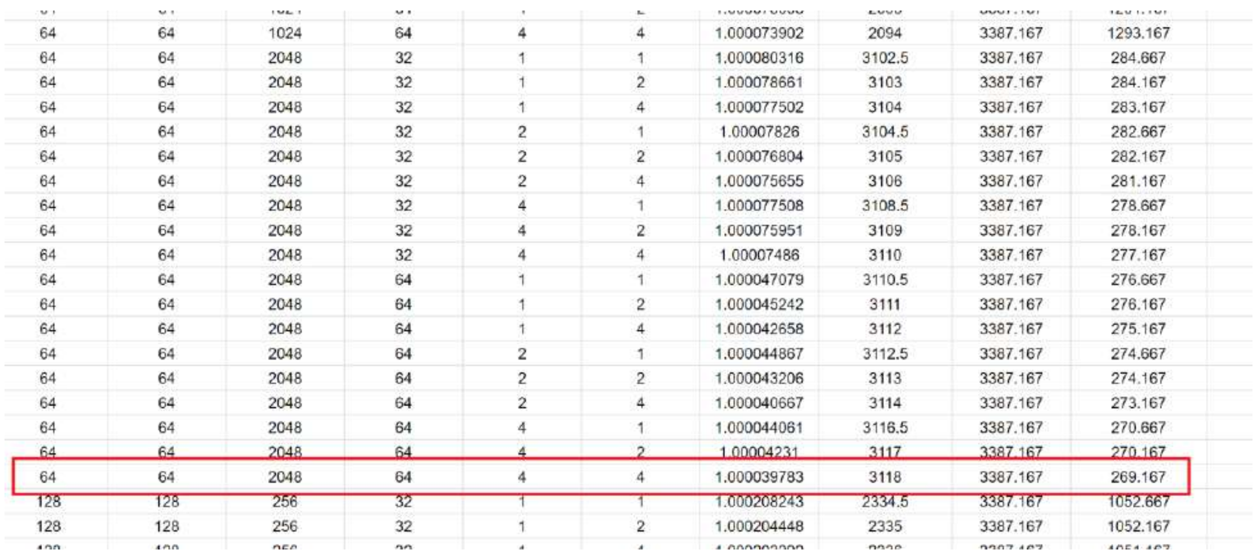
17. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 458.sjeng and cpu-type – DerivO3. Highlighted part shows the best choice. For full data-set look at “458sjeng_DerivO3CPU” in dataset folder.



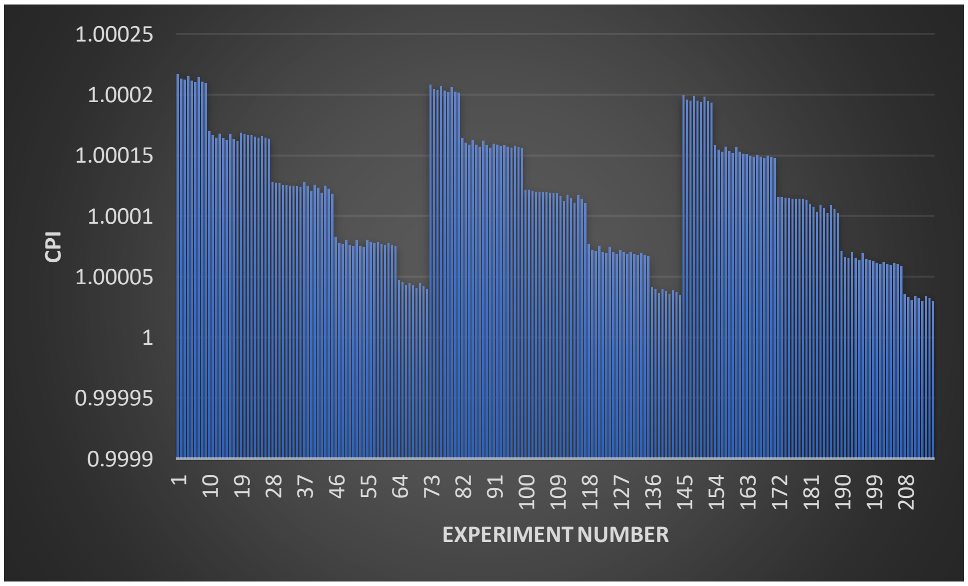
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
128	128	512	64	4	2	2.153474931	2605	3387.167	782.167
128	128	512	64	4	4	2.149313072	2606	3387.167	781.167
128	128	1024	32	1	1	2.081898045	3102.5	3387.167	284.667
128	128	1024	32	1	2	2.054302837	3103	3387.167	284.167
128	128	1024	32	1	4	2.014379369	3104	3387.167	283.167
128	128	1024	32	2	1	2.063468013	3104.5	3387.167	282.667
128	128	1024	32	2	2	2.035877674	3105	3387.167	282.167
128	128	1024	32	2	4	1.995941236	3106	3387.167	281.167
128	128	1024	32	4	1	2.057974811	3108.5	3387.167	278.667
128	128	1024	32	4	2	2.030377724	3109	3387.167	278.167
128	128	1024	32	4	4	1.990452154	3110	3387.167	277.167
128	128	1024	64	1	1	1.732119337	3110.5	3387.167	276.667
128	128	1024	64	1	2	1.685714805	3111	3387.167	276.167
128	128	1024	64	1	4	1.674487544	3112	3387.167	275.167
128	128	1024	64	2	1	1.714664403	3112.5	3387.167	274.667
128	128	1024	64	2	2	1.668263128	3113	3387.167	274.167
128	128	1024	64	2	4	1.657022723	3114	3387.167	273.167
128	128	1024	64	4	1	1.708191863	3116.5	3387.167	270.667
128	128	1024	64	4	2	1.661797373	3117	3387.167	270.167
128	128	1024	64	4	4	1.650561158	3118	3387.167	269.167
128	128	2048	32	1	1	1.609914746	4126.5	3387.167	739.333



