Obtaining CPI Stack for Programs using Hardware Performance Counters and Linear Regression

Aman Choudhary and Shashank Singh

Indian Institute of Science, Bangalore, India

Hardware Performance Counters give us an insight into several events that occur during the course of program execution. In this assignment, we gathered hardware PMU event counts of miss events for a few of the SPEC Benchmark programs and used linear regression to build a model to help us break down CPI into its constituents, creating a CPI stack.

1 Introduction

Modern CPUs have an on-chip hardware that enables monitoring of micro-architectural events (Branches taken, instructions executed, cache misses etc.) which help us analyze the performance of our applications on the processor. This hardware is called a **Performance Monitoring Unit**. The goal of this assignment is to use these event counts to decompose CPI (Cycles Per Instruction) into components that reflect the time spent in various miss events. This is achieved by collecting CPI and miss events data from H/W counters and training a linear regression model to fit the data. The model obtained is then used to create the CPI stack. Since, the model doesn't take specific penalty cycles for various events into account, the CPI stack created is fairly speculative and approximate in nature.

2 Methodology

For collecting performance data, we used the tool *perf* because of its simplicity of use via command line. We then used *R* for pivoting and pre-processing of the data as well as for developing the model and CPI stack. We used *lm* function in *R* to carry out the linear regression.

2.1 Collection of Performance Data

While collecting data through *perf*, we had the option of choosing either generic *perf events* or specific *PMU Hardware Events* to refer the miss events. We decided to go with the latter as former relies on kernel to provide the mapping between generic events and the underlying hardware counter while the latter can be precisely chosen from the Processor Manual.

We sampled 7 performance miss events for all the benchmark programs-input pairs with an interval size of *249 ms*. The sampled events are - *Cycles, Instructions Retired, I-Cache Misses, DTLB Misses, ITLB Misses, Branch Misses, L1, L2, L3 Cache Misses*. *Cycles* and *Instructions Retired* will be used to calculate the observed *CPI* for the model to be trained on.

2.2 Pre-processing and analysis of data

Using *dcast* function in *R*, we pivoted the *perf data* to obtain an interval wise tabular form for all the miss events and corresponding *CPI*. We skipped some intervals from the start to skip the misses due to cold regions. Then, we plotted line charts for all the events to clip areas best suited for regression.

We normalized the miss data using min-max normalization to scale the values between 0 and 1.

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

Also, we looked at correlation between the events which turned out between moderate to high for many of the programs. High correlation means that the determining variables (miss events) are not fully independent events and suggests that there's **multicollinearity** in the data. This can cause an erroneous estimate of the coefficients. We attempted to resolve this by removing some of the events.

2.3 Developing the Model

By observing the line plots for each dataset, we identified the regions best suitable for regression and ***randomly*** divided these into two subsets - ***train*** and ***test*** in a *4:1* ratio. We used linear regression to train the model on the "*train*" *dataset*.

The goal of linear regression is to provide us with the weights $W_0, W_1, W_2, ...$ for the following multivariate linear equation, where M_i denotes the counts of a miss event i.

$$CPI = W_0 + W_1 * M_1 + W_2 * M_2 + ... \tag{2}$$

On the first try, we got several negative values for the regression coefficients. Since, it doesn't make sense to keep negative values, we looked for alternate methods of regression for linear models. We tried *Lasso (Least Absolute Shrinkage Selector Operator)* and *glmnet* regressions to enforce positive coefficients. These methods would take these negative values to zero and adjust the other coefficients accordingly. So, we fell back to Linear Regression and tried with different regions in the data-sets to find the ones giving best coefficients.

After a lot of tries, we could get either all postive coefficients for some benchmarks or a few negative coefficients in others. We removed these events from regression as we attributed this to the high correlation in data.

2.4 Cross Validation of the Model

Using *test* data-set as input to model we predicted the CPI_{pred}. We compared this CPI_{pred} to actual CPI_{obs} for the *test* data-set and calculated the RMSE. The max RMSE found across all 6 benchmark programs was 0.070.

Several other metrics such as R^2, adjusted R^2 values, Residuals, F-statistic and p-value were also measured for every benchmark program - input pair.

2.5 Building the CPI Stack

The coefficients determined by the linear model can be used to estimate the *CPI Stack*. The constant term W_0 and the other linear terms constitute the *CPI* of the program as inferred from $Equation(2)$. Hence, we can say that the term W_0 denotes the *Base CPI* and the other terms - coefficients multiplied by their respective counts, give the portion contributed by each miss event.

We took the average of all miss event counts in the *train* data-set to find each term $W_i * M_i$ and used these to express the contribution of each miss event to the *CPI*, hence forming a *CPI Stack*.

