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

Name: Chetan Pant, Mukul Sharma **Instructor:** Prof. R. Govindarajan
M.Tech D

Introduction :

We are required to obtain the CPI (Cycles Per Instruction) stack for the given benchmarks using hardware performance monitoring counters. We are obtaining Performance counter values using perf Tool. We have to divide the total CPI of an application into time spent in different miss events. We also have to report the different quality parameters like RMSE,R^2 , adjusted R^2 values, etc. and observation in different benchmark programs.

Data Collection :

Using the perf tool, we are collecting the data sets at different interval sizes using the following command:

perf stat -I interval_size -e branch misses:u,L1-dcache-load-misses:u,L1-icache-load-misses:u, dTLB-load-misses:u,dTLB-store-misses:u,iTLB-load-misses:u,branch-load-misses:u,L2-misses:u, cycles:u,instructions:u -o abc.txt bash ./run.sh

We are choosing interval sizes for different benchmarks in such a way that we atleast get 1000 datasets to train the regression model.

The coefficients should be non negative showing they are additive to the total CPI value. We are calculating the coefficients of the following 8 miss events using linear model:

L1 dcache load misses, L1 icache load misses, dTLB load misses, dTLB store misses, iTLB load misses, Branch load misses, L2 misses, branch misses

After collecting data, we parsed our output .txt file to .csv file to run it on the model. We are training our model using no. of miss events per instruction in a particular interval and the model is outputting the total CPI value.

Unit of Coefficients : no. of cycles per miss event

Unit of input features : no. of misses per instruction in each interval.

Normalization is done by dividing the counter value with the number of instructions in the given interval and similarly CPI is computed as total execution cycles divided by number of instructions.

Simple Linear Regression Model:

We have built our model using different libraries in python. We imported the Sklearn library and used the ols model to train the data sets.

Library Used: numpy, pandas, sklearn

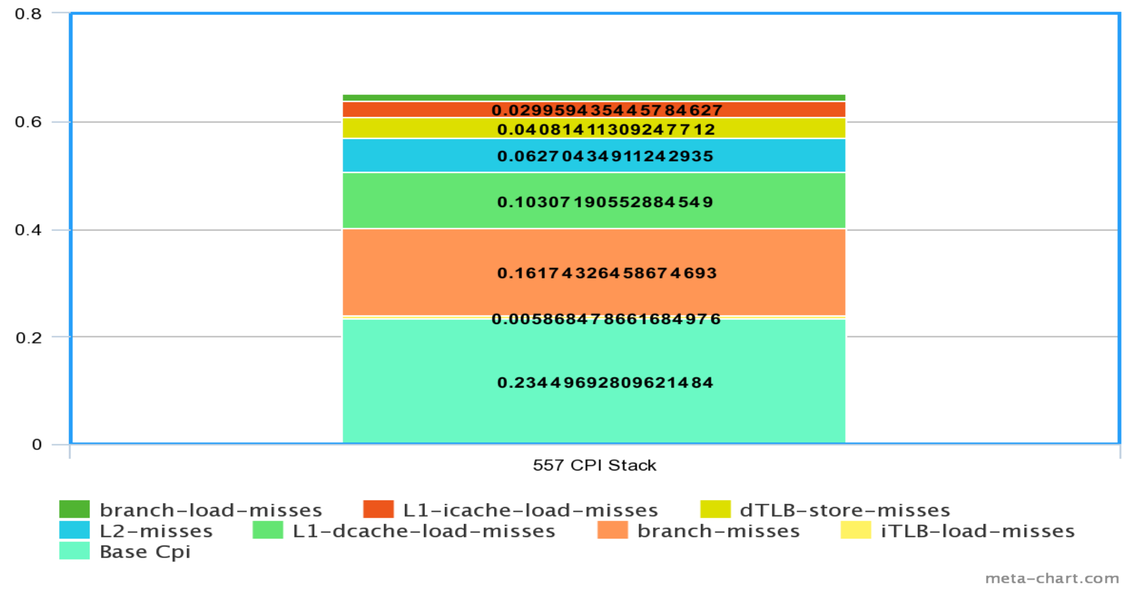
Total CPI : $a_0 + a_1w_1 + a_2w_2 + a_3w_3 + a_4w_4 + a_5w_5 + a_6w_6 + a_7w_7 + a_8w_8$

where a_0 is Base CPI and a_iw_i represents contribution of different miss events in total CPI value.

