# Multiple Regression model for CPI stack

Anishka Ratnawat
*Computer Science and Automation*
*IISc, Bangalore, India*
*anishkar@iisc.ac.in*

Snigdha Shekhar
*Computer Science and Automation*
*IISc, Bangalore, India*
*snigdhas@iisc.ac.in*

Tanya Gautam
*Computer Science and Automation*
*IISc, Bangalore, India*
*tanyagautam@iisc.ac.in*

***Abstract*—CPI can be broken down into various components that comprise of base CPI and various other events like branch-misses, cache-misses, TLB-misses, etc. This breakdown of CPI into its components is called CPI stack, and provides useful information that software developers may utilise to optimize their code. In this assignment, we have used four benchmark programs under SPEC 2017 and recorded the counts of various hardware Performance Monitoring Counters(PMC) obtained using the *perf* utility in Linux. We have also illustrated the results from our linear regression model for predicting the CPI values for the test sets and finally report the CPI stack for all of the assigned SPEC benchmark programs.**

## I. Introduction

Processors today (including those made by ARM, AMD and Intel) provide the functionality to access a wide range of hardware performance counters for measuring and tracking events that occur during the execution of a process, such as counts related to instructions (such as the number of instructions or cycles used in the execution of the instruction), accesses to memory and associated hit/miss events (cache hits or misses, DRAM hits or misses), and typical behaviour of execution of the program on the pipelined CPU, amongst other events. The Performance Monitoring Unit (PMU) is the functional unit that provides access to HPC on Intel processors. [1]

The events associated with these HPCs can be grouped into types, namely, architectural events and micro-architechtural events. While the architectural events supported by these counters are not processor-specific, the micro-architectural events can be tailored to a given processor architecture. Examples of architectural events include *instructions, cycles* or *branches*. Micro-architectural events consist of processor-specific events like cache accesses, TLB (translation lookaside buffer) accesses, or branch prediction. The micro-architectural events can also vary with processor generation.

These events majorly contribute to the CPI (Cycles per Instruction) of a program. For a pipelined processor, the CPI is the sum of the base CPI and CPI components contributed by major stalls caused by these events.

*Pipeline CPI = Ideal pipeline CPI + Structural stalls + Data hazard stalls + Control stalls*

Here, the base CPI or ideal CPI is a measure of the maximum performance attainable by the implementation [2]. The various CPI components indicate the cycles lost to servicing various miss events such as those spent in servicing cache misses/TLB misses or re-fetching the instructions following the execution of the branch address in case of a misprediction. Hence, the CPI can be broken down into a base CPI and several CPI components in the form of a stacked bar with the base CPI at the bottom of the stack. This stack provides a lot of valuable insight into the execution behaviour of the program on a processor and helps in identifying *regions of interest* in the program.

For the evolution and development of system micro architectures and compiler design, benchmarking is a pivotal step. A benchmark suite of applications can be executed on an architecture to identify the performance bottlenecks and evaluate potential optimization techniques that can be used to enhance the performance of the system. For our assignment, we were provided with several benchmark programs to run and evaluate using the performance counters available on our system and report the final CPI stack for these programs. These benchmark programs are a part of the Standard Performance Evaluation Corporation (SPEC) CPU benchmark suite which is used in advancing computer architecture research.

In order to determine the CPI stack, we make use of a multiple regression model that is used to determine the relation between several independent variables to an output or response variable by fitting the observed data onto a line.

## II. Experiments

### A. Tools and software

1) *perf*

The perf tool from Linux is a command-line utility that helps in CPU profiling and performance monitoring in Linux systems. It gives a deep insight into analyzing CPU event data.

**Sample command from perf**

```
perf stat -I 100 -e cycles:u,instructions:u,
branch-misses:u,cache-misses:u
```

2) Processor Configuration
The system on which we ran the *perf* utility had the following configuration.

| | |
|---|---|
| Generation | 11th Generation Intel® Core™ i5 |
| Cores | 6 |
| Base Frequency | 2.70 GHz |
| Cache | 12 MB |

TABLE I
Processor Configuration

3) Benchmarks

The benchmarks assigned to us were the following.

