# High Performance Computer Architecture (CS60003)

***Assignment - 1***

**Group - 1**

Indian Institute of Technology
Kharagpur

## GROUP MEMBERS & THEIR CONTRIBUTIONS

| Name | Roll Number | Contribution |
|---|---|---|
| Utkarsh Patel | 18EC35034 | Analyzing the results and writing code for plotting. |
| Aaditya Agrawal | 19CS10003 | Writing the code to plot the results (analysis.py), analyzing the top 10 configurations and stats.txt and preparing the report. |
| Akshat Vijay | 19CS10008 | Analyzing the statistics. |
| Anindya Sikdar | 19CS10010 | Writing the custom configuration code(config.py), automating the execution of all parameter combinations(run.py) and preparing the report. |
| Chandra Pavan Sai Chowdary | 19CS10023 | Analyzing the top 10 configurations and benchmark program, and preparing the report. |
| Gandhi Abhishek Rajesh | 19CS10031 | Modifying options.py and analyzing the results. |
| Gawai Laukik Thageshwar | 19CS10032 | Analyzing the top 10 configurations and preparing the report. |
| Kotaru Parasurama Sai Mani Surya | 19CS10041 | Automating the execution of parameter combinations and preparing the report. |

**Note:** Each member simulated 32 configuration combinations in their system as each run of configuration was computationally expensive and took a lot of time.

## OBJECTIVE

The gem5 simulator is an open-source system-level and processor simulator supporting multiple ISAs like ARM, MIPS and RISC-V. In this assignment, we configured an out of order CPU with a list of different microarchitectural parameters combinations and ran the benchmarked program on it. The different parameter combinations are automated through a script and the results are sorted based on the CPI. Certain performance parameters are extracted from the generated stats file of top 10 configurations and are plotted to observe their variations.

## CODE SUBMISSION (STRUCTURE)

```
HPCA_GRP_01_ASSGN
├── analysis.py
├── blocked-matmul
├── blocked-matmul.c
├── config.py
├── m5out
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_32kB_l1i_size_8kB_l2_size_512kB_bp_type_BiModeBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_32kB_l1i_size_8kB_l2_size_512kB_bp_type_TournamentBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_16kB_l2_size_256kB_bp_type_BiModeBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_16kB_l2_size_256kB_bp_type_TournamentBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_16kB_l2_size_512kB_bp_type_BiModeBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_16kB_l2_size_512kB_bp_type_TournamentBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_8kB_l2_size_256kB_bp_type_BiModeBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_8kB_l2_size_256kB_bp_type_TournamentBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   ├── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_8kB_l2_size_512kB_bp_type_BiModeBP_ROBEntries_192_numIQEntries_64
│   │   └── stats.txt
│   └── output_LQEntries_64_SQEntries_64_l1d_size_64kB_l1i_size_8kB_l2_size_512kB_bp_type_TournamentBP_ROBEntries_192_numIQEntries_64
│       └── stats.txt
├── options.py
├── Report.pdf
├── requirements.txt
└── run.py
```

## STEPS TO RUN

Follow the following steps to run the program:

1. Copy the HPCA_GRP_01_ASSGN inside the ~/gem5/configs folder.
2. Open the terminal inside the HPCA_GRP_01_ASSGN folder and run `**python3 run.py**`
3. The output can be found inside the ~/gem5/configs/HPCA_GRP_01_ASSGN/outputs folder.
4. Now install seaborn, which is a library used for plotting, (this can be done inside a virtual environment or directly) by executing '**pip install**

**seaborn**'.
5. To generate all the statistics and analyse the outputs, execute '**python3 analysis.py**'. This generates a directory **m5out**/, which will contain stats for best 10 configuration, **best_10_config.csv** that contains parameter combinations of best 10 configurations and **plots**/ where all the plots will be present.

## ANALYSIS

**Total number of configuration combinations:** $2^8 = 256$
The configurations were generated using the following variable parameters.

- **LQEntries**: 32, 64
- **SQEntries**: 32, 64
- **l1d size**: 32kB, 64kB
- **l1i size**: 8kB, 16kB
- **l2 size**: 256kB, 512kB
- **bp type**: TournamentBP, BiModeBP
- **ROBEntries**: 128, 192
- **numIQEntries**: 16, 64

**Benchmark Program:** blocked-matmul.c

The top 10 configuration combinations based with respect to CPI is given in the table below:

| LQEntries | SQEntries | l1d_size | l1i_size | l2_size | bp_type | ROBEntries | numIQEntries | cpi |
|---|---|---|---|---|---|---|---|---|
| 64 | 64 | 64kB | 16kB | 512kB | BiModeBP | 192 | 64 | **0.443066** |
| 64 | 64 | 64kB | 16kB | 512kB | TournamentBP | 192 | 64 | **0.443066** |
| 64 | 64 | 64kB | 8kB | 512kB | TournamentBP | 192 | 64 | **0.443139** |
| 64 | 64 | 64kB | 8kB | 512kB | BiModeBP | 192 | 64 | **0.443139** |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 64 | 64 | 64kB | 16kB | 256kB | TournamentBP | 192 | 64 | **0.443178** |
| 64 | 64 | 64kB | 16kB | 256kB | BiModeBP | 192 | 64 | **0.443178** |
| 64 | 64 | 64kB | 8kB | 256kB | BiModeBP | 192 | 64 | **0.44324** |
| 64 | 64 | 64kB | 8kB | 256kB | TournamentBP | 192 | 64 | **0.44324** |
| 64 | 64 | 32kB | 8kB | 512kB | TournamentBP | 192 | 64 | **0.444871** |
| 64 | 64 | 32kB | 8kB | 512kB | BiModeBP | 192 | 64 | **0.444871** |

**Analysis of best configurations:**

The above configurations work best for the given benchmark program for the following reasons:

1. The program divides the matrices into 2X2 submatrices and performs block wise multiplication. Since it needs to access blocks of contiguous memory multiple times cache size plays a crucial role in the performance as evident in the above configuration combinations, CPI is reducing gradually with the increase of the size of both the L1 and L2 caches.
2. The TournamentBP and the BiModeBP appear an equal number of times in the top 10 configurations. The BiModeBP is a better choice for our benchmark program since the for loops used are mostly simple loops with the pattern T*NT which can be effectively predicted with a simple BiModeBP.
3. A larger ROB size helps the CPU to issue and execute more instructions in-flight increasing performance. The same can be observed in our case as well.
4. Since our program executes a lot of load and store instructions a larger load queue and store queue size help boost the performance as evident in the above configurations.
5. A larger Instruction Queue size lets the CPU prefetch more instructions. Therefore the CPU never has to wait for the IQ to get filled to issue the next instruction. This can be seen in our case of the Benchmark program as well.