3 Experiments

We performed these steps on all of the benchmarks and selected six out of them. The experiments were done on Ubuntu running on an *Intel i5-4210U* CPU. The six benchmark programs are as follows - **xz_r, deepsjeng_r, leela_r, blender_r, cactuBSSN_r, povray_r**. The Figures 2 to 13 show line charts, CPI stack and model summaries for all these programs. The following table summarizes the *RMSE* values and *CPI* derived from the model using mean of the miss event counts and coefficients of the model for the six benchmark programs.

Benchmark	RMSE Value	CPI derived from model using Eq 2
xz_r	0.070	0.696
deepsjeng_r	0.010	0.604
leela_r	0.009	0.795
blender_r	0.035	0.641
cactuBSSN_r	0.034	0.816
povray_r	0.003	0.433

3.1 xz_r

xz_r is a compression/decompression program, hence we expect it to be memory intensive. This is confirmed by a large Cache Miss share in the CPI stack.

From the line graphs in *Figure 2*, it can be observed that the program execution has two parts and each is periodic in nature across all the events. This suggests that program is executed in loops. A spike is observed in ITLB Misses, I-Cache Misses, Branch Misprediction at the boundary of these two parts. This can be attributed to a jump or branch instruction which subsequently lead to spike in CPI.

3.2 deepsjeng_r

deepsjeng_r is based on tree search algorithm. From the CPI stack in *Figure 3*, it appears that the program is CPU bound. Base CPI has the dominant share indicating that the instruction mix has more complex instructions or low Instruction Level Parallelism or both. The regular spikes in *CPI* can be attributed to the corresponding spikes in *L1* and *L2 Cache Misses*.

3.3 leela_r

leela_r is based on tree search algorithm and pattern recognition and its CPI stack looks quite similar to that of deepsjeng_r. From *Figure 6*, it appears that this program is also CPU bound and has complex instructions and/or low ILP. From the line charts of miss events, it appears that the execution period has two identical halves suggesting that program is doing some logical operation twice.

3.4 blender_r

blender_r is a 3D rendering and animation program. On the first regression attempt, we got several negative coefficients. We tried with different regions in the program but couldn't attain positive coefficients, so we had to remove 3 miss events from the equation. We attributed this to the presence of high correlation between miss events in the data, particularly cache events. The CPI stack for this can be referred from *Figure 7*.

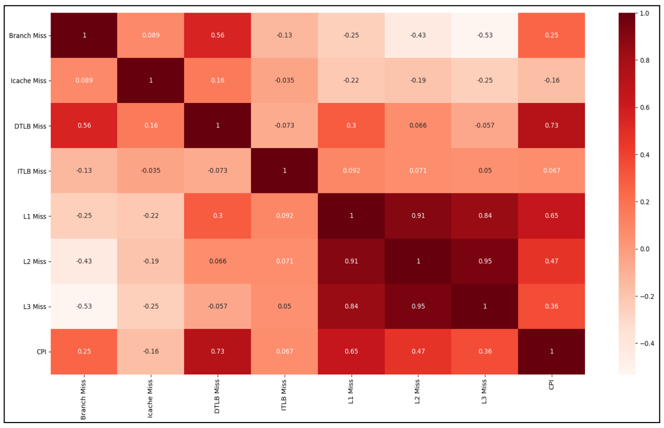


Figure 1: *Correlation Heatmap for miss events of blender_r*

3.5 cactuBSSN_r

cactuBSSN_r is a number crunching program to solve complex equations. From *Figure 10*, we can observe that almost all miss events have peak values throughout the duration. This program has the highest CPI across the six benchmark programs.

3.6 povray_r

povray_r is a graphics rendering program which renders an image and calculates a similarity index SSIM. The CPI stack for this program in *Figure 11* indicates that less cycles have been spent due to miss events and that the program spends much of its cycles in functional execution. Also, this program has the least CPI across the six benchmark programs.