- **521.wrf_r**: Performs Weather Forecasting and simulates the January 2000 North American Blizzard
- **525.x264_r**: Performs Video Compression and compresses portions of Blender Open Movie Project's "Big Buck Bunny"
- **526.blender_r**: Performs 3D Rendering and Animation and simulates reduced version of the Weybec Crazy Glue shot 3 data set to image

- **531.deepsjeng_r**: Plays Chess variants employing alpha-beta tree search

4) Python modules *sklearn, pandas, numpy, matplotlib* and *seaborn*

### B. Data Collection

In order to obtain the dataset, we followed a standard procedure to run the benchmark programs and obtain the performance monitoring counters. We have illustrated the steps that were taken through the explanation of the command below that we have run on an assigned benchmark, namely, *521.wrf_r*.

```
taskset -c 0 sudo perf stat -C 0 -o out.txt
-I 100 -e cycles:u,instructions:u,
branch-misses:u, cache-misses:u,
L1-dcache-load-misses:u,
L1-icache-load-misses:u, LLC-load-misses:u,
LLC-store-misses:u, branch-load-misses:u,
dTLB-load-misses:u, dTLB-store-misses:u
./wrf_r_base.mytest-m64 namelist.input
```

- We set the sampling interval for *perf* at 100 milliseconds after experimenting with various interval sizes.
- We closed all applications while running the program and pinned the process to core 0 using the *taskset* utlity in Linux that is used to set retrieve the CPU affinity of the running process.
- We suitably picked various events associated per benchmark program by reading the processor manual to find out about the various counters available for our processor generation and also by analyzing various factors about the program, namely whether it incurs more frontend misses or backend misses or a mix of both. [3]
- We have not considered the interaction between the chosen events.
- We collected approximately 1000 or more samples for every benchmark. For *521.wrf_r*, we collected close to 5000 samples.

### C. Data Preprocessing

After data collection, we took several steps to ensure that the raw data (event counts generated from the *perf* utility) is processed in order to transform it appropriately before feeding it to the LinearRegression model. We applied the following transformations on the output text file.

- We converted the data from .txt format to .csv format.
- As part of data cleanup, we excluded first and last few rows of counter values for every benchmark to account for outliers. This was done to ensure that we do not include exceptionally high CPI values or skewed counter values which can typically occur at the beginning and at the end of the program. We also removed the events which recorded zero counts throughout the excecution of the program.
- In order to normalise the data, we divided the counter values of the interval by the number of instructions per interval so as to ensure that the units for every event match with the unit for CPI. Table II shows an example of normalised data.

  Instructions: 526301578
  CPI: 0.867150712
- We generated a correlation matrix between all the events in order to understand the interdependence of the variables amongst themselves and on the final CPI. We also dropped some features (events) so as to remove duplicates and insignificant features i.e. features that have less correlation with CPI. This is explored in detail in the next section.

| Event | Before normalising | After normalising |
|---|---|---|
| branch-misses:u | 2308222 | 0.004386 |
| cache-misses:u | 18105979 | 0.040797 |

TABLE II
NORMALISING THE DATA

### D. Feature Selection

We performed feature selection in order to reduce the number of input variables/events to only those that are independent, have non-zero values, and have an acceptable correlation with CPI. This was done in order to determine the dependence of CPI only on significant features while disregarding features that fall below this threshold. The complete list of the events that we took under consideration are the following.

```
branch-misses:u
cache-misses:u
L1-dcache-load-misses
L1-icache-load-misses
LLC-load-misses
LLC-store-misses
branch-load-misses
dTLB-load-misses
dTLB-store-misses:u
iTLB-load-misses:u
l2_rqsts.code_rd_miss:u
l2_rqsts.demand_data_rd_miss:u
l2_rqsts.all_demand_miss:u
dtlb_load_misses.walk_pending:u
itlb_misses.walk_pending:u
dtlb_store_misses.walk_pending:u
offcore_requests.l3_miss_demand_data_rd:u,
ocr.hwpf_l2_rfo.l3_miss:u
ocr.demand_data_rd.l3_miss:u
icache_64b.iftag_miss:u
l2_rqsts.swpf_miss:u
page-faults:u
mem-stores:u
frontend_retired.itlb_miss:u
```

The exhaustive explanation of the events can be found on the last page of the report.