## PLOTS

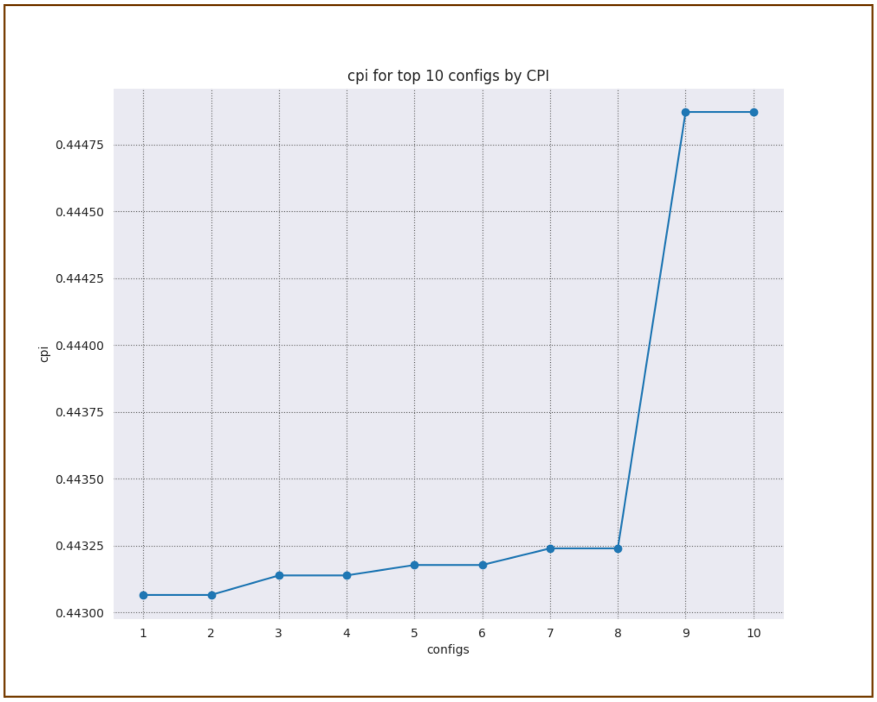


**Figure 1:** CPI for top 10 configurations

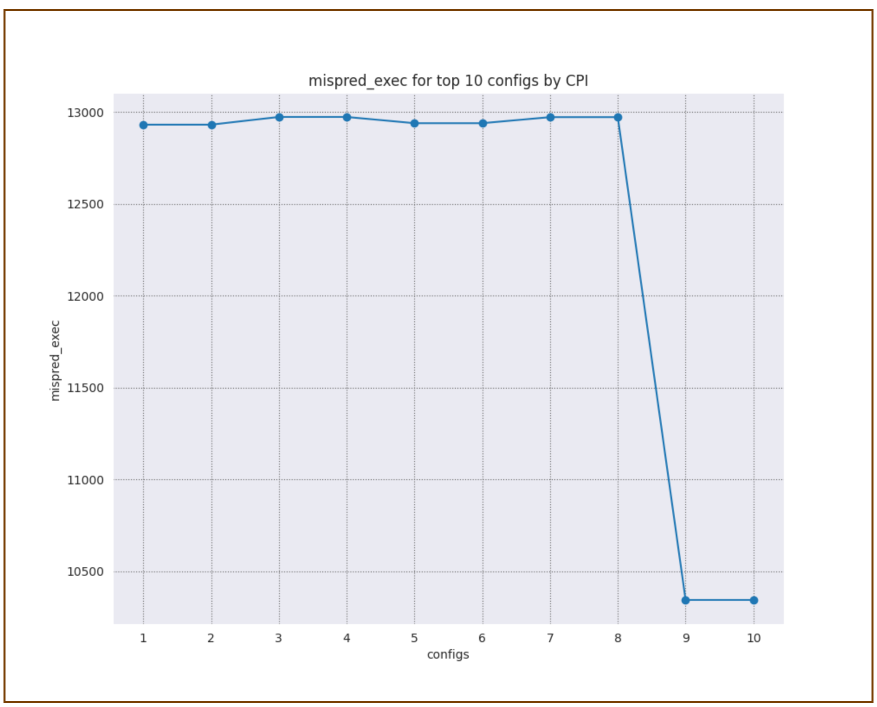


**Figure 2:** Mispredicted branches detected during execution

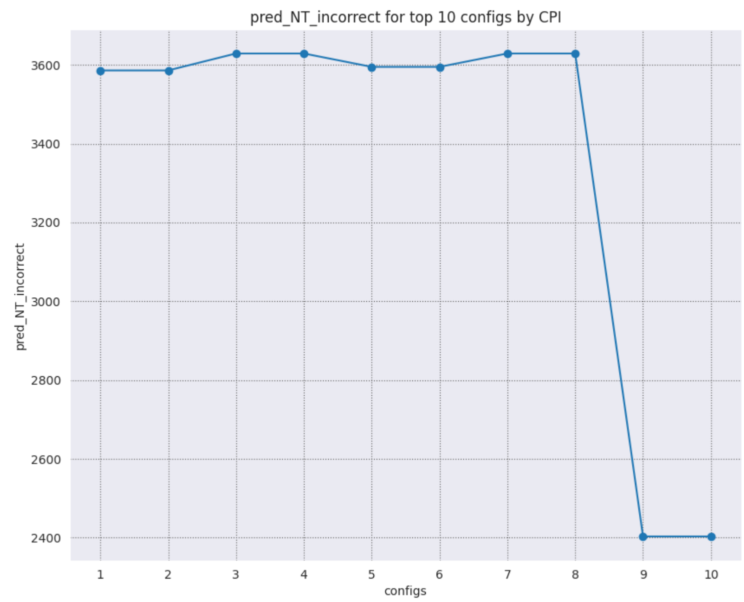