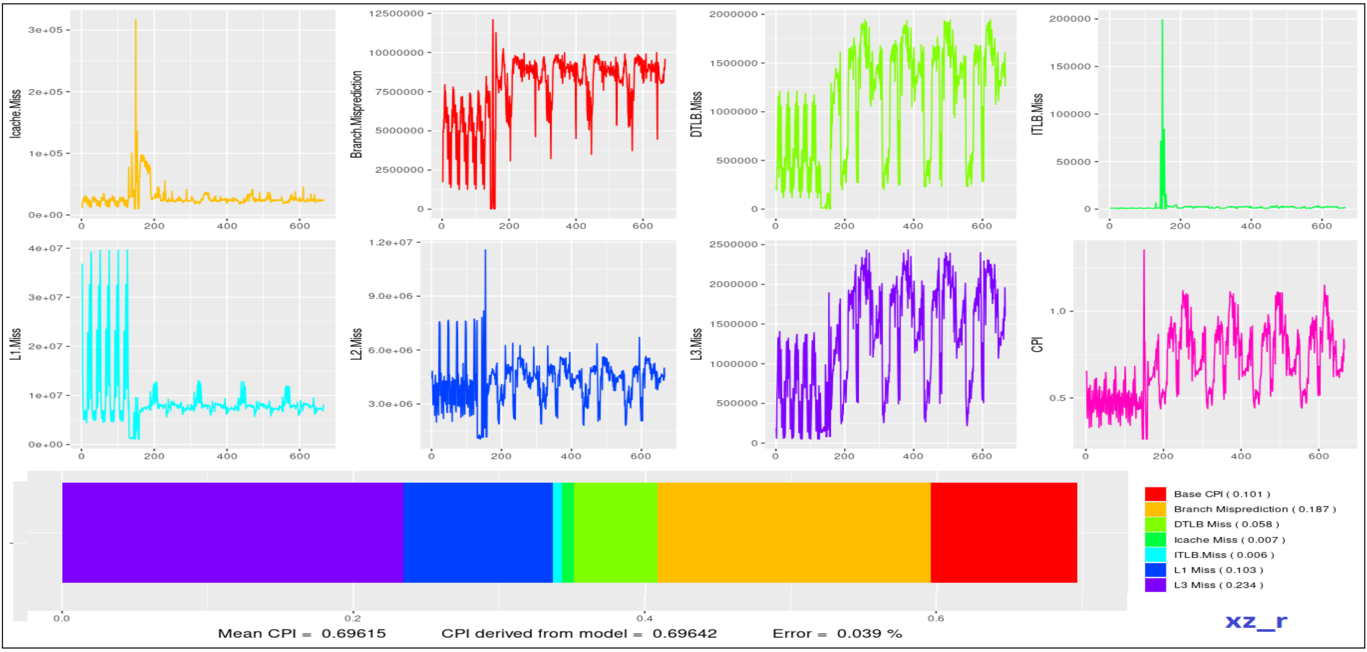


Figure 2: *Charts and CPI Stack for xz_r benchmark program*

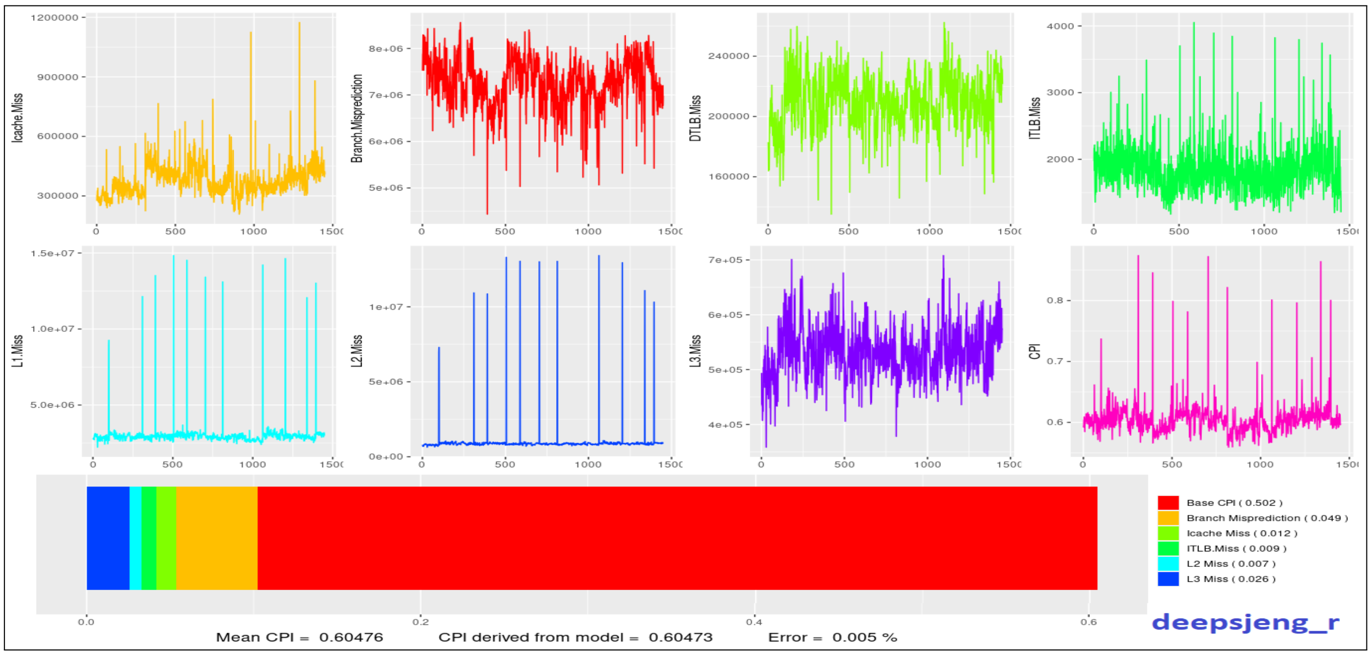
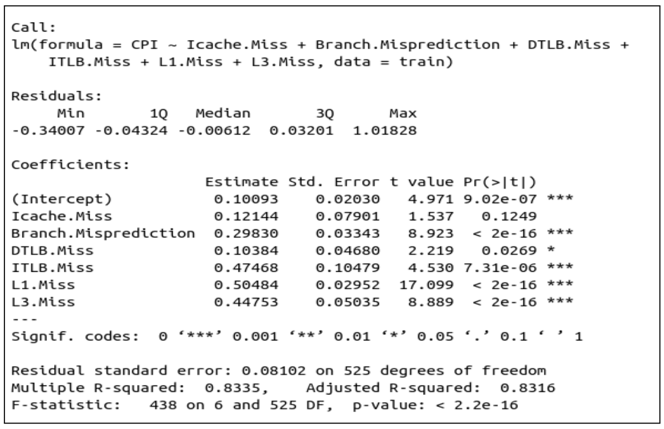