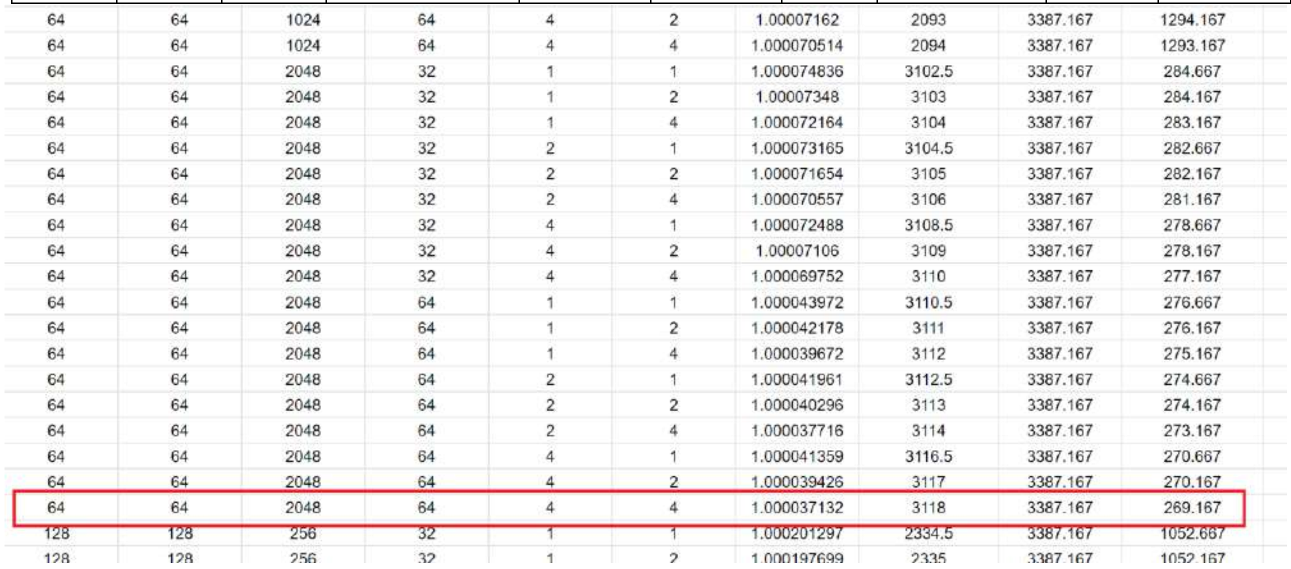
18. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 470.lbm and cpu-type – Minor. Highlighted part shows the best choice. For full data-set look at “470lbm_MinorCPU” in dataset folder.