1. 557. XZ_R SPEC INT Benchmark

Mean CPI : 0.652287368664906 Calculated CPI : 0.6527602354509167

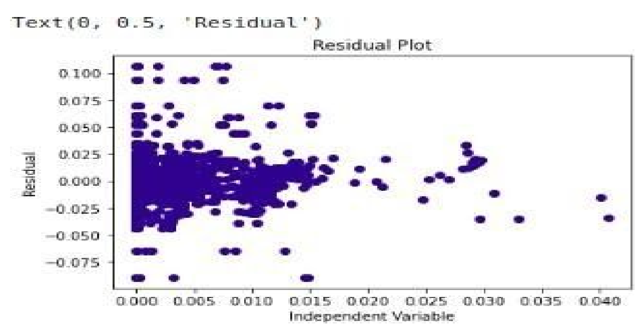
CPI STACK :



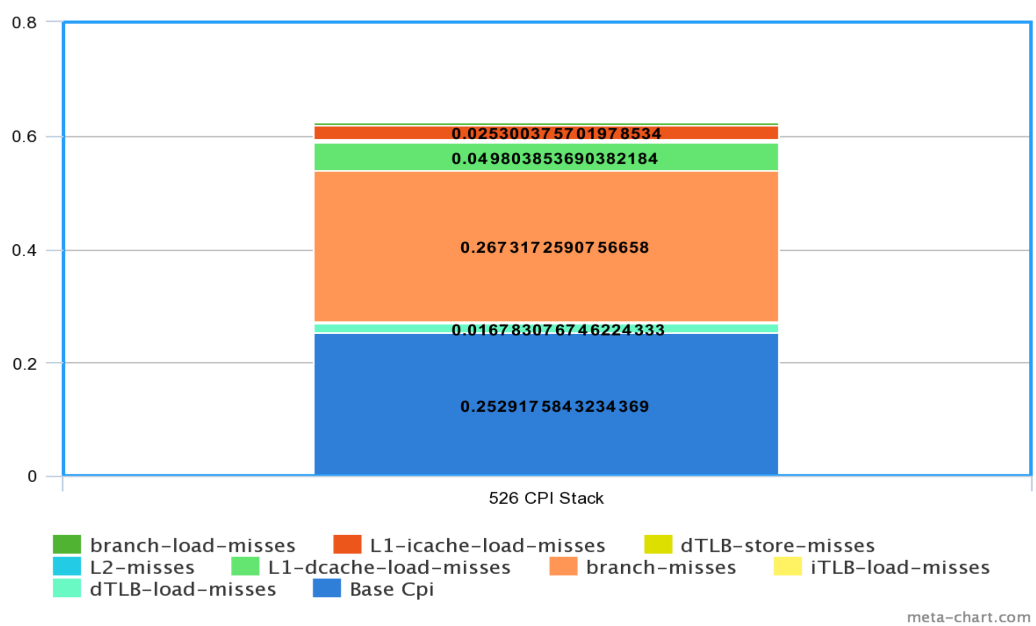
	O/p Coefficients of model	P value
Intercept	0.2345	0.000
dTLB store misses	72.6314	0.000
L2-misses	80.9938	0.000
L1 dcache load misses	10.0994	0.000
Branch misses	28.0711	0.000
iTLB load misses	8326.1279	0.000
dTLB cache misses	0.000	0.000
L1 icache load misses	518.4965	0.000
Branch load misses	2.4531	0.331

RMSE: 0.018398
Plot: R^2 Value: 0.99
Adjusted R^2 Value: 0.99
F statistic: 9.164e+05

Residuals



2. 526. BLENDER_R SPEC FP Benchmark



Mean CPI : 0.624209263928867 Calculated CPI : 0.624523905638984

	O/p Coefficients of model	P value
Intercept	0.2529	0.000
dTLB load misses	111.4956	0.000
dTLB store misses	758.0573	0.000
L2-misses	0.9785	0.745
L1 dcache load misses	6.2962	0.000
Branch misses	45.3494	0.000
iTLB load misses	997.2668	0.004
L1 icache load misses	31.1157	0.000
Branch load misses	1.1502	0.668