*1) Correlation matrix:* A correlation matrix is a method which is used to statistically measure the relation between two variables in a data set where a correlation coefficient of 1 with the output variable is strong, 0 is neutral and -1 is a weak relationship. All the correlation matrices that we produced on the benchmarks resulted in values between -1 and 1.

*2) Feature dropping:* Figure 1 gives a correlation between all the events and CPI as well as their dependence on one another for the dataset 521.wrf_r. For all the datasets, we performed similar analysis and dropped the features/columns after considering two factors:
1. Whether they are negatively correlated with the CPI ($< 0$) or have low correlation ($\sim 0$).
2. Whether they resulted in a negative coefficient after fitting them to the linear regression model as CPI components must be additive in nature.

In the next section, we present the final coefficients obtained using LinearRegression() module in Python and the final CPI stack after finalising the features significant to a particular benchmark.

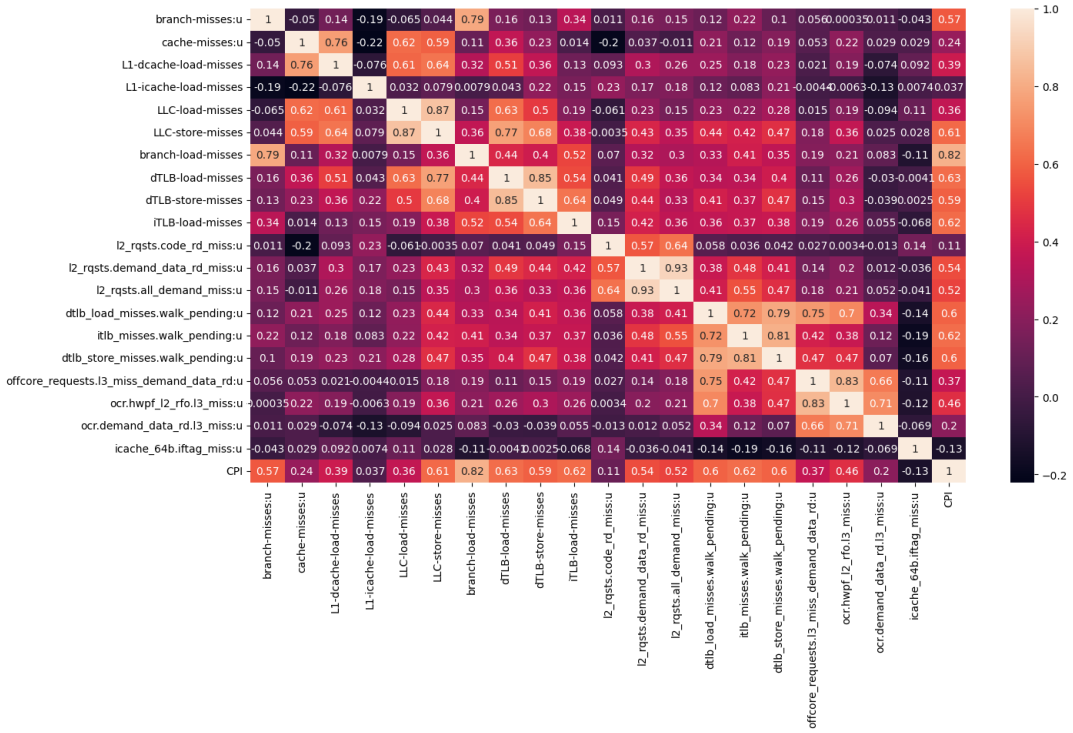


Fig. 1. Correlation matrix for 521.wrf_r

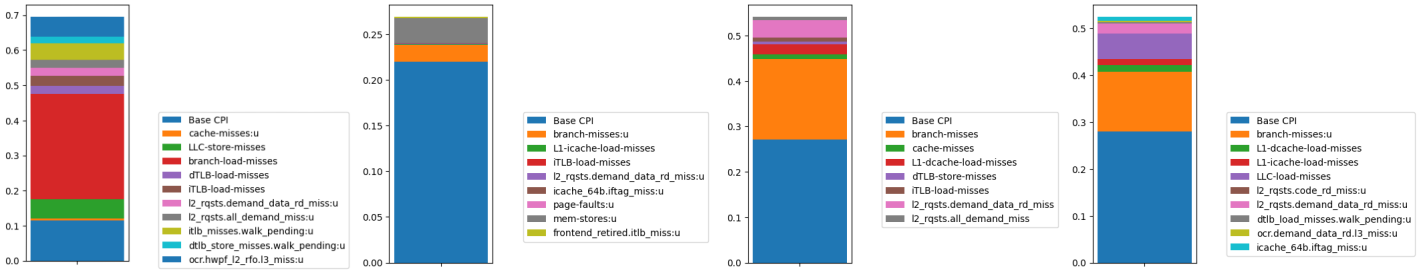


Fig. 2. CPI stack for 521.wrf_r, 525.x264_r,526.blender_r and 531.deepsjeng_r

| Events | Coefficients |
|---|---|
| **Base CPI** | 0.1155292879347648 |
| **cache-misses** | 2.3257585501568783 |
| **LLC-store-misses** | 173.45187095920957 |
| **branch-load-misses** | 216.31246068659655 |
| **TLB-load-misses** | 1467.2244979550828 |
| **iTLB-load-misses** | 11157.704106118174 |
| **l2_rqsts.demand_data_rd_miss** | 46.05350207280364 |
| **l2_rqsts.all_demand_miss** | 26.59748507518176 |
| **itlb_misses.walk_pending** | 188.00567760654022 |
| **dtlb_store_misses.walk_pending** | 15.43236022816734 |
| **ocr.hwpf_l2_rfo.l3_miss** | 95.19859286017424 |

TABLE III
EVENTS & THEIR RESPECTIVE COEFFICIENTS FOR 521.WRF_R

| Events | Coefficients |
|---|---|
| **Base CPI** | 0.21958288584656732 |
| **branch-misses** | 21.417395570619078 |
| **L1-icache-load-misses** | 0.3950663131367277 |
