PROJECT 2: "FINE-TUNING CACHE HIERARCHY ON X86 ARCHITECTURE USING GEM5 SIMULATOR"

CE6304: COMPUTER ARCHITECTURE

Department of Electrical and Computer Engineering

Submitted By:
Srivishnu Srinivas (sxs220196)
Prathamesh Gadad (psg220003)

OVERVIEW:

- Part 1 : Description of Gem5 Setup and Cache.
- Part 2 : Finding CPI.
- Part 3 : Optimize CPI for Benchmark.
- Part 4 : Define Cost Function.
- Part 5 : Evaluation Function Showing Trade Offs
  (CPI Vs Cache Parameter Trade Off Curves And Analysis/Discussion).

PART 1: Description of GEM5 Setup And Cache

Initial Gem5 Setup:

- For this project, we used Gem5 installed on the UTD server with the necessary dependencies. We copied Gem5 to our local directory using the command:

  "cp–rf/usr/local/gem5/home/eng/s/sxs220196/CA_Project“.

- We then compiled it with "scons build/X86/gem5.opt".
- We downloaded the 456.hmmer and 458.sjeng benchmark files from the provided GitHub link; “git clone https://github.com/timberjack/Project1_SPEC.git”. to our local directory for use in this project.
- We set up the Gem5 simulation environment, compiled it, and obtained standard benchmark programs to run simulations for this project.

What is Cache?

A cache is on-chip memory that stores data and instructions to speed up subsequent requests. Cached data may be the output of a previous computation or a copy of data stored elsewhere. Caches use fast SRAM technology compared to slower DRAM.

Terminologies:

- Cache Block/Line - Minimum cache allocation unit.
- Hit - Data found in cache.
- Hit Rate - Fraction of accesses served by cache.
- Hit Time - Time to access cache.
- Miss Rate - (1 - Hit Rate).
- Miss Penalty - Time to fetch data and load cache.
- Average Memory Access Time (AMAT) = Hit Time + (Miss Rate * Miss Penalty).

Importance of Cache in Today's Computer System:

Growing gap between fast processors and slow memory. Caches bridge gap by storing frequently used data/instructions close to processor. Enable low-latency access, hiding memory latency. Absorb majority of memory requests. feeding processors at needed speeds. Make modern processor speeds possible.

PART 2: Finding CPI From Equation

- Equation to calculate Cycles Per Instructions (CPI):

  CPI = *1 + ((IL1.miss_num + DL1.miss_num)*6)+(L2.miss_num*50)/Total_Inst_num).*

- Changes made in the "runGem.sh" files for the given Benchmarks.

  - 456.hmmer:

    $GEM5_DIR/build/X86/gem5.opt -d /home/eng/s/sxs220196/CA_Project/OutputsP3/Outputs_456_2/456L1S_128_128_L2S_8192_L1a_4_L2a_2_cbs_64 $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 50000000 --cpu-type=timing --caches --l2cache --l1d_size=128kB --l1i_size=128kB --l2_size=8192kB --l1d_assoc=4 --l1i_assoc=4 --l2_assoc=2 --cacheline_size=64.

  - 458.sjeng:

    $GEM5_DIR/build/X86/gem5.opt -d /home/eng/s/sxs220196/CA_Project/OutputsP3/Outputs_458_1/458L1S_64_128_L2S_8192_L1a_4_L2a_2_cbs_64 $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 50000000 --cpu-type=timing --caches --l2cache --l1d_size=64kB --l1i_size=128kB --l2_size=8192kB --l1d_assoc=4 --l1i_assoc=4 --l2_assoc=2 --cacheline_size=64

- Run the Script "sh runGem5.sh".

For 456.hmmer:

$CPI=1+((IL1.miss\_num+DL1.miss\_num)*6)+(L2.miss\_num*50)/Total\_Inst\_num).$

```
system.cpu.dcache.overall_misses::total       199525                    # number of overall misses
system.cpu.icache.overall_misses::total         1331                    # number of overall misses
system.l2.overall_misses::total                 7036                    # number of overall misses
```

$CPI = 1 + ((199525+1331)*6)+(7036*50)/50000000)$

$= 1.01118622$

For 458.sjeng:

CPI=*1+((IL1.miss_num+DL1.miss_num)*6)+(L2.miss_num*50)/Total_Inst_num).*

```
system.cpu.dcache.overall_misses::total      8422909                          # number of overall misses
system.cpu.icache.overall_misses::total         2773                          # number of overall misses
system.l2.overall_misses::total              8381043                          # number of overall misses
```