**Figure 3:** Number of branches that were predicted not taken incorrectly

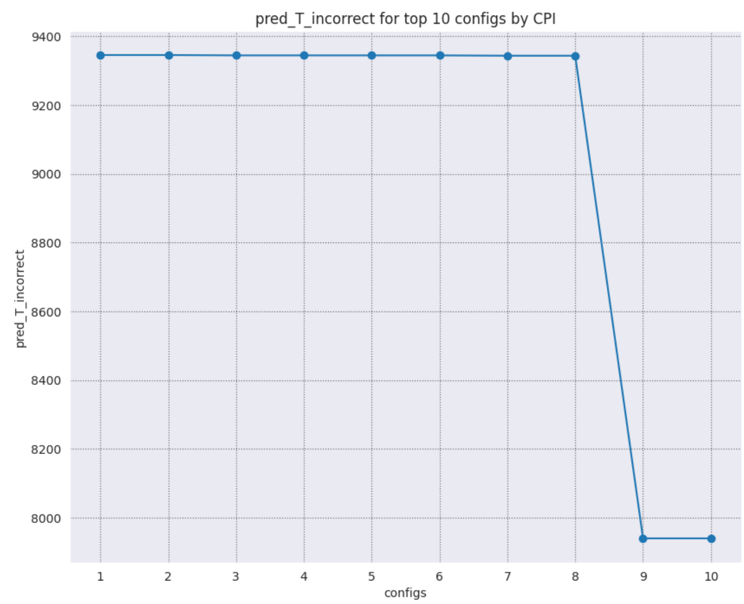


**Figure 4:** Number of branches that were predicted taken incorrectly

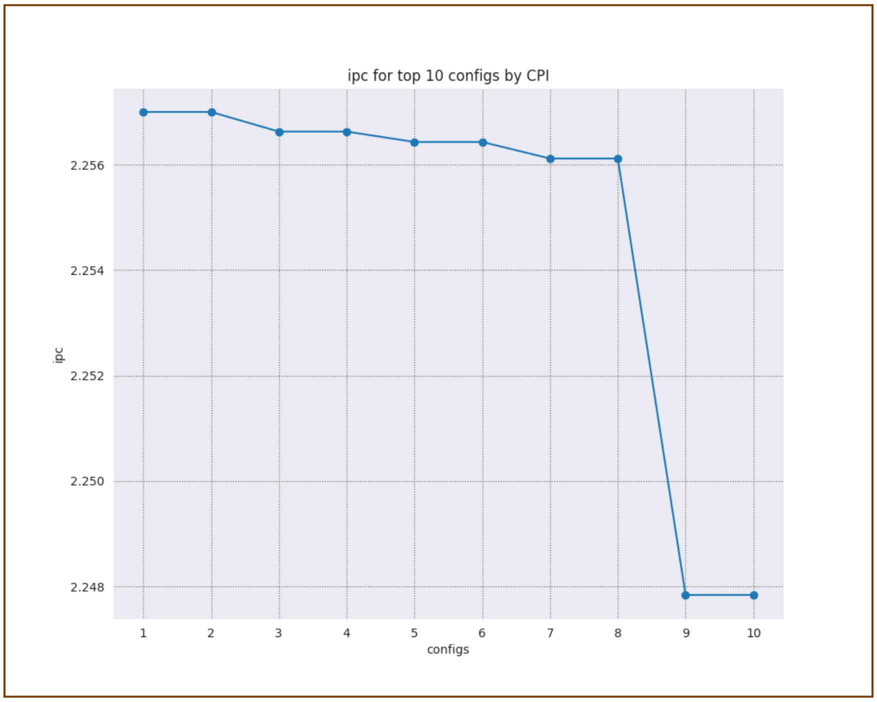


**Figure 5:** Instructions Per Cycle (IPC)

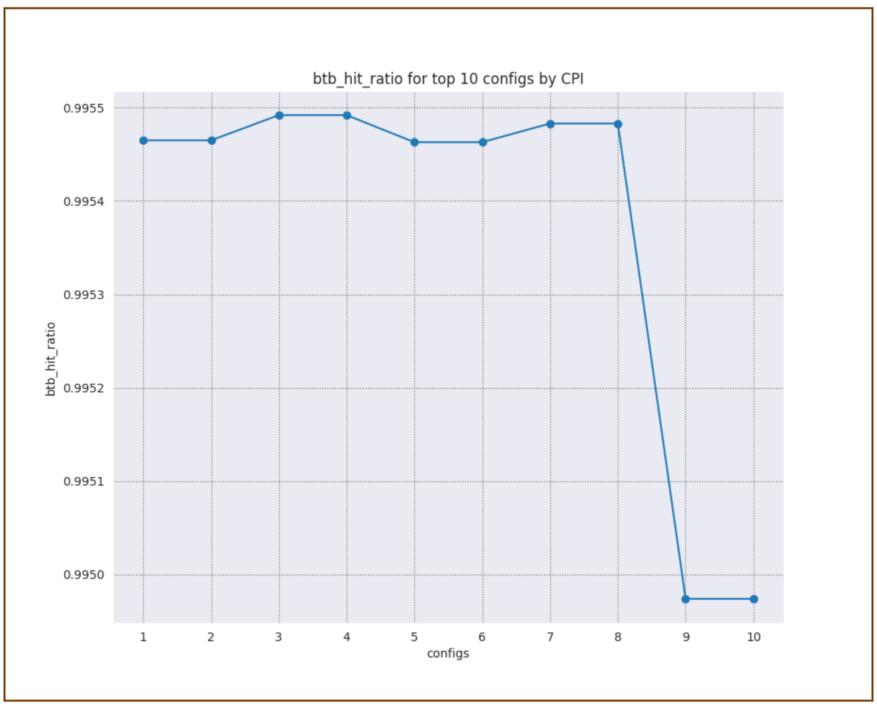