| **iTLB-load-misses** | 219.7146872368477 |
| **l2_rqsts.demand_data_rd_miss** | 25.61208694157674 |
| **icache_64b.iftag_miss** | 0.3556381594862765 |
| **page-faults** | 9045.555834210809 |
| **mem-stores** | 0.4754501001646133 |
| **frontend_retired.itlb_miss** | 893.9874014442757 |

TABLE IV
EVENTS & THEIR RESPECTIVE COEFFICIENTS FOR 525.X264_R

### *E. Experimental Results*

The equation for the CPI stack is of the form:

$$CPI = a_0 + a_1 miss\_event_1 + a_2 miss\_event_2 + ... + a_k miss\_event_k$$

where k is the number of miss events/features considered for each benchmark which may or may not be same for all. Here, $a_0$ represents the intercept or in other words the CPI base which is the CPI in absence of any miss events. The coefficients $a_i \forall i \in 1, 2, .., k$ of the various miss events representing the number of cycles spent in miss penalty.

We present the final CPI stack and the final additive terms and their coefficients obtained after building the regression model for every benchmark.

*1)* ***521.wrf_r****:* Based on the Fig 2 and Table III, we can observe that this benchmark has a significant CPI component due to branch stalls,i.e., a lot of cycles are spent in its pipeline due to branch misses. Other significant CPI components are LLC stores and iTLB misses that invoke a page walk. Therefore, there is a mix of both long latency *front-end misses* (branch mispredictions and iTLB) as well as *back-end misses* (LLC stores). This may indicate that this benchmark has a balance between compute-intensive and memory-intensive instructions with short irregular bursts of data at a high trans-

| Events | Coefficients |
|---|---|
| **Base CPI** | 0.27047848147962467 |
| **branch-misses** | 37.813429306942574 |
| **cache-misses** | 5.39227382013639 |
| **L1-dcache-load-misses** | 2.8441824012383705 |
| **dTLB-store-misses** | 353.844863790132 |
| **iTLB-load-misses** | 170.75028736429695 |
| **l2_rqsts.demand_data_rd_miss** | 29.102903403560045 |
| **l2_rqsts.all_demand_miss** | 2.6279204798520377 |