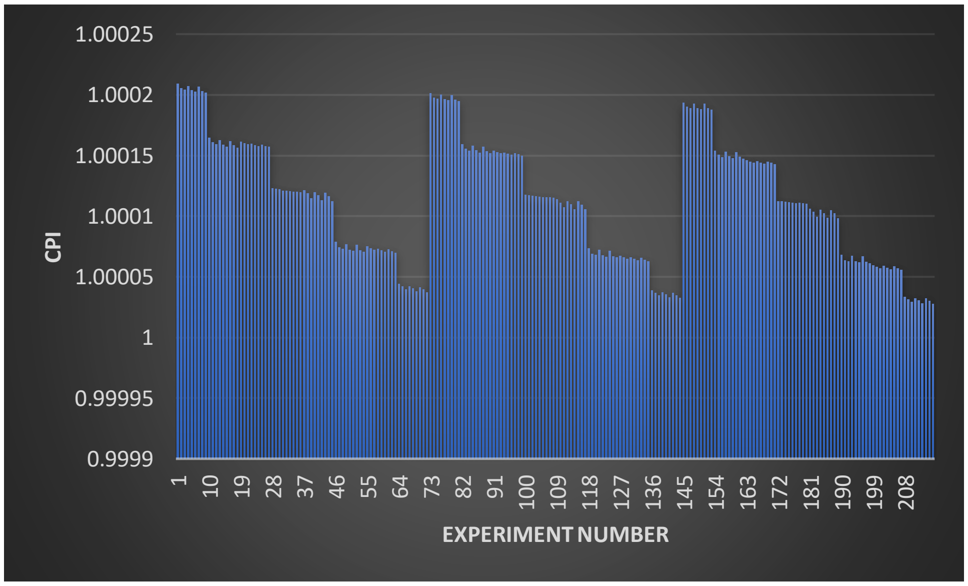
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
64	64	1024	64	4	4	1.000073902	2094	3387.167	1293.167
64	64	2048	32	1	1	1.000080316	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.000078661	3103	3387.167	284.167
64	64	2048	32	1	4	1.000077502	3104	3387.167	283.167
64	64	2048	32	2	1	1.00007826	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.000076804	3105	3387.167	282.167
64	64	2048	32	2	4	1.000075655	3106	3387.167	281.167
64	64	2048	32	4	1	1.000077508	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.000075951	3109	3387.167	278.167
64	64	2048	32	4	4	1.00007486	3110	3387.167	277.167
64	64	2048	64	1	1	1.000047079	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.000045242	3111	3387.167	276.167
64	64	2048	64	1	4	1.000042658	3112	3387.167	275.167
64	64	2048	64	2	1	1.000044867	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.000043206	3113	3387.167	274.167
64	64	2048	64	2	4	1.000040667	3114	3387.167	273.167
64	64	2048	64	4	1	1.000044061	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.00004231	3117	3387.167	270.167
64	64	2048	64	4	4	1.000039783	3118	3387.167	269.167
128	128	256	32	1	1	1.000208243	2334.5	3387.167	1052.667
128	128	256	32	1	2	1.000204448	2335	3387.167	1052.167