**Figure 6:** Number of BTB hit percentage

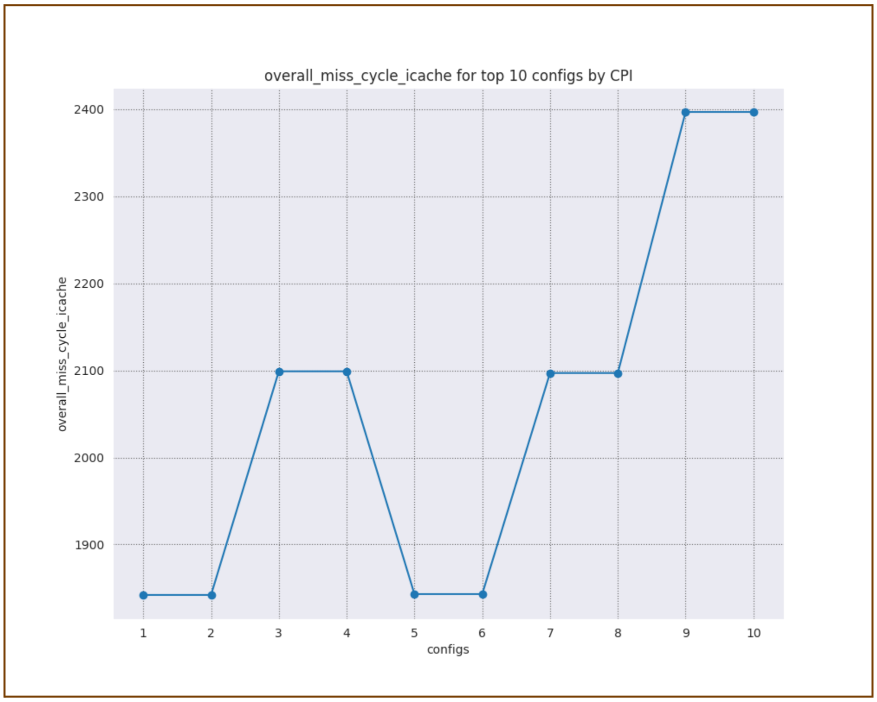


**Figure 7.a:** Number of overall miss cycles - icache

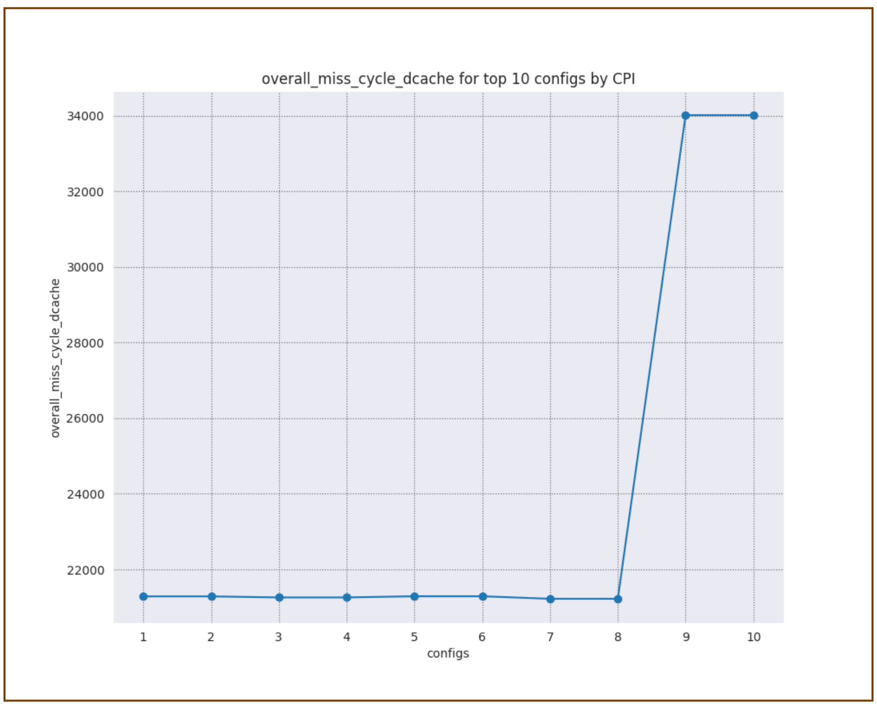


**Figure 7.b:** Number of overall miss cycles - dcache

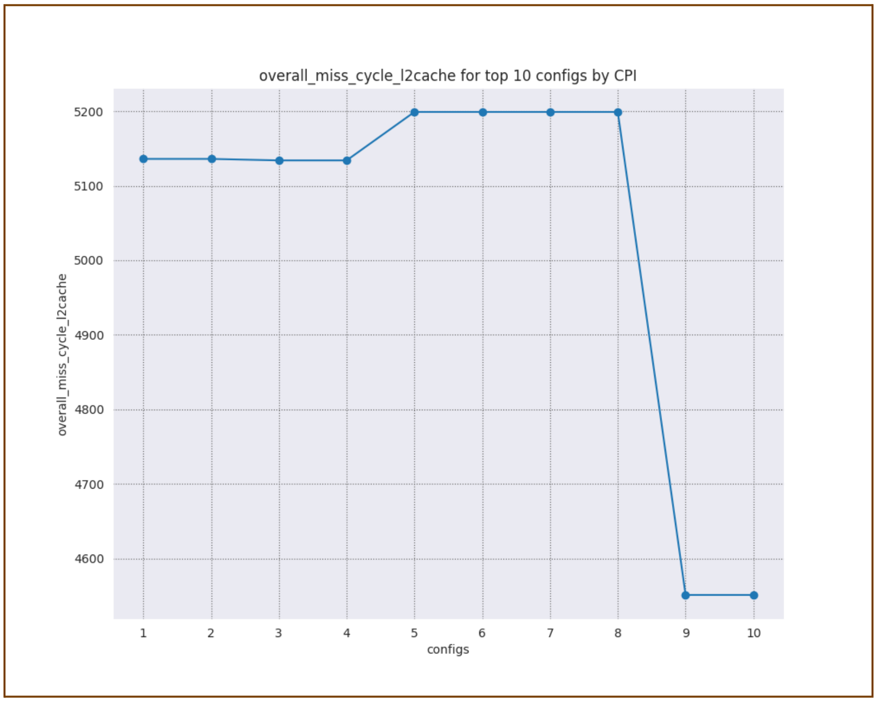