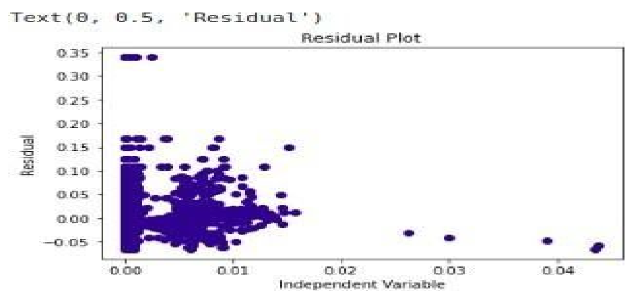
RMSE: 0.032615

R^2 Value: 0.890

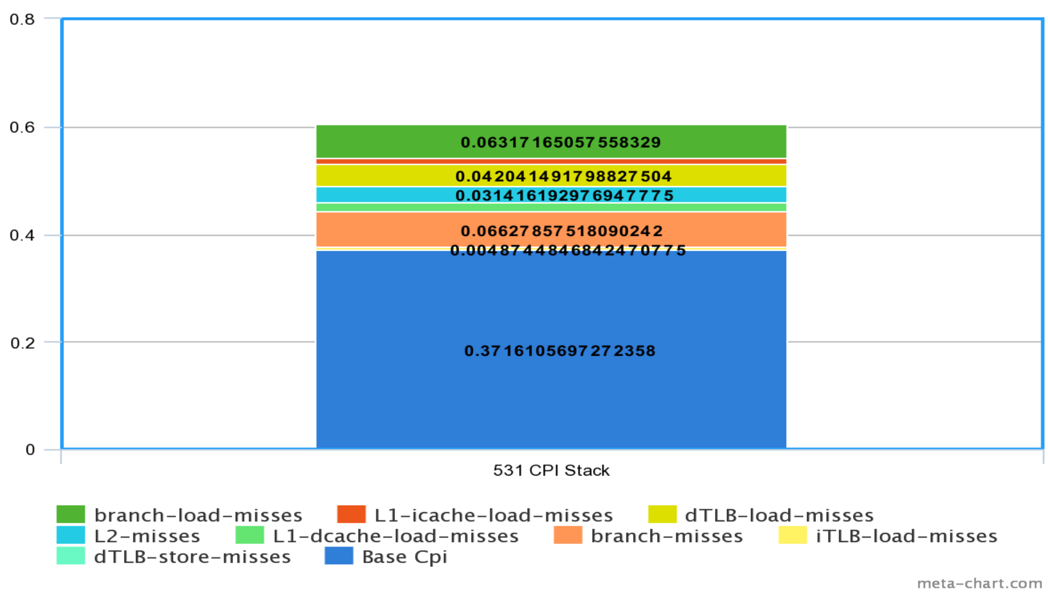
Adjusted R^2 Value: 0.889

F statistic: 995.0

Residuals Plot:



3. 531. DEEPJENG_R SPEC INT Benchmark

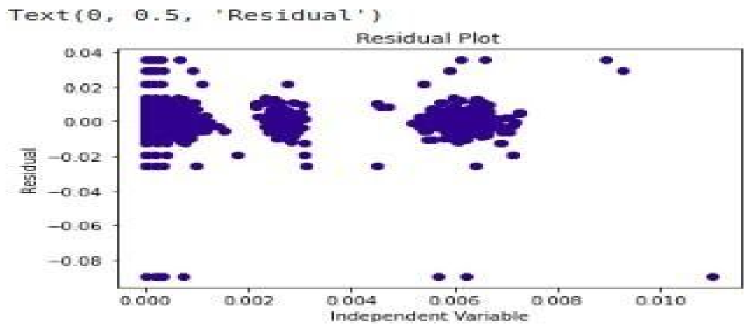


Mean CPI : 0.6033488246615073 Calculated CPI : 0.6034427269001889

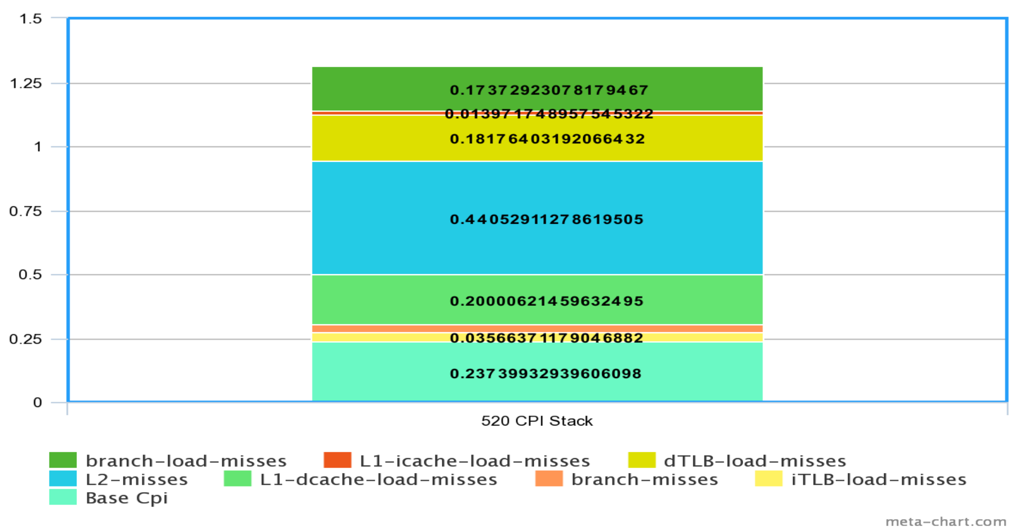
	O/p Coefficients of model	P value
Intercept	0.3588	0.000
dTLB load misses	129.0523	0.000
dTLB store misses	244.9779	0.000
L2-misses	98.3668	0.000
L1 dcache load misses	8.6719	0.000
Branch misses	10.5167	0.000
iTLB load misses	15290.0	0.000

L1 icache load misses	20.5085	0.000
Branch load misses	9.8010	0.000

RMSE: 0.007711 **Residual**
Plot: R^2 Value: 0.99
Adjusted R^2 Value: 0.99
F statistic: 553e+09



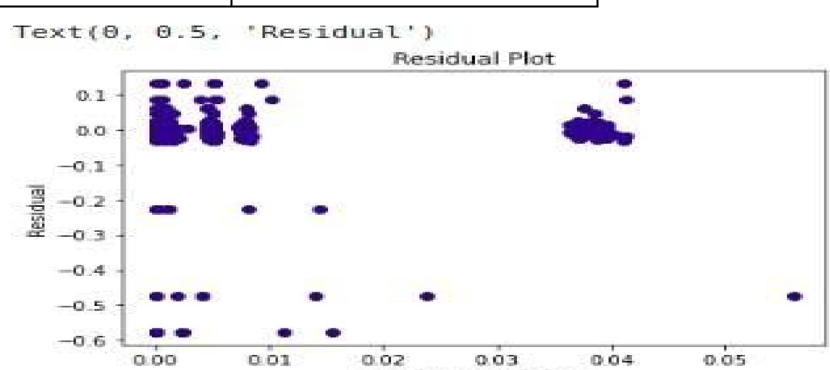
4. 520. OMNETPP_R SPEC INT Benchmark



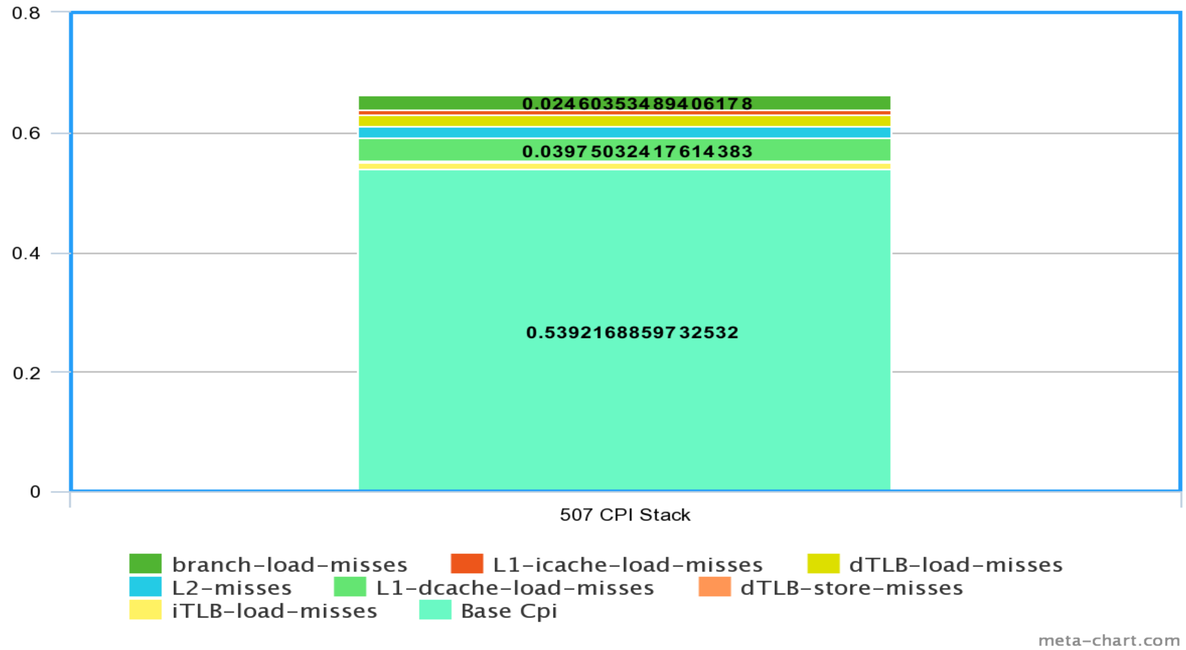
Mean CPI : 1.2118403126125008 **Calculated CPI** : 1.2116445375576184