Figure 3: *Charts and CPI Stack for deepsjeng_r benchmark program*



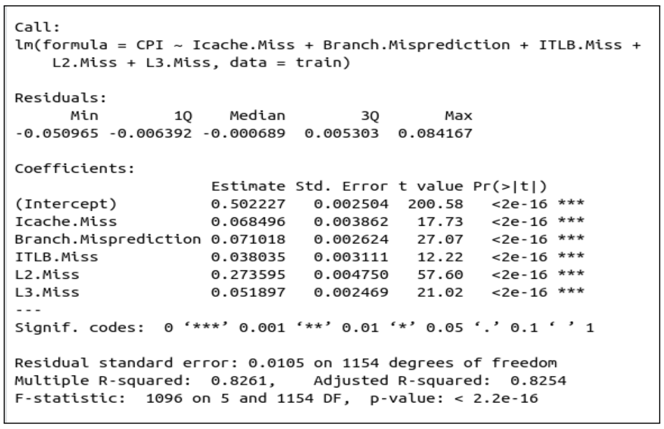
```
Call:
lm(formula = CPI ~ Icache.Miss + Branch.Misprediction + DTLB.Miss +
    ITLB.Miss + L1.Miss + L3.Miss, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-0.34007 -0.04324 -0.00612  0.03201  1.01828

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)           0.10093    0.02030   4.971 9.02e-07 ***
Icache.Miss           0.12144    0.07901   1.537   0.1249
Branch.Misprediction  0.29830    0.03343   8.923  < 2e-16 ***
DTLB.Miss             0.10384    0.04680   2.219   0.0269 *
ITLB.Miss             0.47468    0.10479   4.530 7.31e-06 ***
L1.Miss               0.50484    0.02952  17.099  < 2e-16 ***
L3.Miss               0.44753    0.05035   8.889  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.08102 on 525 degrees of freedom
Multiple R-squared:  0.8335,    Adjusted R-squared:  0.8316
F-statistic:   438 on 6 and 525 DF,  p-value: < 2.2e-16
```

Figure 4: *Model summary for xz_r program*



```
Call:
lm(formula = CPI ~ Icache.Miss + Branch.Misprediction + ITLB.Miss +
    L2.Miss + L3.Miss, data = train)

Residuals:
      Min        1Q    Median        3Q       Max
-0.050965 -0.006392 -0.000689  0.005303  0.084167

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          0.502227   0.002504  200.58   <2e-16 ***
Icache.Miss          0.068496   0.003862   17.73   <2e-16 ***
Branch.Misprediction 0.071018   0.002624   27.07   <2e-16 ***
ITLB.Miss            0.038035   0.003111   12.22   <2e-16 ***
L2.Miss              0.273595   0.004750   57.60   <2e-16 ***
L3.Miss              0.051897   0.002469   21.02   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0105 on 1154 degrees of freedom
Multiple R-squared:  0.8261,    Adjusted R-squared:  0.8254
F-statistic:  1096 on 5 and 1154 DF,  p-value: < 2.2e-16
```