**Figure 7.c:** Number of overall miss cycles - l2cache

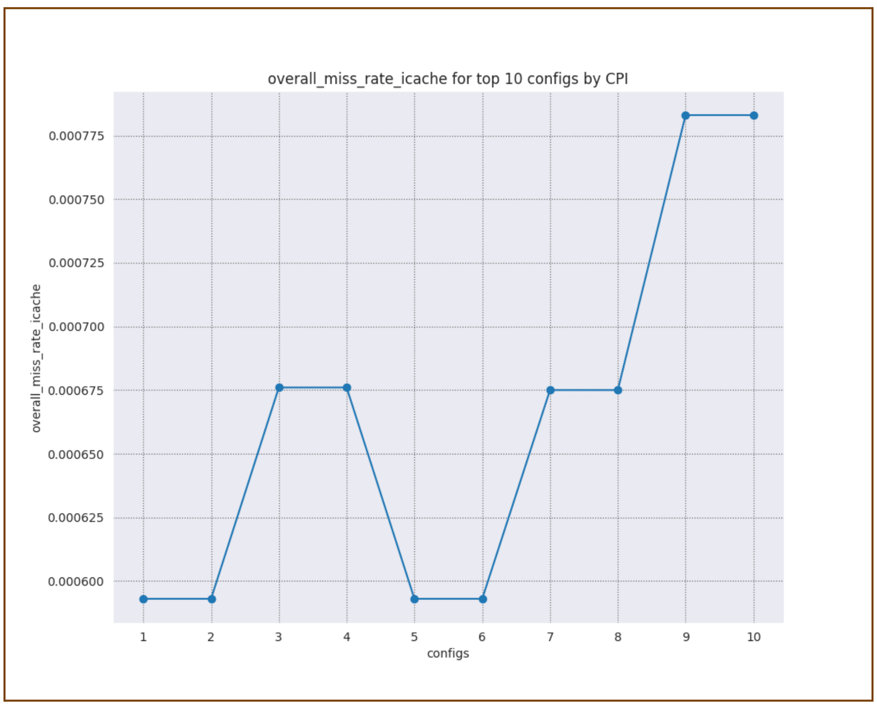


**Figure 8.a:** Overall miss rate - icache

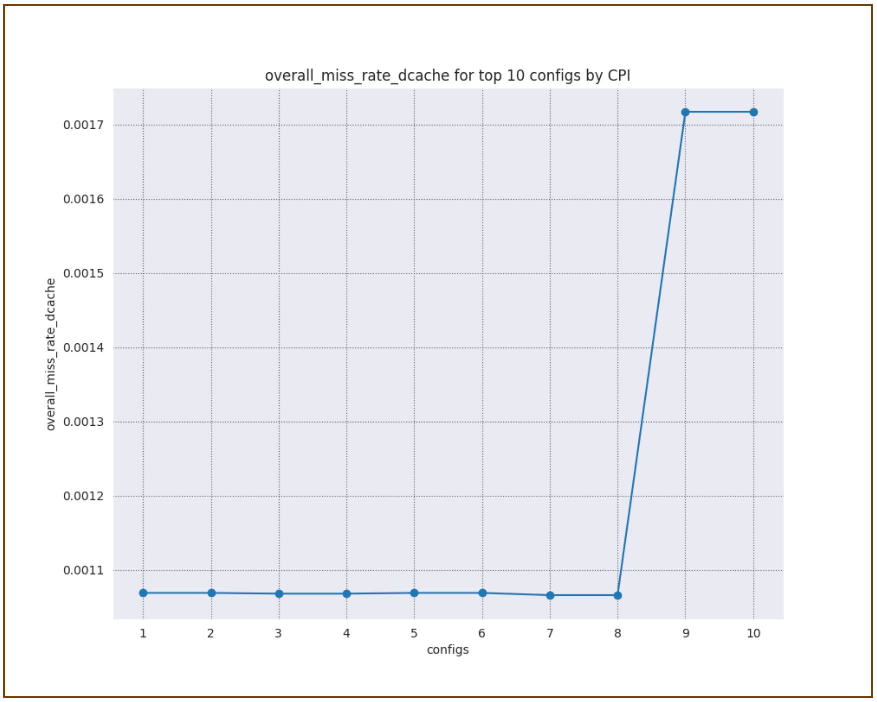


**Figure 8.b:** Overall miss rate - dcache

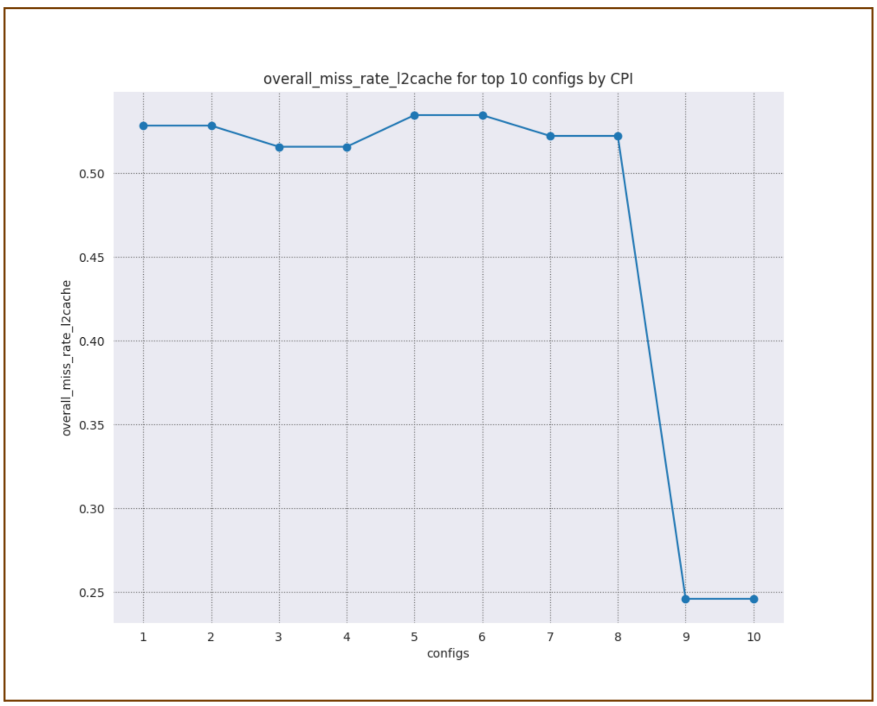