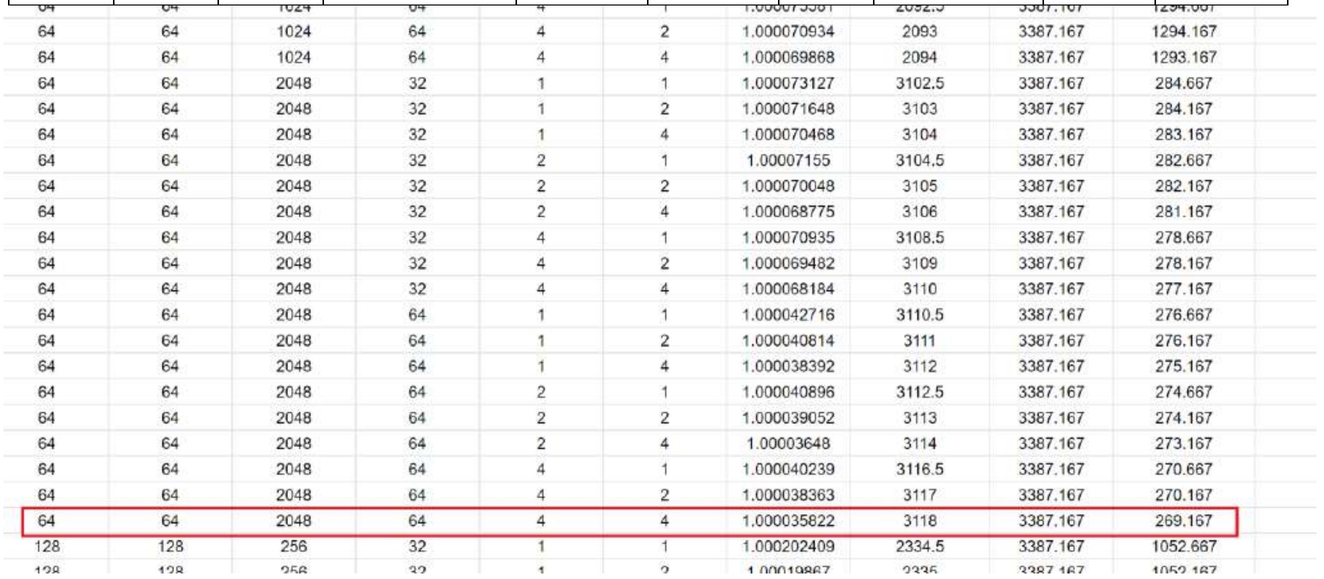
19. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 470.lbm and cpu-type – TimingSimple. Highlighted part shows the best choice. For full data-set look at “470lbm_TimingSimpleCPU” in dataset folder.



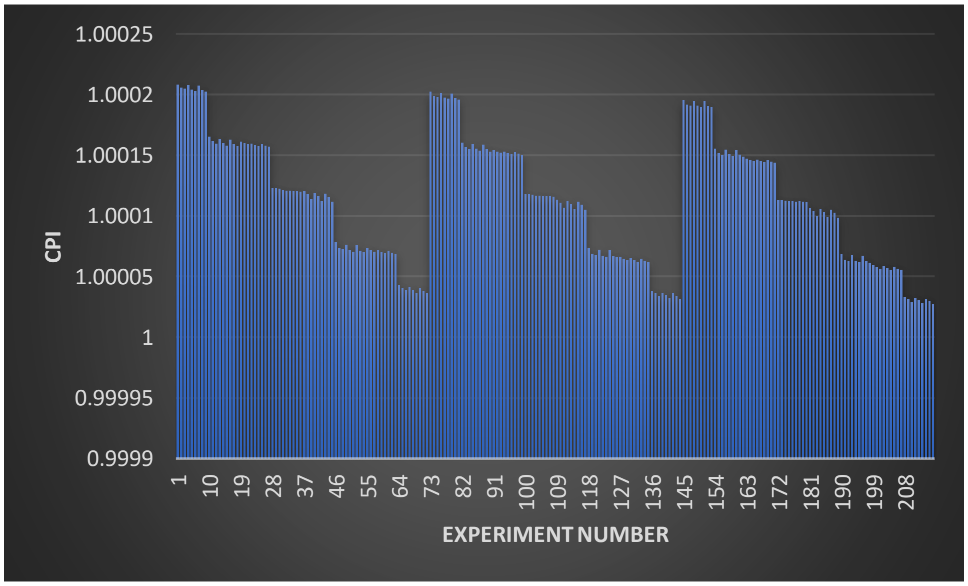
IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
64	64	1024	64	4	2	1.00007162	2093	3387.167	1294.167
64	64	1024	64	4	4	1.000070514	2094	3387.167	1293.167
64	64	2048	32	1	1	1.000074836	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.00007348	3103	3387.167	284.167
64	64	2048	32	1	4	1.000072164	3104	3387.167	283.167
64	64	2048	32	2	1	1.000073165	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.000071654	3105	3387.167	282.167
64	64	2048	32	2	4	1.000070557	3106	3387.167	281.167
64	64	2048	32	4	1	1.000072488	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.00007106	3109	3387.167	278.167
64	64	2048	32	4	4	1.000069752	3110	3387.167	277.167
64	64	2048	64	1	1	1.000043972	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.000042178	3111	3387.167	276.167
64	64	2048	64	1	4	1.000039672	3112	3387.167	275.167
64	64	2048	64	2	1	1.000041961	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.000040296	3113	3387.167	274.167
64	64	2048	64	2	4	1.000037716	3114	3387.167	273.167
64	64	2048	64	4	1	1.000041359	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.000039426	3117	3387.167	270.167
64	64	2048	64	4	4	1.000037132	3118	3387.167	269.167
128	128	256	32	1	1	1.000201297	2334.5	3387.167	1052.667
128	128	256	32	1	2	1.000197699	2335	3387.167	1052.167