Figure 5: *Model summary for deepsjeng_r program*

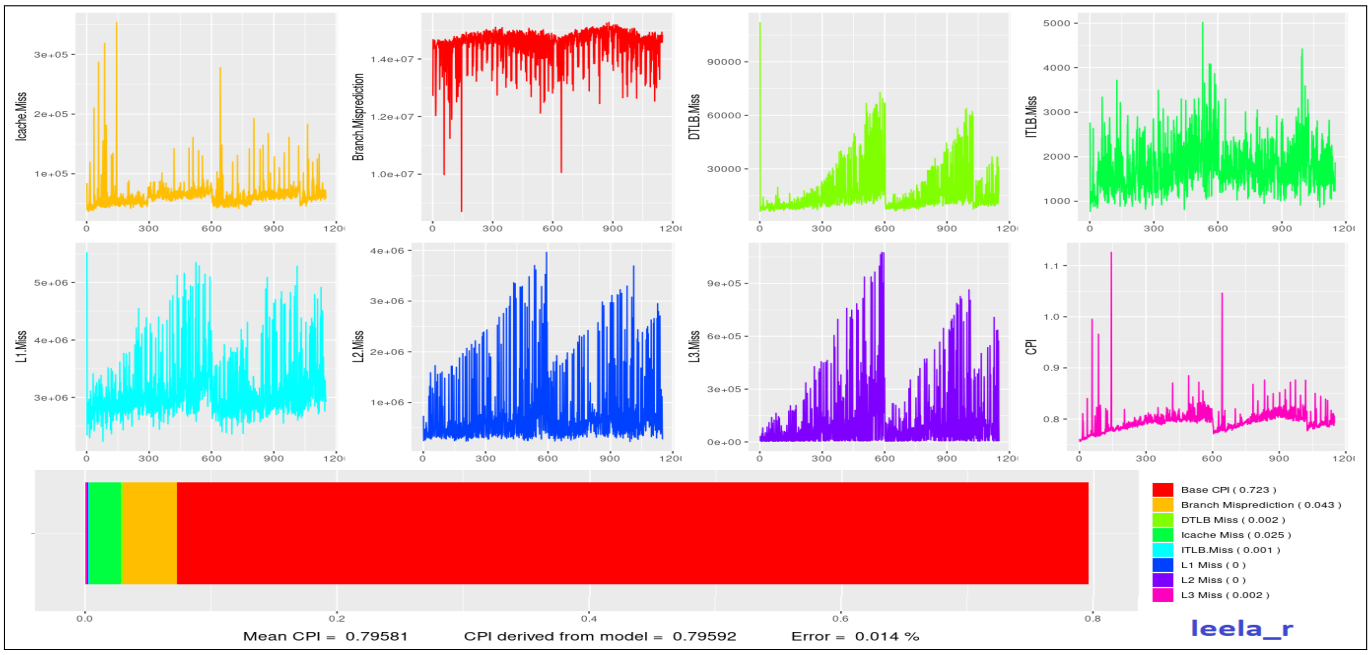


Figure 6: *Charts and CPI Stack for leela_r benchmark program*

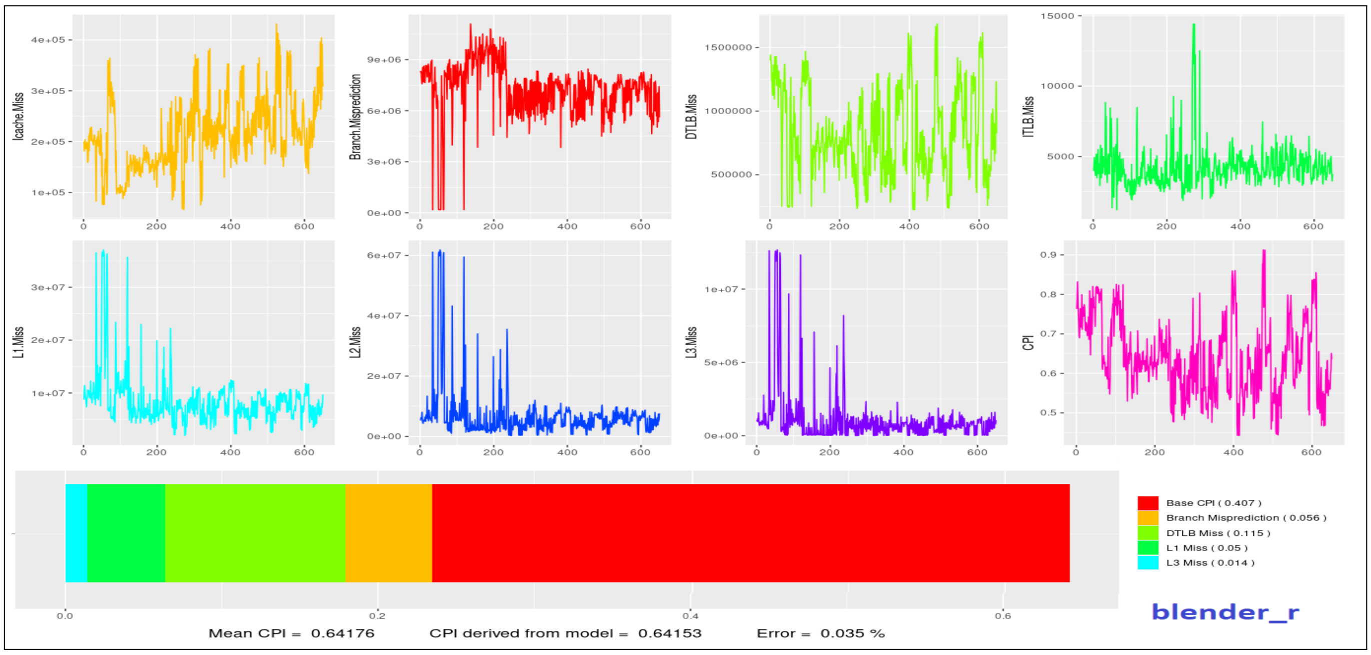
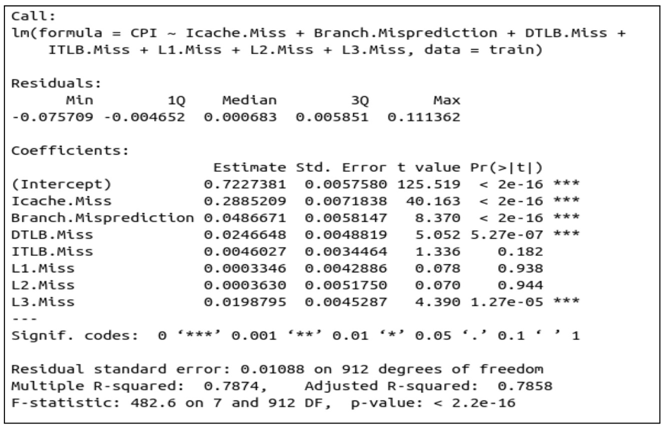