**Figure 8.c:** Overall miss rate - l2cache

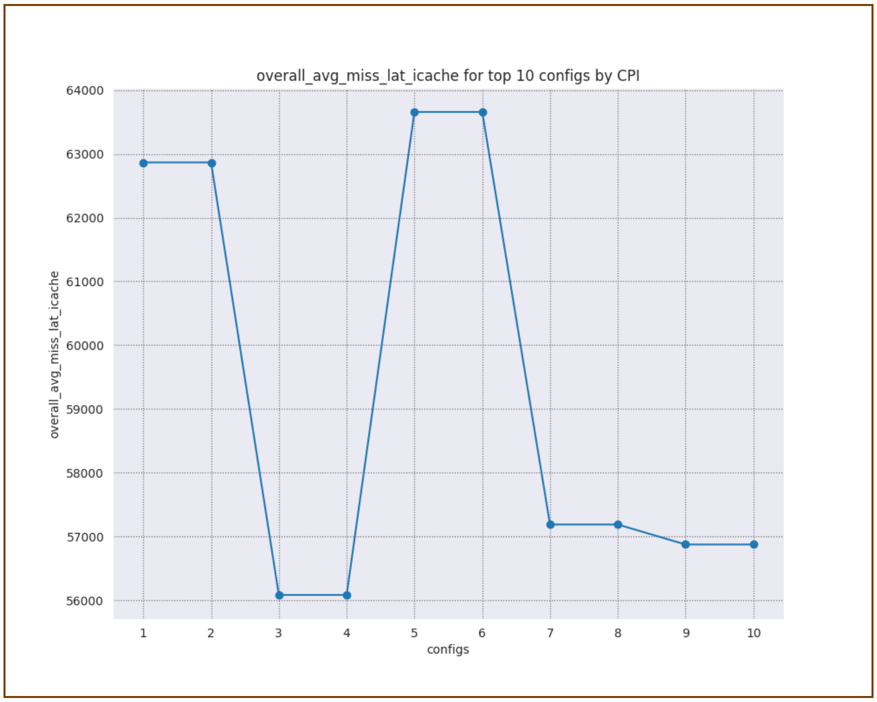


**Figure 9.a:** Overall average miss latency - icache

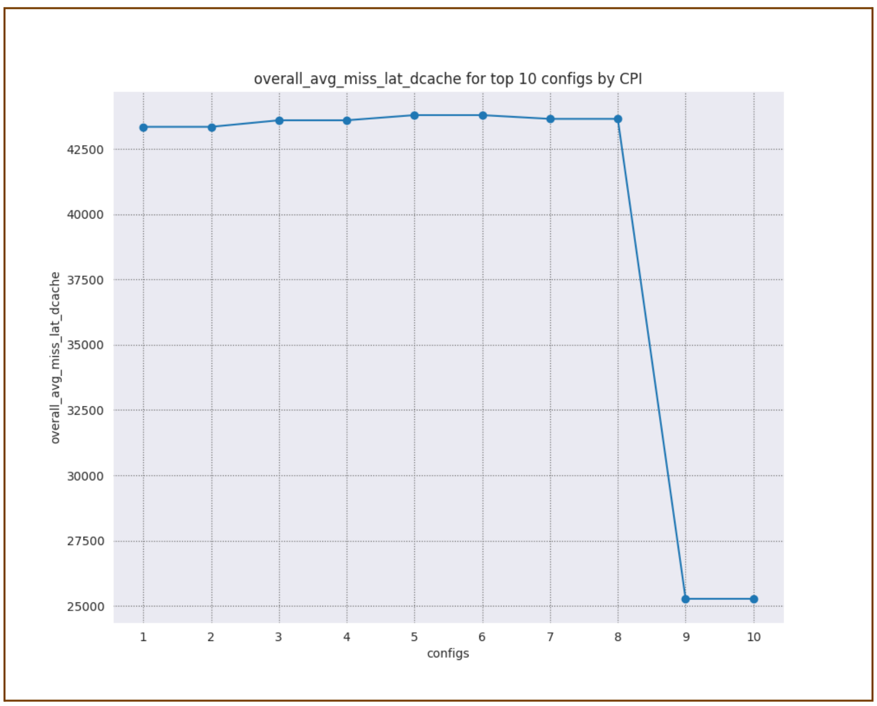


**Figure 9.b:** Overall average miss latency - dcache

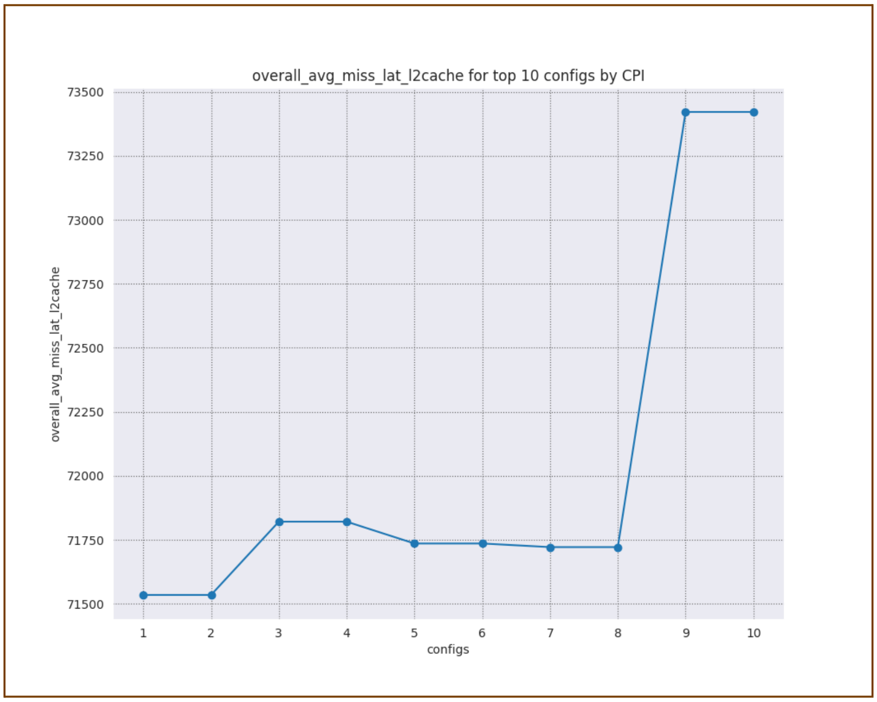