CPI = *1 + ((8422909+2773)*6)+(8381043*50)/50000000)*

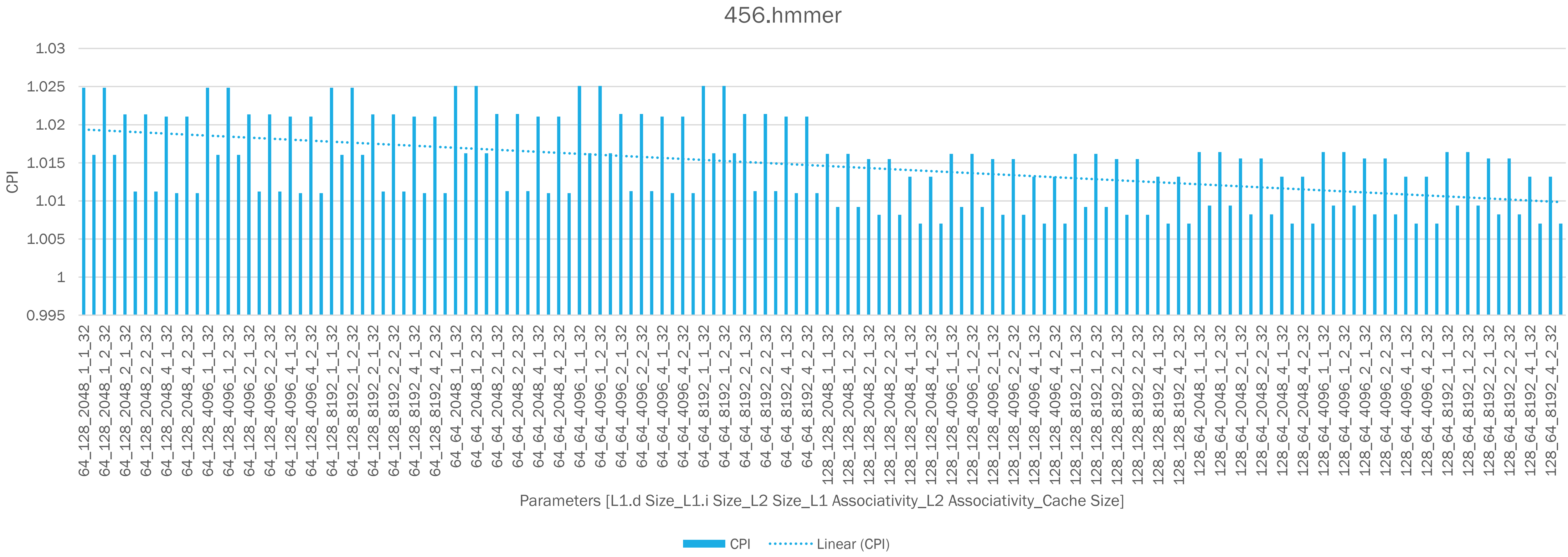
= ***9.549833***

PART 3: Optimize CPI For Benchmark

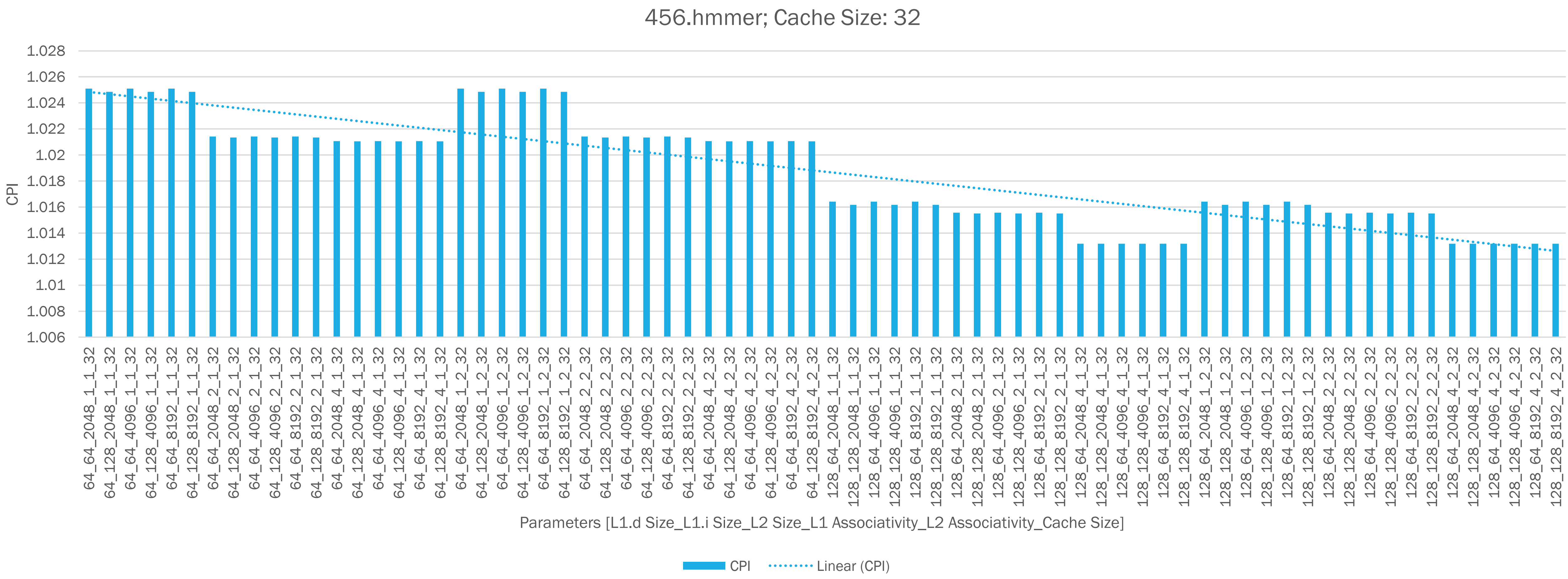
Configuration:

Benchmarks	L1d	L1i	L2	L1a	L2a	Cache Size
456.hmmer	64, 128	64, 128	2048, 4096, 8192	1, 2, 4	1, 2	32, 64
458.sjeng	64, 128	64, 128	2048, 4096, 8192	1, 2, 4	1, 2	32, 64

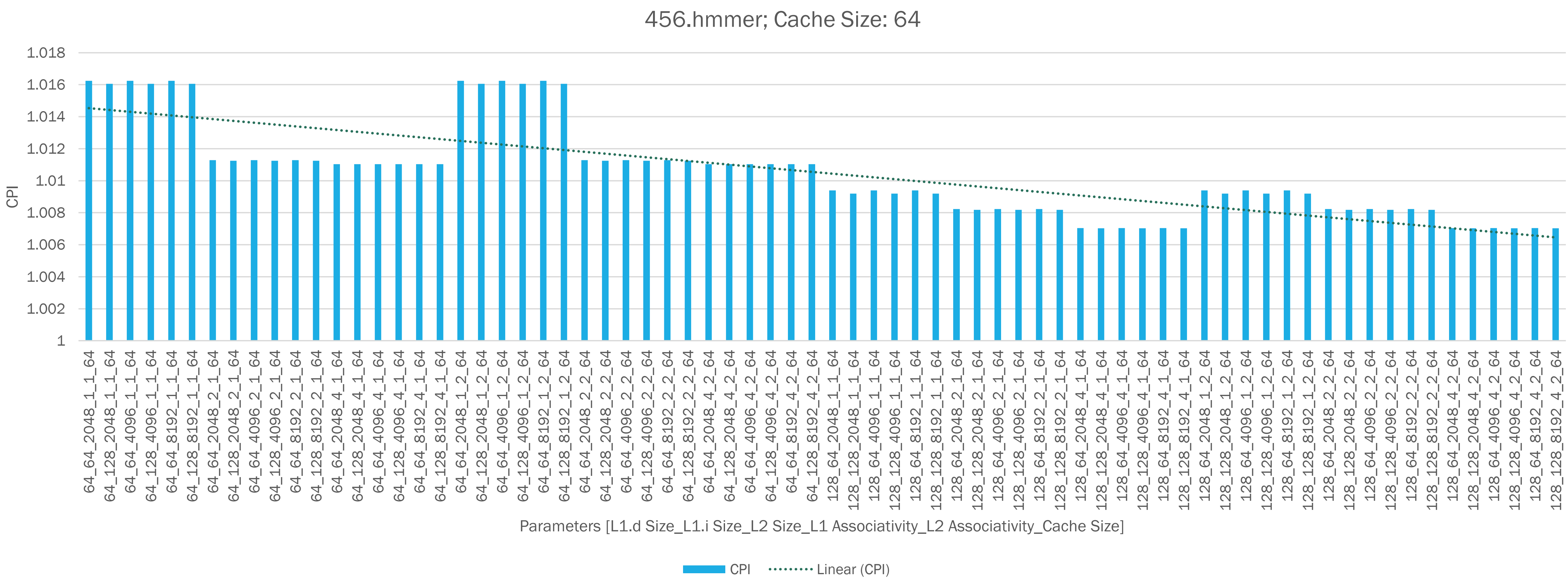
For 456.hmmer:



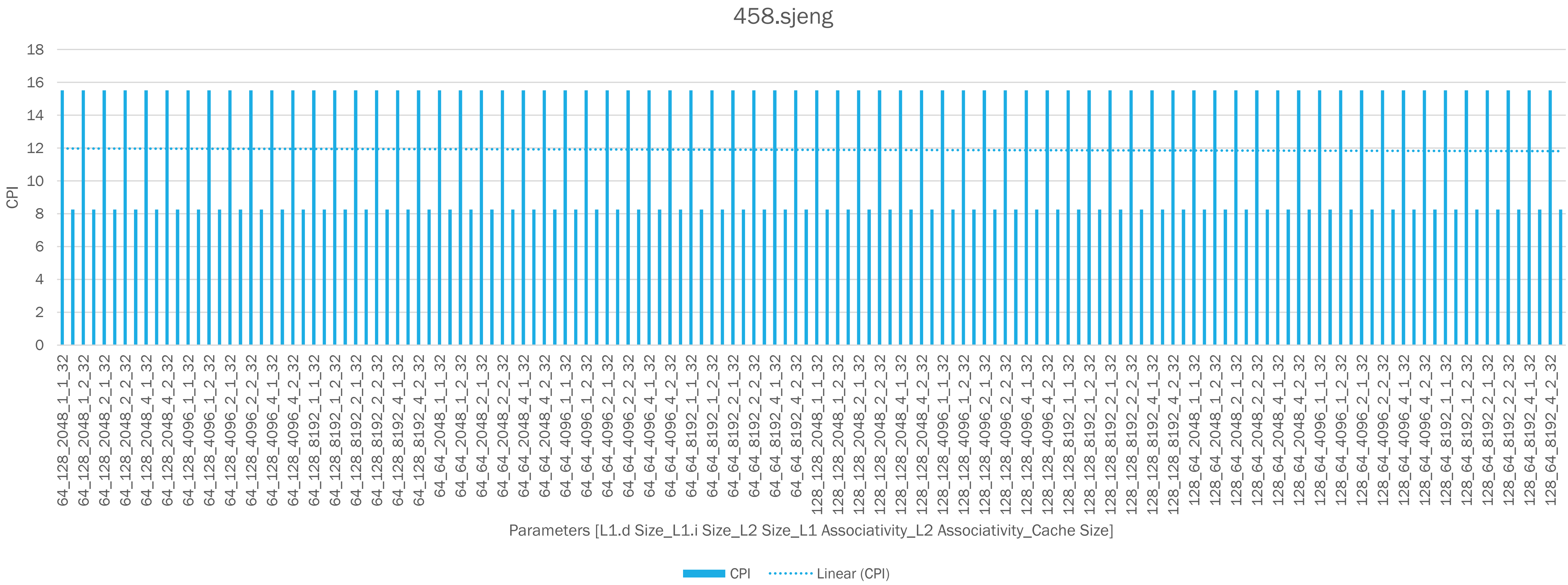
For 456.hmmer:



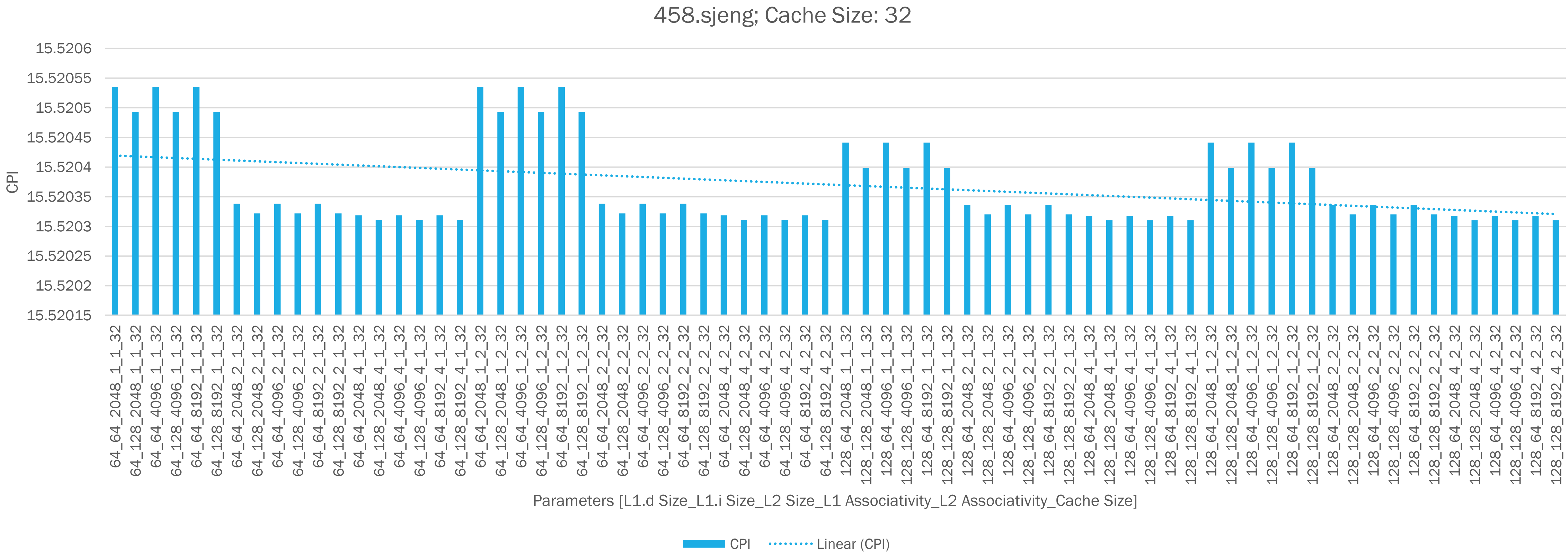
For 456.hmmer:



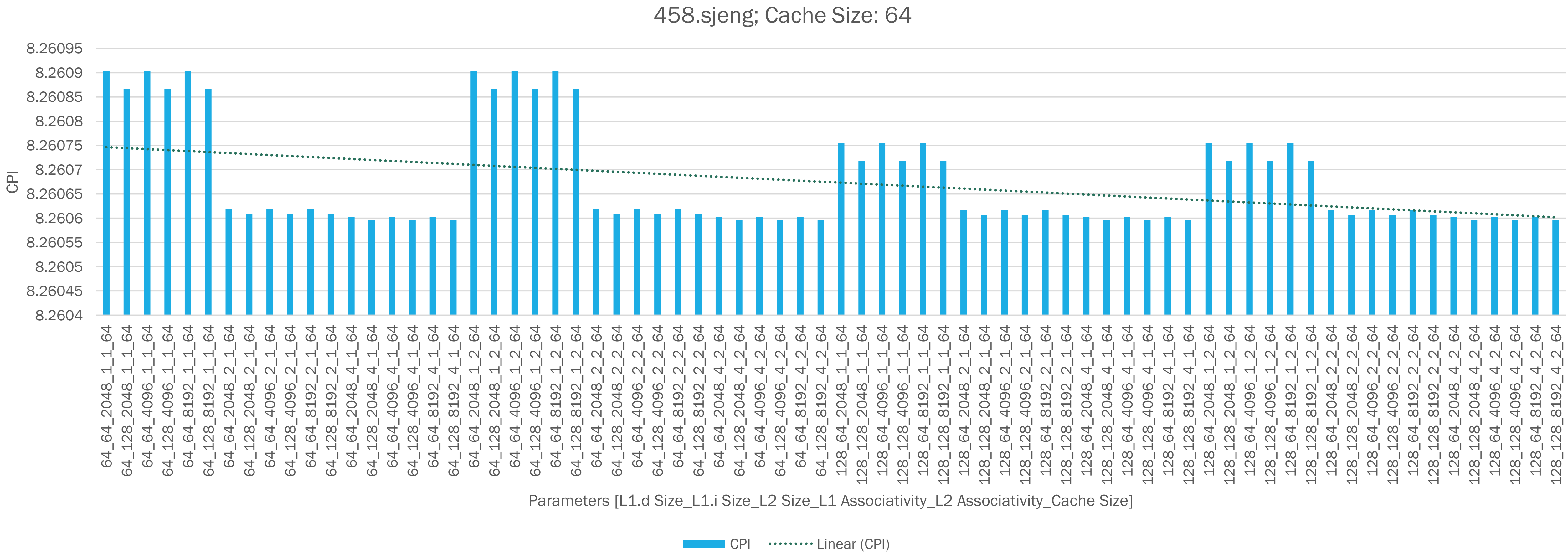
For 458.sjeng:



For 458.sjeng:



For 458.sjeng:



From the Graphs, for 456.hmmer and 458.sjeng:

- Observations:
  - Cache Size, 32 → 64, CPI ↓
  - L1 & L2 Cache Size and Associativity ↑, CPI ↓ → As CPI ↓ , Performance ↑ .
- Optimal Configuration i.e., Lowest CPI:
  - 456.hmmer:

    L1.d = 128, L1.i = 128, L2 = 2048, L1 Associativity = 4, L2 Associativity = 2, Cache Size = 64, the CPI is 1.07033.
  - 458.sjeng:

    L1.d = 128, L1.i = 128, L2 = 2048, L1 Associativity = 4, L2 Associativity = 2, Cache Size = 64, the CPI is 7.26059.