TABLE V
EVENTS & THEIR RESPECTIVE COEFFICIENTS FOR 526.BLENDER_R

| Events | Coefficients |
|---|---|
| **Base CPI** | 0.27942480836008743 |
| **branch-misses** | 25.46253146522196 |
| **L1-dcache-load-misses** | 8.80748891139935 |
| **L1-icache-load-misses** | 11.509268102173772 |
| **LLC-load-misses** | 200.98887905289612 |
| **l2_rqsts.code_rd_miss** | 28.48061944017447 |
| **l2_rqsts.demand_data_rd_miss** | 52.00258318656579 |
| **dtlb_load_misses.walk_pending** | 0.19920759291780143 |
| **ocr.demand_data_rd.l3_miss** | 9.604610401884429 |
| **icache_64b.iftag_miss** | 7.881887103671135 |

TABLE VI
EVENTS & THEIR RESPECTIVE COEFFICIENTS FOR 531.DEEPSJENG_R

fer rate. The high coefficients associated with these misses account for the high number of cycles spent in servicing these long latency miss events.

| | |
|---|---|
| RMSE | 0.1579929534137685 |
| $R^2$ | 0.8697 |
| Adj. $R^2$ | 0.8694678089435813 |
| F-statistic | 3475.3491660915547 |
| p-value | $1.1102230246251565 * e^{-16}$ |
| Avg. CPI | 0.695873779629864 |

TABLE VII
STATISTICAL VALUES FOR 521.WRF_R

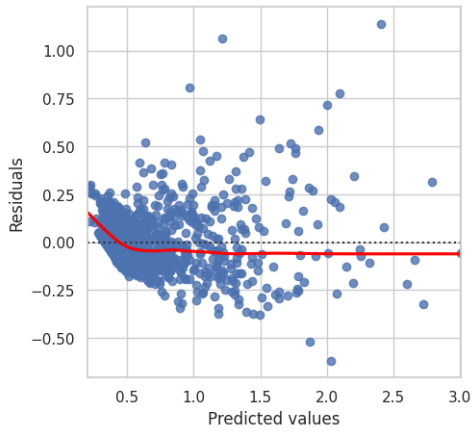


Fig. 3. Residual plot for 521.wrf_r

From the reported values, we can see that our model has a prediction accuracy of 86.97% with the *p-value* (obtained from the F-test) below significance level 0.05 indicating that it is statistically significant. Then we can reject the null hypothesis which states that the sample data occurs purely from chance. From the residual plot 3, we can also conclude that the linear model is a good but crude fit for our data, and a higher-order polynomial regression model could fit the data better(as indicated by the red line). We have also cross-verified that the individual CPI components and base CPI $a_o$ sum up to actual CPI, i.e.,

$$CPI = \sum a_i X_i + Base\ CPI \tag{1}$$

where $a_i X_i$ is the CPI component.

| | |
|---|---|
| Base CPI | 0.115529 |
| cache-misses:u | 0.005415 |
| LLC-store-misses | 0.054975 |
| branch-load-misses | 0.299545 |
| dTLB-load-misses | 0.022207 |
| iTLB-load-misses | 0.030061 |
| l2_rqsts.all_demand_miss:u | 0.022600 |
| l2_rqsts.demand_data_rd_miss:u | 0.022600 |
| itlb_misses.walk_pending:u | 0.046884 |
| dtlb_store_misses.walk_pending:u | 0.018569 |
| ocr.hwpf_l2_rfo.l3_miss:u | 0.056981 |
| **Predicted CPI** | 0.695427 |
| **True CPI** | 0.695873 |

TABLE VIII
CPI STACK FOR 521.WRF_R

*2) **525.x264_r**:* From Fig 2 we can observe that most of the CPI stack is made up of just the base CPI and all other components have nominal contributions. This could be due to the fact that the benchmark is CPU-bound. As the miss event components are taking less space in the stack, the 525.x264_r benchmark's overall performance is optimal, in other words, the *IPC* is highest in comparison to the other 3 benchmarks.