**Figure 9.c:** Overall average miss latency - l2cache

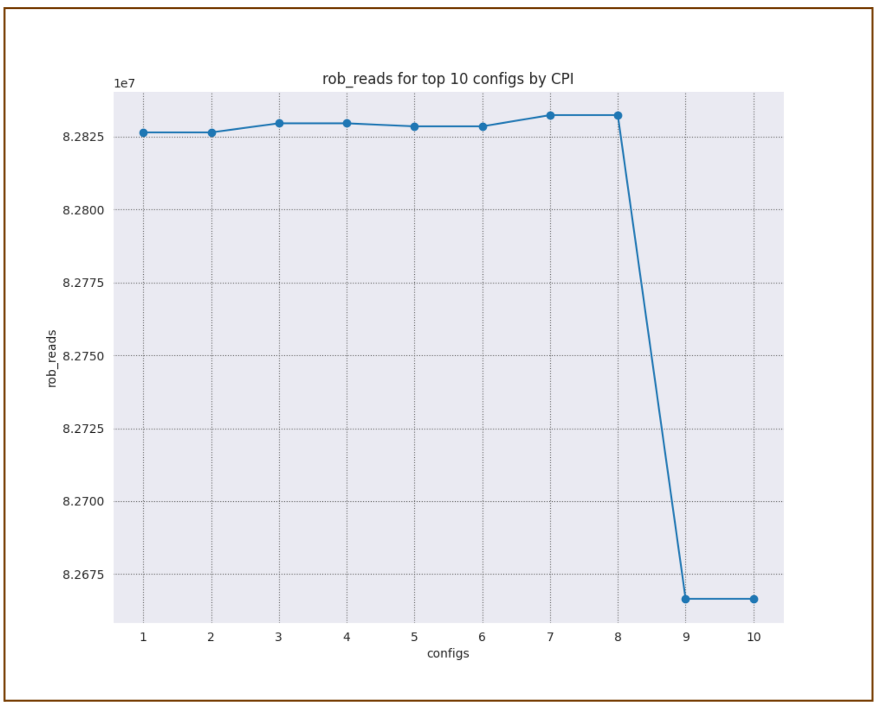


**Figure 10.a:** The number of ROB accesses - read

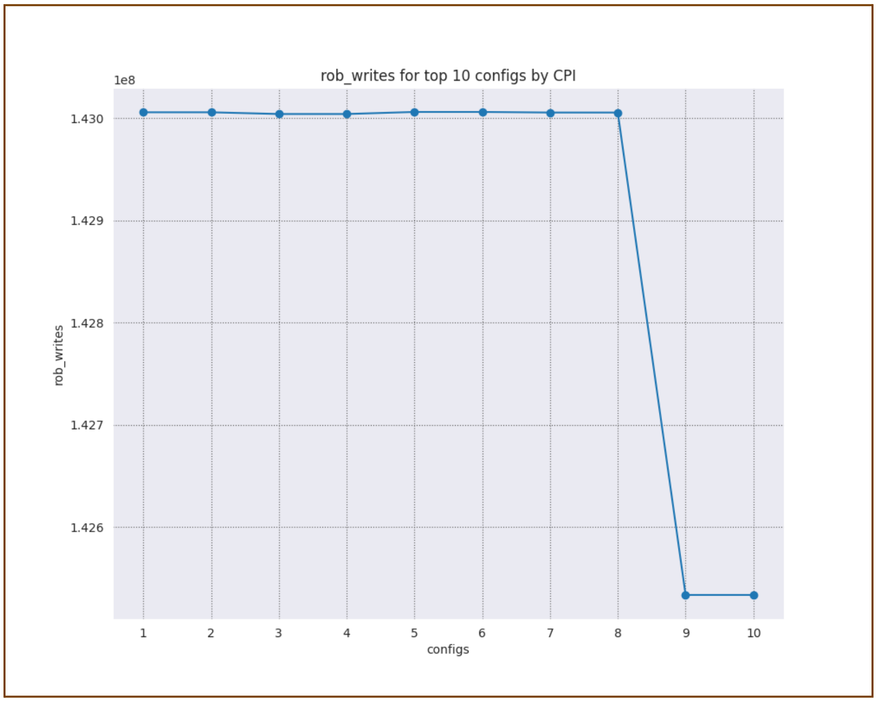


**Figure 10.b:** The number of ROB accesses - write

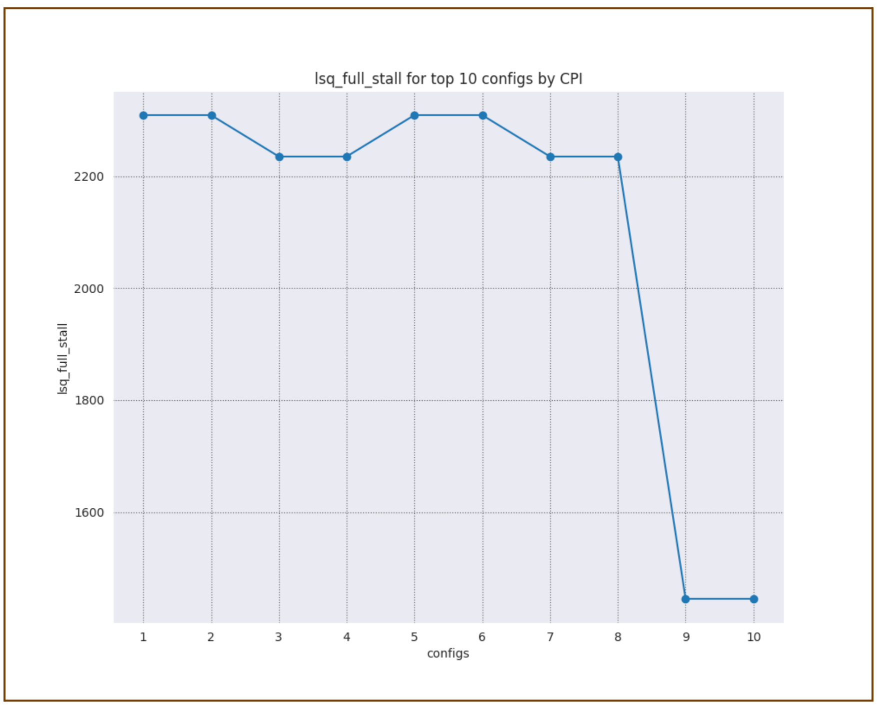