20. In this section, we represent the experiments and tradeoff performed and analyzed over various cache configuration in benchmark – 470.lbm and cpu-type – DerivO3. Highlighted part shows the best choice. For full data-set look at "470lbm_DerivO3CPU" in dataset folder.



IL1 Size	DL1 Size	L2 Size	Cacheline Size	L1 Assoc	L2 Assoc	CPI	Cost Function	Avg Cost	Mean diff
64	64	1024	64	4	1	[illegible]	2092.5	3387.167	1294.667
64	64	1024	64	4	2	1.000070934	2093	3387.167	1294.167
64	64	1024	64	4	4	1.000069868	2094	3387.167	1293.167
64	64	2048	32	1	1	1.000073127	3102.5	3387.167	284.667
64	64	2048	32	1	2	1.000071648	3103	3387.167	284.167
64	64	2048	32	1	4	1.000070468	3104	3387.167	283.167
64	64	2048	32	2	1	1.00007155	3104.5	3387.167	282.667
64	64	2048	32	2	2	1.000070048	3105	3387.167	282.167
64	64	2048	32	2	4	1.000068775	3106	3387.167	281.167
64	64	2048	32	4	1	1.000070935	3108.5	3387.167	278.667
64	64	2048	32	4	2	1.000069482	3109	3387.167	278.167
64	64	2048	32	4	4	1.000068184	3110	3387.167	277.167
64	64	2048	64	1	1	1.000042716	3110.5	3387.167	276.667
64	64	2048	64	1	2	1.000040814	3111	3387.167	276.167
64	64	2048	64	1	4	1.000038392	3112	3387.167	275.167
64	64	2048	64	2	1	1.000040896	3112.5	3387.167	274.667
64	64	2048	64	2	2	1.000039052	3113	3387.167	274.167
64	64	2048	64	2	4	1.00003648	3114	3387.167	273.167
64	64	2048	64	4	1	1.000040239	3116.5	3387.167	270.667
64	64	2048	64	4	2	1.000038363	3117	3387.167	270.167
64	64	2048	64	4	4	1.000035822	3118	3387.167	269.167
128	128	256	32	1	1	1.000202409	2334.5	3387.167	1052.667
128	128	256	32	1	2	1.00019867	2335	3387.167	1052.167



Conclusion

With the increase in cache size CPI decreases along with the miss rate as more data can be store in the cache at a time. Also, if we increase the cache-line size the value of the CPI decreases. Although these things happen, with such changes cost of the CPU also increases.

Therefore, to determine the optimum cache configuration, we created a cost function and evaluation function which tells which configuration is cheapest and the best.

Overall for each benchmark, below is the best configuration of CPU and cache configuration.

- 401.bzip2 -> TimingSimpleCPU (L1 = 64kB, L2 = 2048kB, Cache-Line = 64, $L1_{assoc} = 4$, $L2_{assoc} = 4$)
- 429.mcf -> DerivO3CPU (L1 = 64kB, L2 = 2048kB, Cache-Line = 64, $L1_{assoc} = 2$, $L2_{assoc} = 4$)
- 456.hmmer -> DerivO3CPU (L1 = 64kB, L2 = 2048kB, Cache-Line = 64, $L1_{assoc} = 4$, $L2_{assoc} = 4$)
- 458.sjeng -> DerivO3CPU (L1 = 128kB, L2 = 1024kB, Cache-Line = 64, $L1_{assoc} = 4$, $L2_{assoc} = 4$)
- 470.lbm -> DerivO3CPU (L1 = 64kB, L2 = 2048kB, Cache-Line = 64, $L1_{assoc} = 4$, $L2_{assoc} = 4$)