Though the benchmark is performing video compression that suggests a lot of memory reads and writes, that led us to expect significant load and store misses. As this is not the case, one can infer that the program exploits the data layout of the benchmark input well. From table IV, the highest coefficient is of page-faults indicating it has maximum miss penalty.

| | |
|---|---|
| RMSE | 0.0034582267941255583 |
| $R^2$ | 0.7950 |
| Adj. $R^2$ | 0.793975677307056 |
| F-statistic | 1553 |
| p-value | $1.1102230246251565 * e^{-16}$ |
| Avg. CPI | 0.2689105549705506 |

TABLE IX
STATISTICAL VALUES FOR 525.X264_R

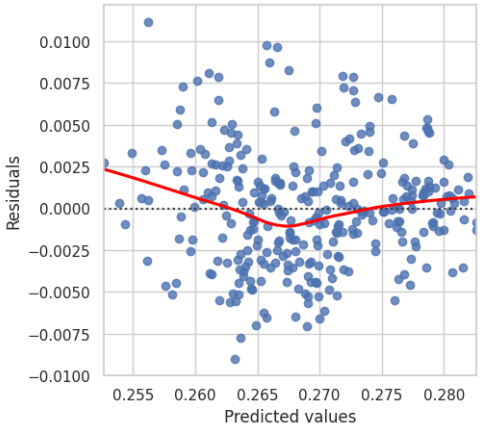


Fig. 4. Residual plot for 525.x264_r

The statistical data in the table IX suggests that the model is acceptable. The model predicts the CPI with 0.345%

error and $R^2$ suggesting that 79.5% of the data fits in the model correctly. With $F - statistic$ and $p - value$ and $\alpha$ (significance) $= 0.05$ we can safely reject the null hypothesis, i.e.,

$$H0 : a_0 = a_1 = ... = a_k = 0$$

In other words, the miss events have a significant contribution. Similarly, using equation 1, one can cross-check the CPI is the summation of CPI components.

| | |
|---|---|
| Base CPI | 0.219583 |
| branch-misses:u | 0.018199 |
| L1-icache-load-misses | 0.000616 |
| iTLB-load-misses | 0.000065 |
| l2_rqsts.demand_data_rd_miss:u | 0.001064 |
| icache_64b.iftag_miss:u | 0.000554 |
| page-faults:u | 0.000013 |
| mem-stores:u | 0.027617 |
| frontend_retired.itlb_miss:u | 0.001232 |
| **Predicted CPI** | 0.26894 |
| **True CPI** | 0.268910 |

TABLE X
CPI STACK FOR 525.X264_R

From figure 4, we can see that the residual plot shows that there is a good scatter of residuals *(y_test - y_pred )* vs *y_pred* showing that the residuals are spread equally along the ranges of our predictors. Hence, the model will not be unpredictable for high values. The residual plot is almost linear too, hence, linear regression is a good fit for this data.

*3)* ***526.blender_r****:* On analyzing the CPI stack obtained from this benchmark that is shown in Fig 2 and Table V, a significant part of the CPI (except for the base CPI) is taken up by branch-misses and l2_rqsts.demand_data_rd_miss, after that by L1-dcache-load-misses and rest by other miss events. This with a high base CPI indicates that this benchmark has more CPU-intensive and somewhat memory-bounded instructions which is to be expected as the benchmark is rendering a 3D model that needs more CPU cycles for computations. L1 with L2 data read/load miss events' portion in CPI stack suggests that it could be due to compulsory misses or its data layout not being optimal (but it is not entirely unsatisfactory), unlike the case of 525.x264_r benchmark where it was inferred (from its CPI stack) to be almost optimal.