**Figure 11:** Number of times the LSQ has become full, causing a stall

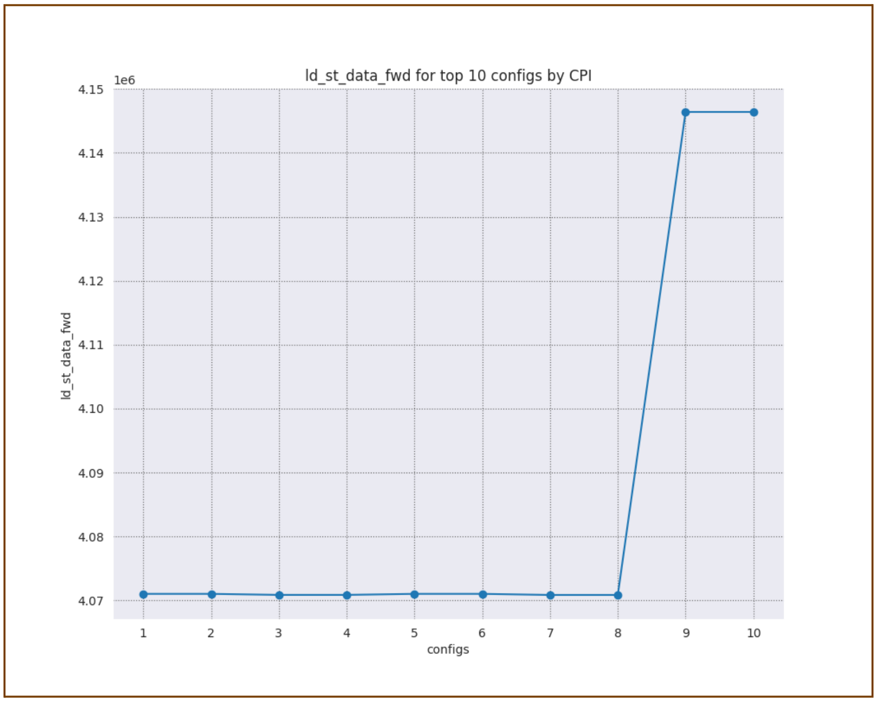


**Figure 12:** Number of loads that had data forwarded from stores

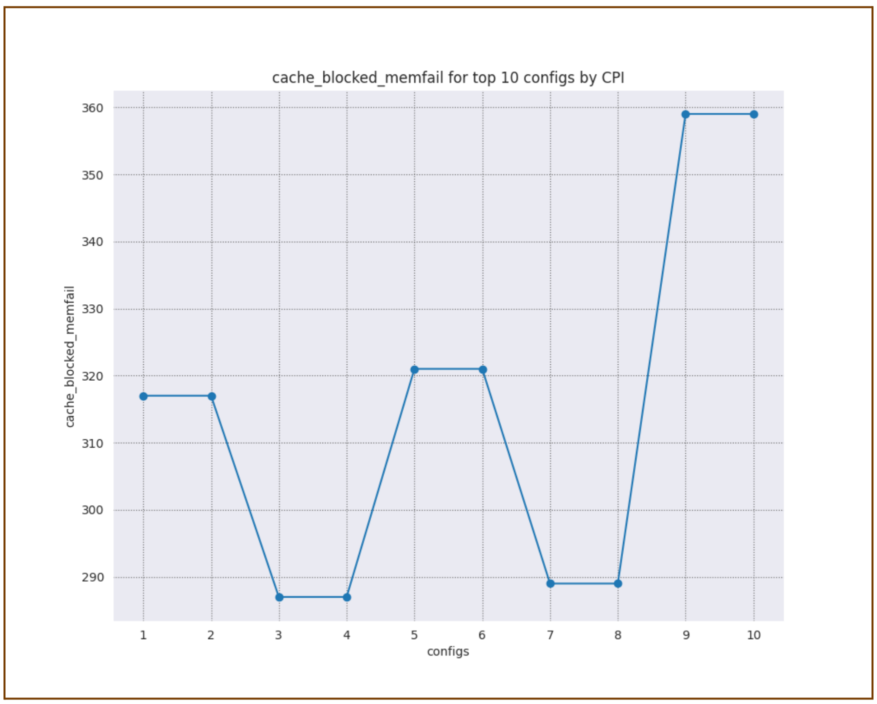


**Figure 13:** Number of times access to memory failed due to the cache being blocked