References (Research Articles)

1) http://www.cs.ucr.edu/~gupta/hpca9/HPCA-PDFs/30-jeong.pdf
2) https://pdfs.semanticscholar.org/8b3c/ba0d289955488ddc022d14050a3786afb8ed.pdf
3) http://pages.cs.wisc.edu/~markhill/papers/toc89_cpu_cache_associativity.pdf
4) https://www.diva-portal.org/smash/get/diva2:116965/FULLTEXT01.pdf

Appendix

Experiment Number	IL1 Size (kB)	DL1 Size (kB)	L2 Size (kB)	Cache-line Size	L1 Associativit y	L2 Associativit y
1	64	64	256	32	1	1
2	64	64	256	32	1	2
3	64	64	256	32	1	4
4	64	64	256	32	2	1
5	64	64	256	32	2	2
6	64	64	256	32	2	4
7	64	64	256	32	4	1
8	64	64	256	32	4	2
9	64	64	256	32	4	4
10	64	64	256	64	1	1
11	64	64	256	64	1	2
12	64	64	256	64	1	4
13	64	64	256	64	2	1
14	64	64	256	64	2	2
15	64	64	256	64	2	4
16	64	64	256	64	4	1
17	64	64	256	64	4	2
18	64	64	256	64	4	4
19	64	64	512	32	1	1
20	64	64	512	32	1	2
21	64	64	512	32	1	4
22	64	64	512	32	2	1
23	64	64	512	32	2	2
24	64	64	512	32	2	4
25	64	64	512	32	4	1
26	64	64	512	32	4	2
27	64	64	512	32	4	4
28	64	64	512	64	1	1
29	64	64	512	64	1	2
30	64	64	512	64	1	4
31	64	64	512	64	2	1
32	64	64	512	64	2	2
33	64	64	512	64	2	4
34	64	64	512	64	4	1

35	64	64	512	64	4	2
36	64	64	512	64	4	4
37	64	64	1024	32	1	1
38	64	64	1024	32	1	2
39	64	64	1024	32	1	4
40	64	64	1024	32	2	1
41	64	64	1024	32	2	2
42	64	64	1024	32	2	4
43	64	64	1024	32	4	1
44	64	64	1024	32	4	2
45	64	64	1024	32	4	4
46	64	64	1024	64	1	1
47	64	64	1024	64	1	2
48	64	64	1024	64	1	4
49	64	64	1024	64	2	1
50	64	64	1024	64	2	2
51	64	64	1024	64	2	4
52	64	64	1024	64	4	1
53	64	64	1024	64	4	2
54	64	64	1024	64	4	4
55	64	64	2048	32	1	1
56	64	64	2048	32	1	2
57	64	64	2048	32	1	4
58	64	64	2048	32	2	1
59	64	64	2048	32	2	2
60	64	64	2048	32	2	4
61	64	64	2048	32	4	1
62	64	64	2048	32	4	2
63	64	64	2048	32	4	4
64	64	64	2048	64	1	1
65	64	64	2048	64	1	2
66	64	64	2048	64	1	4
67	64	64	2048	64	2	1
68	64	64	2048	64	2	2
69	64	64	2048	64	2	4
70	64	64	2048	64	4	1
71	64	64	2048	64	4	2
72	64	64	2048	64	4	4
73	128	128	256	32	1	1

74	128	128	256	32	1	2
75	128	128	256	32	1	4
76	128	128	256	32	2	1
77	128	128	256	32	2	2
78	128	128	256	32	2	4
79	128	128	256	32	4	1
80	128	128	256	32	4	2
81	128	128	256	32	4	4
82	128	128	256	64	1	1
83	128	128	256	64	1	2
84	128	128	256	64	1	4
85	128	128	256	64	2	1
86	128	128	256	64	2	2
87	128	128	256	64	2	4
88	128	128	256	64	4	1
89	128	128	256	64	4	2
90	128	128	256	64	4	4
91	128	128	512	32	1	1
92	128	128	512	32	1	2
93	128	128	512	32	1	4
94	128	128	512	32	2	1
95	128	128	512	32	2	2
96	128	128	512	32	2	4
97	128	128	512	32	4	1
98	128	128	512	32	4	2
99	128	128	512	32	4	4
100	128	128	512	64	1	1
101	128	128	512	64	1	2
102	128	128	512	64	1	4
103	128	128	512	64	2	1
104	128	128	512	64	2	2
105	128	128	512	64	2	4
106	128	128	512	64	4	1
107	128	128	512	64	4	2
108	128	128	512	64	4	4
109	128	128	1024	32	1	1
110	128	128	1024	32	1	2
111	128	128	1024	32	1	4
112	128	128	1024	32	2	1