Figure 7: *Charts and CPI Stack for blender_r benchmark program*



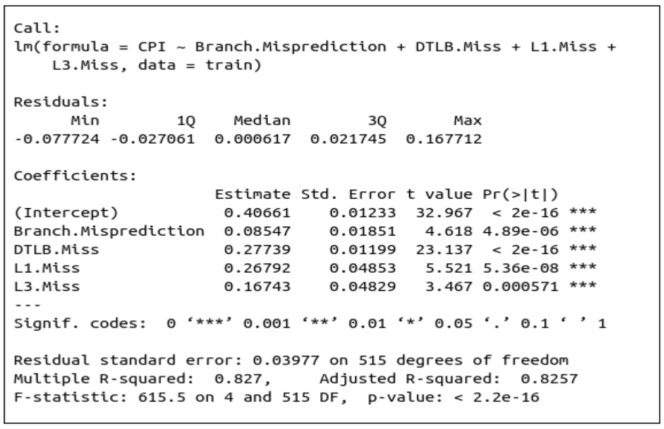
```
Call:
lm(formula = CPI ~ Icache.Miss + Branch.Misprediction + DTLB.Miss +
    ITLB.Miss + L1.Miss + L2.Miss + L3.Miss, data = train)

Residuals:
      Min        1Q    Median        3Q       Max
-0.075709 -0.004652  0.000683  0.005851  0.111362

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          0.7227381  0.0057580 125.519  < 2e-16 ***
Icache.Miss          0.2885209  0.0071838  40.163  < 2e-16 ***
Branch.Misprediction 0.0486671  0.0058147   8.370  < 2e-16 ***
DTLB.Miss            0.0246648  0.0048819   5.052 5.27e-07 ***
ITLB.Miss            0.0046027  0.0034464   1.336    0.182
L1.Miss              0.0003346  0.0042886   0.078    0.938
L2.Miss              0.0003630  0.0051750   0.070    0.944
L3.Miss              0.0198795  0.0045287   4.390 1.27e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01088 on 912 degrees of freedom
Multiple R-squared:  0.7874,    Adjusted R-squared:  0.7858
F-statistic: 482.6 on 7 and 912 DF,  p-value: < 2.2e-16
```

Figure 8: *Model summary for leela_r program*



```
Call:
lm(formula = CPI ~ Branch.Misprediction + DTLB.Miss + L1.Miss +
    L3.Miss, data = train)

Residuals:
      Min        1Q    Median        3Q       Max
-0.077724 -0.027061  0.000617  0.021745  0.167712

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)           0.40661    0.01233  32.967  < 2e-16 ***
Branch.Misprediction  0.08547    0.01851   4.618 4.89e-06 ***
DTLB.Miss             0.27739    0.01199  23.137  < 2e-16 ***
L1.Miss               0.26792    0.04853   5.521 5.36e-08 ***
L3.Miss               0.16743    0.04829   3.467 0.000571 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03977 on 515 degrees of freedom
Multiple R-squared:  0.827,     Adjusted R-squared:  0.8257
F-statistic: 615.5 on 4 and 515 DF,  p-value: < 2.2e-16
```