| | |
|---|---|
| RMSE | 0.025202481936819978 |
| $R^2$ | 0.9322 |
| Adj. $R^2$ | 0.9317904662873451 |
| F-statistic | 2479.442586654288 |
| p-value | $1.1102230246251565 * e^{-16}$ |
| Avg. CPI | 0.5411213739866246 |

TABLE XI
STATISTICAL VALUES FOR 531.DEEPSJENG_R

From the reported values in table XI, we can see that our model has a prediction accuracy of 72.70% with the *p-value* (obtained from the F-test) below significance $\alpha = 0.05$ indicating that it is statistically significant. Then we can reject the null hypothesis which states that the sample data occurs purely from chance. Similarly, using equation 1, we can cross-check that the CPI is the summation of its CPI components.

From the residual plot 5, we can also conclude that the linear model is a good but crude fit for our data, and a higher-order polynomial regression model could fit the data better(as indicated by the red line).

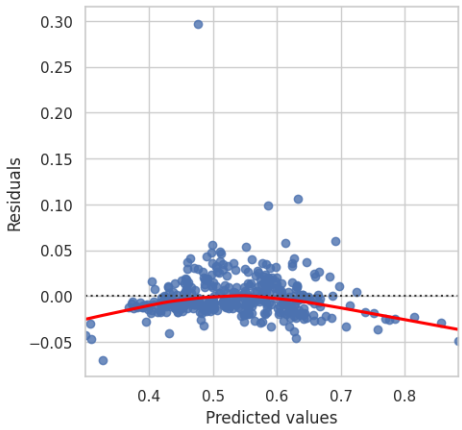


Fig. 5. Residual plot for 526.blender_r

| | |
|---|---|
| Base CPI | 0.270478 |
| branch-misses | 0.177636 |
| cache-misses | 0.010561 |
| L1-dcache-load-misses | 0.021212 |
| dTLB-store-misses | 0.007202 |
| iTLB-load-misses | 0.007735 |
| l2_rqsts.demand_data_rd_miss | 0.038154 |
| l2_rqsts.all_demand_miss | 0.008234 |
| **Predicted CPI** | 0.541211 |
| **True CPI** | 0.541121 |

TABLE XII
CPI STACK FOR 526.BLENDER_R

*4)* ***531.deepsjeng_r****:* From figure 2, we can observe that this benchmark has a significant CPI component due to branch-misses and LLC-load-misses. LLC-load-misses point out that the higher level caches saw a miss and finally, LLC has also reported a miss that eventually invokes page walk. In table VI also, the coefficient of LLC-load-miss is the highest confirming that it is the most time-consuming event. As the benchmark is doing alpha-beta tree searching, the tree nodes that are traversed might be generating the load-misses. But most part of the CPI stack is taken up by the Base CPI suggesting the benchmark is more CPU-bound.

| | |
|---|---|
| RMSE | 0.010438852796344497 |
| $R^2$ | 0.7270 |
| Adj. $R^2$ | 0.7259094929515474 |
| F-statistic | 638.6830648489755 |
| p-value | $1.1102230246251565 * e^{-16}$ |
| Avg. CPI | 0.5243999179723248 |

TABLE XIII
STATISTICAL VALUES FOR 526.BLENDER_R

The statistical data in the table XIII suggests that the model is acceptable. The model predicts the CPI with 2.520% error and $R^2$ suggesting that 93.22% of the data fits in the model correctly. With $F - statistic$ and $p - value$ and $\alpha$ (significance) $= 0.05$ we can safely reject the null hypothesis. In other words, the miss events have a significant contribution. Similarly, using equation 1, one can cross-check check the CPI is the summation of CPI components.