PART 4: Define Cost Function

- The Cost Function is directly proportional to L1i, L1d, L2 size, L1a, and L2a while it is indirectly proportional to the overall cache size (CS)
- L1 Caches are more costly than L2 Caches.
- Thus, the cost function is defined as below:

  CF = $(L1.i + L1.d) * 0.02 + L2 * 0.01 + L1.a + L2.a * 0.5 + (20/CS)$

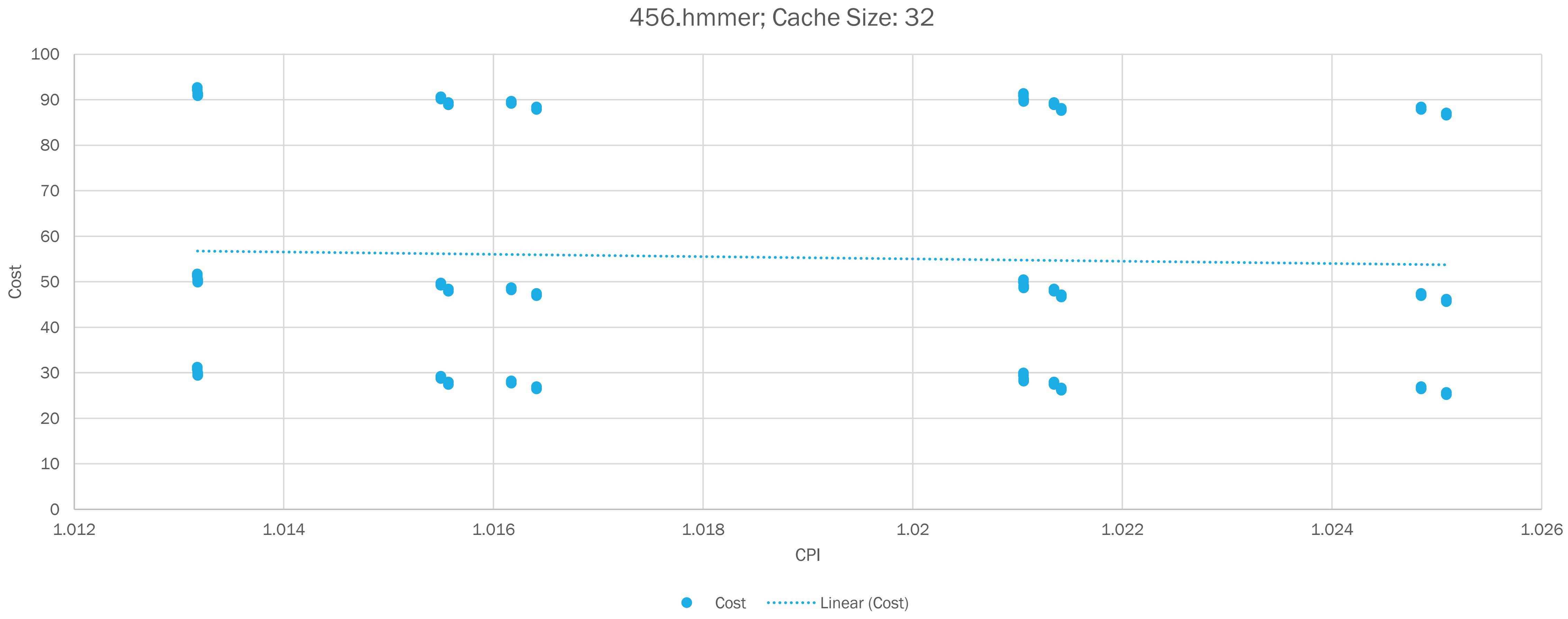
- Optimal Configuration i.e., Lowest Cost:

  L1.d = 64, L1.i = 64, L2 = 2048, L1 Associativity = 1, L2 Associativity = 1, Cache Size = 64.