Figure 9: *Model summary for blender_r program*

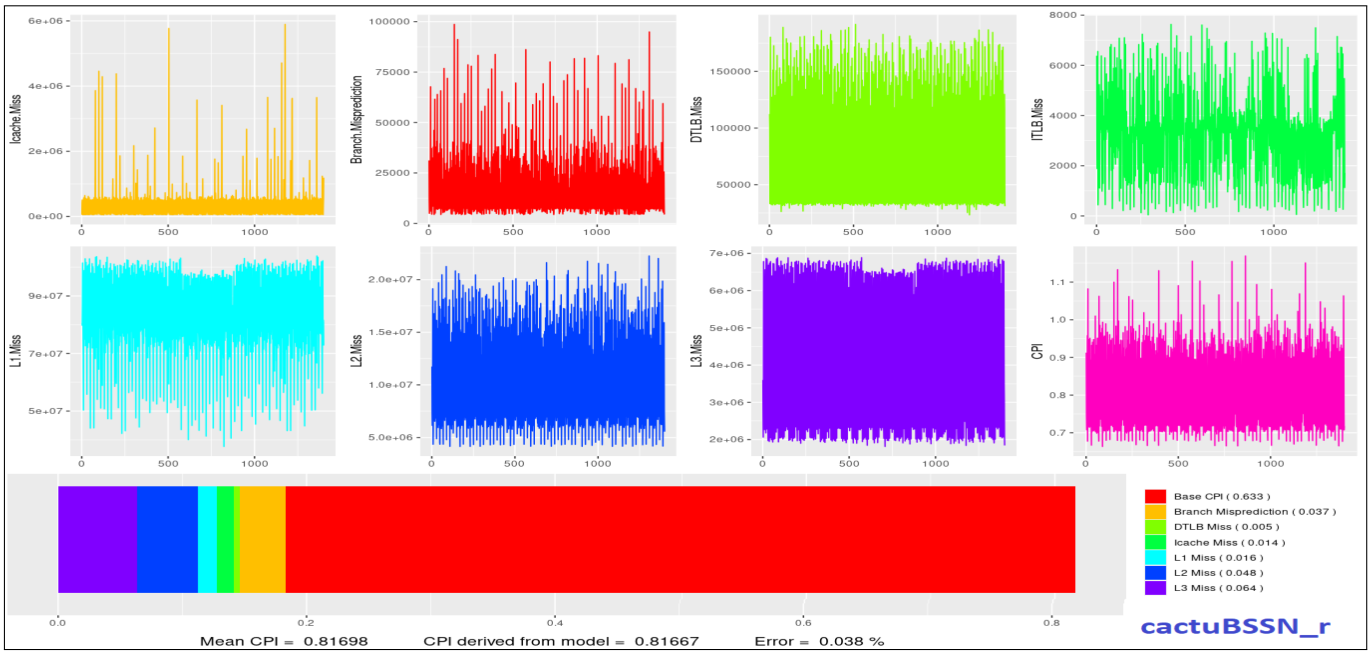


Figure 10: *Charts and CPI Stack for cactuBSSN_r benchmark program*

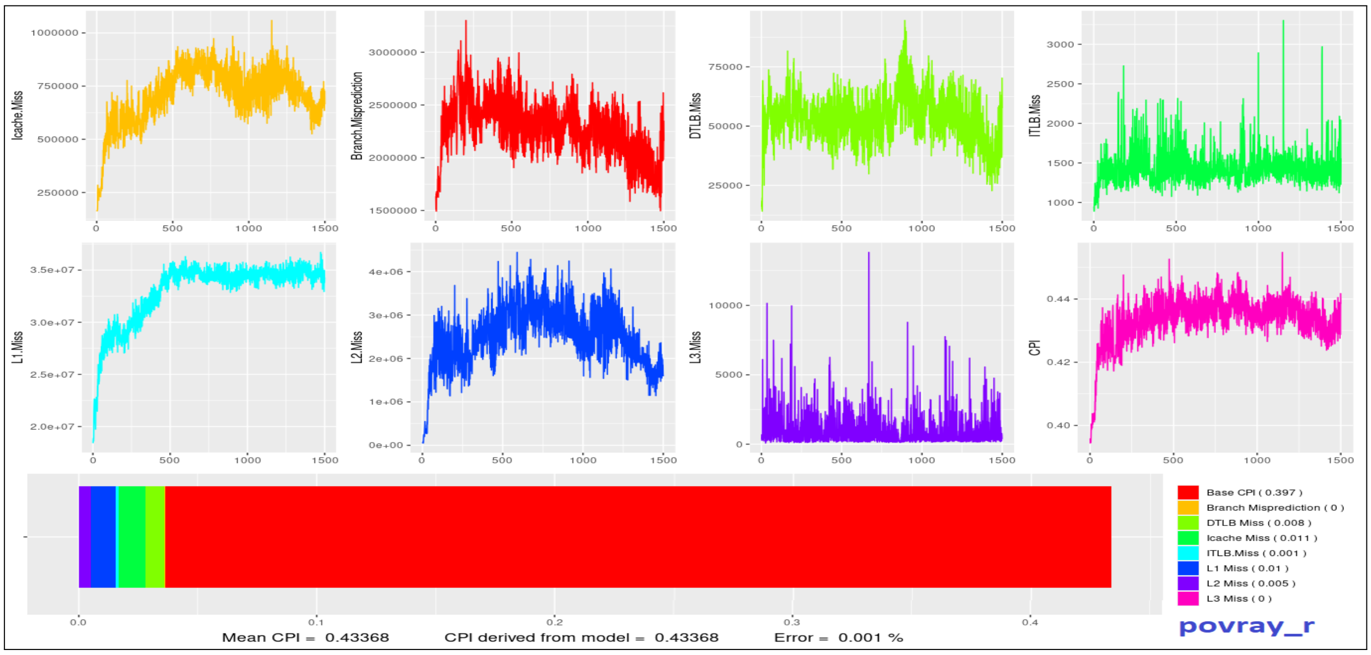


Figure 11: *Charts and CPI Stack for povray_r benchmark program*

```
Call:
lm(formula = CPI ~ Icache.Miss + Branch.Misprediction + DTLB.Miss +
    L1.Miss + L2.Miss + L3.Miss, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-0.16592 -0.01480 -0.00039  0.01132  0.33396

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          0.633223   0.009577  66.122   <2e-16 ***
Icache.Miss          0.281255   0.015293  18.391   <2e-16 ***
Branch.Misprediction 0.314478   0.013176  23.867   <2e-16 ***
DTLB.Miss            0.013144   0.008468   1.552    0.121
L1.Miss              0.023965   0.015194   1.577    0.115
L2.Miss              0.152812   0.009171  16.662   <2e-16 ***
L3.Miss              0.158095   0.007807  20.250   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03751 on 1113 degrees of freedom
Multiple R-squared:  0.8554,	Adjusted R-squared:  0.8546
F-statistic:  1097 on 6 and 1113 DF,  p-value: < 2.2e-16
```

Figure 12: *Model summary for cactuBSSN_r program*

```
Call:
lm(formula = CPI ~ Icache.Miss + Branch.Misprediction + DTLB.Miss +
    ITLB.Miss + L1.Miss + L2.Miss + L3.Miss, data = train)

Residuals:
       Min         1Q     Median         3Q        Max
-0.0144719 -0.0027089 -0.0000355  0.0025635  0.0133394

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          0.3973475  0.0008359 475.333  < 2e-16 ***
Icache.Miss          0.0181833  0.0015596  11.659  < 2e-16 ***
Branch.Misprediction 0.0004614  0.0011258   0.410   0.6820
DTLB.Miss            0.0168484  0.0011548  14.589  < 2e-16 ***
ITLB.Miss            0.0056537  0.0013292   4.253 2.27e-05 ***
L1.Miss              0.0128877  0.0013434   9.593  < 2e-16 ***
L2.Miss              0.0090949  0.0011834   7.685 3.17e-14 ***
L3.Miss              0.0024210  0.0013457   1.799   0.0723 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.003936 on 1192 degrees of freedom
Multiple R-squared:  0.7193,	Adjusted R-squared:  0.7176
F-statistic: 436.3 on 7 and 1192 DF,  p-value: < 2.2e-16