113	128	128	1024	32	2	2
114	128	128	1024	32	2	4
115	128	128	1024	32	4	1
116	128	128	1024	32	4	2
117	128	128	1024	32	4	4
118	128	128	1024	64	1	1
119	128	128	1024	64	1	2
120	128	128	1024	64	1	4
121	128	128	1024	64	2	1
122	128	128	1024	64	2	2
123	128	128	1024	64	2	4
124	128	128	1024	64	4	1
125	128	128	1024	64	4	2
126	128	128	1024	64	4	4
127	128	128	2048	32	1	1
128	128	128	2048	32	1	2
129	128	128	2048	32	1	4
130	128	128	2048	32	2	1
131	128	128	2048	32	2	2
132	128	128	2048	32	2	4
133	128	128	2048	32	4	1
134	128	128	2048	32	4	2
135	128	128	2048	32	4	4
136	128	128	2048	64	1	1
137	128	128	2048	64	1	2
138	128	128	2048	64	1	4
139	128	128	2048	64	2	1
140	128	128	2048	64	2	2
141	128	128	2048	64	2	4
142	128	128	2048	64	4	1
143	128	128	2048	64	4	2
144	128	128	2048	64	4	4
145	256	256	256	32	1	1
146	256	256	256	32	1	2
147	256	256	256	32	1	4
148	256	256	256	32	2	1
149	256	256	256	32	2	2
150	256	256	256	32	2	4
151	256	256	256	32	4	1

152	256	256	256	32	4	2
153	256	256	256	32	4	4
154	256	256	256	64	1	1
155	256	256	256	64	1	2
156	256	256	256	64	1	4
157	256	256	256	64	2	1
158	256	256	256	64	2	2
159	256	256	256	64	2	4
160	256	256	256	64	4	1
161	256	256	256	64	4	2
162	256	256	256	64	4	4
163	256	256	512	32	1	1
164	256	256	512	32	1	2
165	256	256	512	32	1	4
166	256	256	512	32	2	1
167	256	256	512	32	2	2
168	256	256	512	32	2	4
169	256	256	512	32	4	1
170	256	256	512	32	4	2
171	256	256	512	32	4	4
172	256	256	512	64	1	1
173	256	256	512	64	1	2
174	256	256	512	64	1	4
175	256	256	512	64	2	1
176	256	256	512	64	2	2
177	256	256	512	64	2	4
178	256	256	512	64	4	1
179	256	256	512	64	4	2
180	256	256	512	64	4	4
181	256	256	1024	32	1	1
182	256	256	1024	32	1	2
183	256	256	1024	32	1	4
184	256	256	1024	32	2	1
185	256	256	1024	32	2	2
186	256	256	1024	32	2	4
187	256	256	1024	32	4	1
188	256	256	1024	32	4	2
189	256	256	1024	32	4	4
190	256	256	1024	64	1	1

191	256	256	1024	64	1	2
192	256	256	1024	64	1	4
193	256	256	1024	64	2	1
194	256	256	1024	64	2	2
195	256	256	1024	64	2	4
196	256	256	1024	64	4	1
197	256	256	1024	64	4	2
198	256	256	1024	64	4	4
199	256	256	2048	32	1	1
200	256	256	2048	32	1	2
201	256	256	2048	32	1	4
202	256	256	2048	32	2	1
203	256	256	2048	32	2	2
204	256	256	2048	32	2	4
205	256	256	2048	32	4	1
206	256	256	2048	32	4	2
207	256	256	2048	32	4	4
208	256	256	2048	64	1	1
209	256	256	2048	64	1	2
210	256	256	2048	64	1	4
211	256	256	2048	64	2	1
212	256	256	2048	64	2	2
213	256	256	2048	64	2	4
214	256	256	2048	64	4	1
215	256	256	2048	64	4	2
216	256	256	2048	64	4	4