From figure 6, we can see that the residual plot in this case is almost linear which depicts that the linear regression model

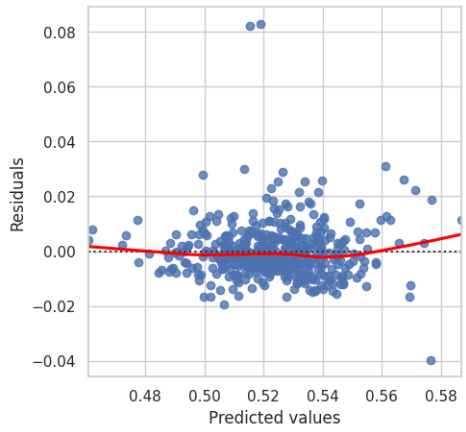


Fig. 6. Residual plot for 531.deepsjeng_r

| | |
|---|---|
| Base CPI | 0.279425 |
| branch-misses:u | 0.127181 |
| L1-dcache-load-misses | 0.014698 |
| L1-icache-load-misses | 0.012830 |
| LLC-load-misses | 0.053782 |
| l2_rqsts.code_rd_miss:u | 0.000890 |
| l2_rqsts.demand_data_rd_miss:u | 0.020932 |
| dtlb_load_misses.walk_pending:u | 0.003491 |
| ocr.demand_data_rd.l3_miss:u | 0.002569 |
| icache_64b.iftag_miss:u | 0.008696 |
| **Predicted CPI** | 0.524493 |
| **True CPI** | 0.524399 |

TABLE XIV
CPI STACK FOR 531.DEEPSJENG_R

is a good fit for our data, i.e., CPI is linearly dependent on the features that we have finalised for this benchmark.

### *F. Exhaustive list of miss events considered*

Complete list of events (before dropping):

1) branch-misses
2) branch-load-misses
3) cache-misses
4) LLC-load-misses
5) LLC-store-misses
6) dTLB-load-misses
7) dTLB-store-misses
8) iTLB-load-misses
9) page-faults
10) l2_rqsts.code_rd_miss (L2 cache misses when fetching instructions)
11) l2_rqsts.demand_data_rd_miss (Demand Data Read miss L2, no rejects)
12) dtlb_load_misses.walk_pending (Number of page walks outstanding for a demand load in the PMH each cycle)
13) dtlb_store_misses.walk_pending (Number of page walks outstanding for a store in the PMH each cycle)
14) itlb_misses.walk_pending (Number of page walks outstanding for an outstanding code request in the PMH each cycle)
15) icache_64b.iftag_miss (Instruction fetch tag lookups that miss in the instruction cache)
16) offcore_requests.l3_miss_demand_data_rd (Counts demand data read requests that miss the L3 cache)
17) ocr.hwpf_l2_rfo.l3_miss (Counts hardware prefetch RFOs (which bring data to L2) that was not supplied by the L3 cache)
18) ocr.demand_data_rd.l3_miss (Counts hardware prefetch data reads (which bring data to L2) that was not supplied by the L3 cache)
19) l2_rqsts.all_demand_miss (Demand requests that miss L2 cache)
20) frontend_retired.itlb_miss (Retired Instructions who experienced iTLB true miss)

## FINAL OBSERVATION

Various important insights can be drawn from the CPI stack in Fig 2. First observation is that the significant part of the CPI for 521, 526 & 531 benchmarks is taken up by branch-misses, which could be due to large number of iterations each having significant number of branch instructions in the code, small Pattern History Table or Branch Table Buffer leading to aliasing. Also the distance between consecutive branch instructions can also affect he number of mispredictions. For performance optimization, this distance could be reduced or one could try to varying the pattern length appropriately.

For data read misses, whether from LLC or L2, data prefetching could enhance the performance.

Finally, we also observed that some datasets can be modelled better using polynomial regression rather than multiple linear regression model.

## REFERENCES

[1] S. Das, J. Werner, M. Antonakakis, M. Polychronakis and F. Monrose, "SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 20-38, doi: 10.1109/SP.2019.00021.
[2] Hennessy, J. L., Patterson, D. A. (2019). Computer Architecture: A Quantitative Approach. India: Elsevier Science.
[3] Eyerman, Stijn & Eeckhout, Lieven & Karkhanis, Tejas & Smith, James. (2007). A Top-Down Approach to Architecting CPI Component Performance Counters. Micro, IEEE. 27. 84-93. 10.1109/MM.2007.3.
[4] Milena Milenkovic, Aleksandar Milenkovic, Jeffrey Kulick, "Demystifying Intel Branch Predictors"