```

Figure 13: *Model summary for povray_r program*

4 Quality of the Model

A good way to assess the quality of the fit of the model is to look at the residuals or the differences between the observed and predicted values which are negligible (order of 10^{-3}) in all our models. Another parameter for a good fit is R^2, which is fairly high in all the models $(0.7 - 0.8)$ but at the same time not so high to indicate over-fitting. Across all models, we find that majority of features prove to be of significance in predicting the *CPI* as their *p-values* are very less compared to 0.05 indicating that the null hypothesis can be rejected comfortably.

5 Conclusion

While measuring the counts from *perf*, we also noticed that multiplexing of hardware counters occurs when we try to monitor more than 4 events in our machine. This multiplexing can lead to inaccuracies in data. While trying to train the models, it was a challenge to ensure non-negative coefficients as negative values don't make sense. Ultimately, we had to drop a few events to come up with the desired coefficients. A non-linear approach may prove to be more apt in this scenario.

We could also try conducting a more meaningful analysis by leveraging techniques such as Principal Components Analysis (PCA) to extract truly independent variables. The CPI stacks built by this exercise is an approximate one and sometimes agrees with reality. However, in some cases, we admit that we are unable to interpret the CPI stack in a meaningful and sensible way. So, there is a scope of lot of improvement in this regard.

6 References

[1] *A Top-Down Approach to Architecting CPI Component Performance Counters Article in IEEE Micro · February 2007*

[2] *Intel® 64 and IA-32 Architectures Software Developer's Manual - Volume 3B: System Programming Guide, Part 2*

[3] *Linux Kernel Profiling with perf. Available at https://perf.wiki.kernel.org/index.php/Tutorial*

[4] *SPEC CPU®2017 Overview. Available at https://www.spec.org/cpu2017/Docs/overview.html*

[5] *Brendan Gregg's blog for perf. Available at http://www.brendangregg.com/perf.html*

[6] *R Documentation at https://www.rdocumentation.org/*