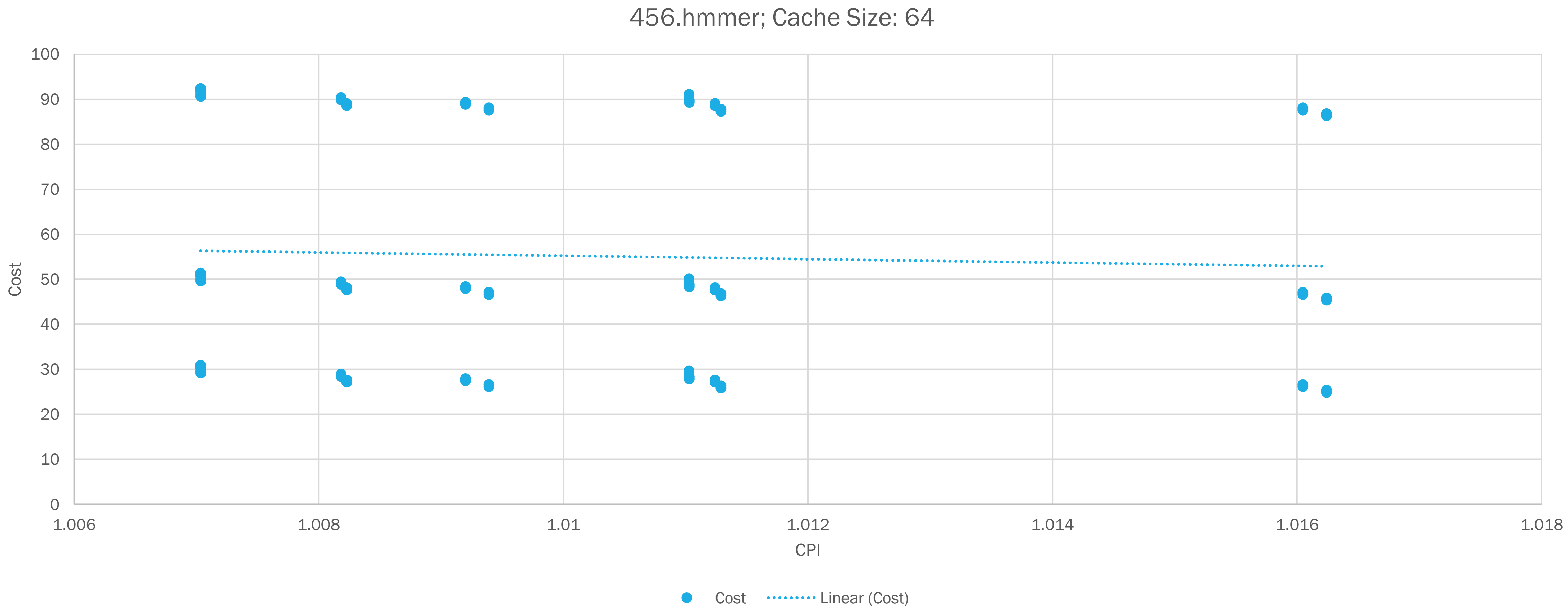
  the **Cost** is **24.3525.**

PART 5: Evaluation Function Showing Trade Offs (CPI Vs Cache Parameter Trade Off Curves And Analysis/Discussion)

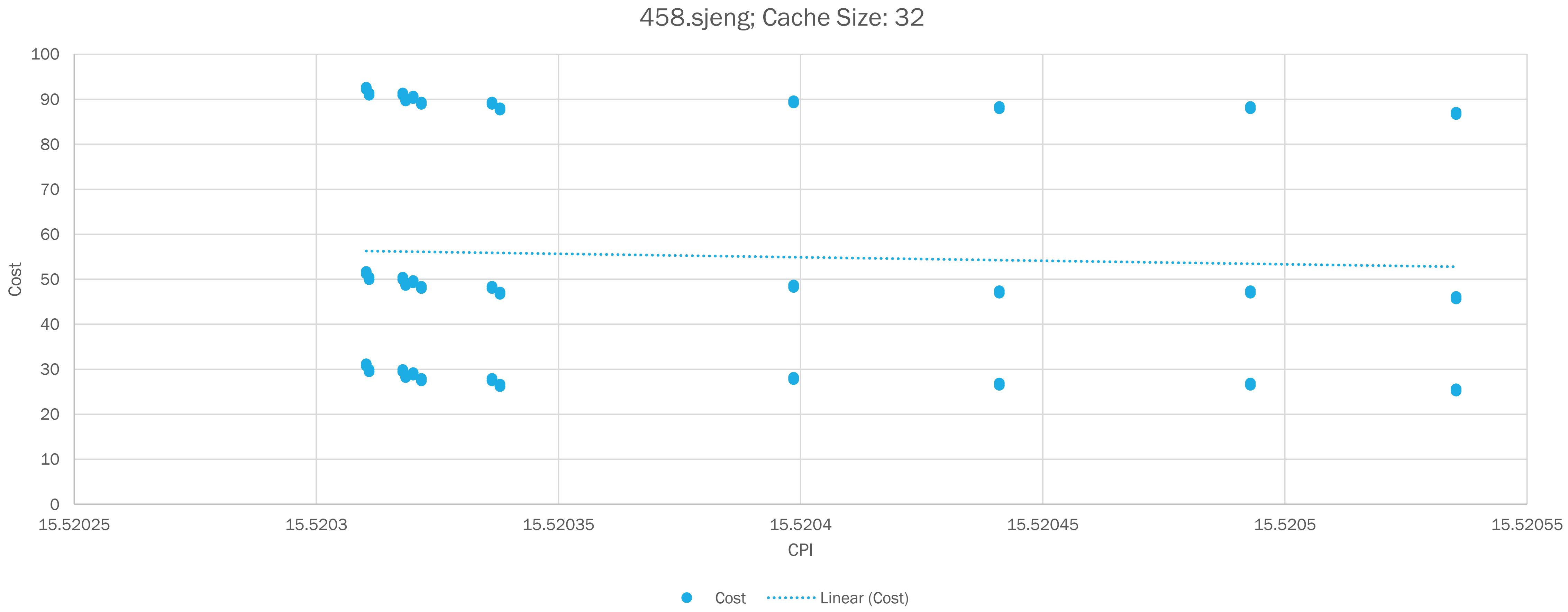
For 456.hmmer:



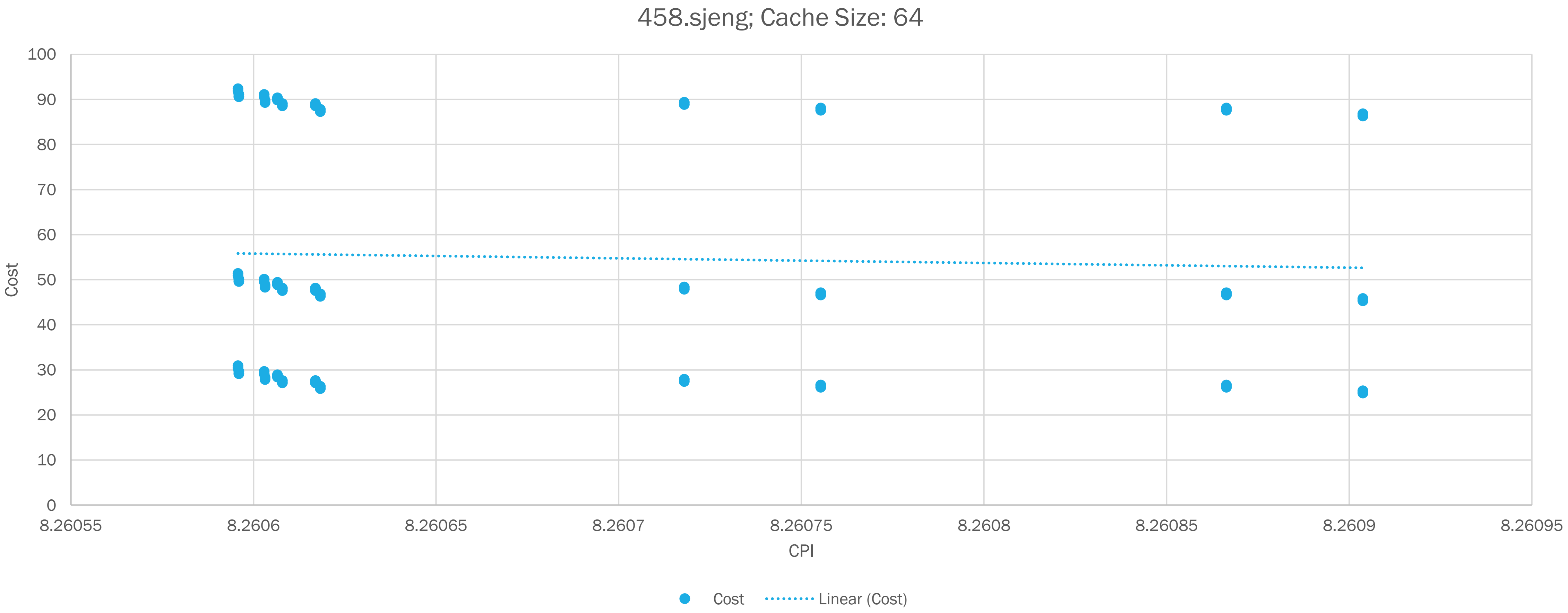
For 456.hmmer:



For 458.sjeng:



For 458.sjeng:



From the Graphs, for 456.hmmer and 458.sjeng:

- Observations:
  - CPI $\propto \frac{1}{\text{Cost Function}}$; Highest CPI will have Lowest Cost.
  - By Increasing Cache Block Size, we get Good Performance with Minimum Cost.
- Optimal Configuration, Low CPI with Low Cost:
  - 456.hmmer:

    L1.d = 64, L1.i = 64, L2 = 2048, L1 Associativity = 2, L2 Associativity = 1, Cache Size = 64,

    the **CPI** is **1.01128888** and **Cost Function** is **24.8525.**
  - 458.sjeng:

    L1.d = 64, L1.i = 64, L2 = 2048, L1 Associativity = 2, L2 Associativity = 1, Cache Size = 64,

    the **CPI** is **8.26061832** and **Cost Function** is **24.8525.**

    .

Appendix

Automation Steps:

- Defined cache parameter ranges.
- Executed "runGem5.sh" in benchmark folders, specifying output locations.
- Ran scripts iteratively using "sh runGem5.sh" for each cache size.
- Created two scripts per benchmark to expedite the process.
- Executed scripts in parallel, completing all iterations simultaneously (each script takes approximately 12 hours).
- Generated 144 combinations for each benchmark.
- Extracted essential data from outputs and inputted values into an Excel sheet.
- Calculated CPI for each combination.

THANK YOU!