	O/p Coefficients of model	**P value**
Intercept	0.2374	0.000
dTLB load misses	37.8037	0.000
L2-misses	57.1497	0.000
L1 dcache load misses	12.0728	0.000
Branch misses	6.1642	0.514
iTLB load misses	2368.8909	0.000
dTLB store misses	0.000	0.000
L1 icache load misses	12.0728	0.000
Branch load misses	36.7068	0.000

RMSE: 0.0148765 **Residual Plot:**
R^2 Value: 0.943
Adjusted R^2 Value: 0.943
F statistic: 2406.0



5. 507. CactuBSSN_r SPEC FP Benchmark



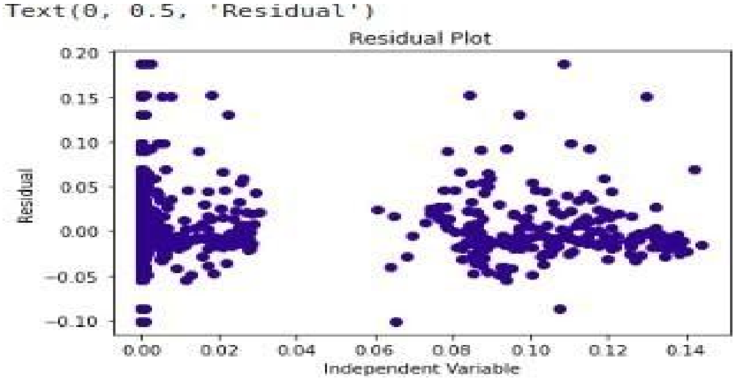
Mean CPI : 0.6581094039305134 **Calculated CPI** : 0.6598032723834679

	O/p Coefficients of model	P value
Intercept	0.5392	0.000
dTLB load misses	195.8784	0.000
L2-misses	8.2052	0.000
Branch misses	0.000	0.000
L1 dcache load misses	0.3944	0.004
dTLB Store misses	120.7613	0.000
iTLB load misses	1.504e+04	0.000
L1 icache load misses	0.5626	0.000
Branch load misses	1685.8574	0.000

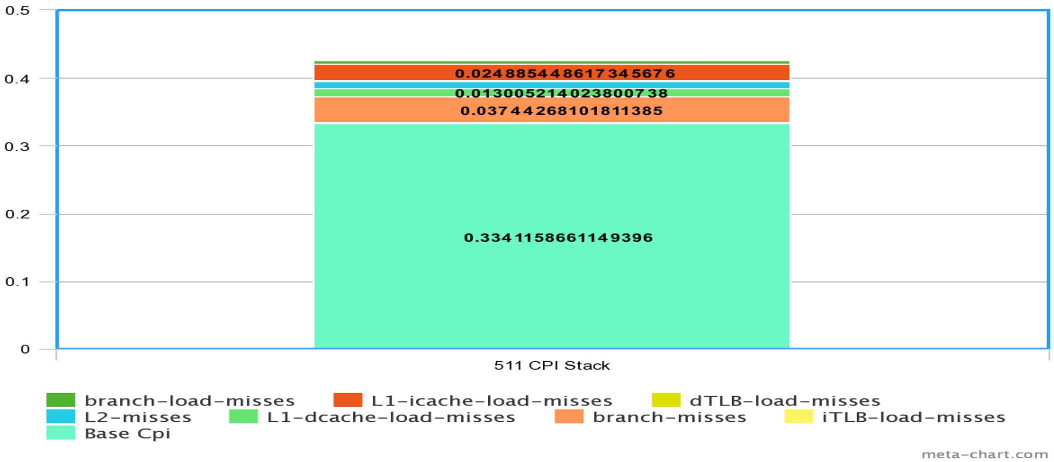
RMSE: 0.01487650
R^2 Value: 0.713
Adjusted R^2 Value: 0.711
F statistic: 356.8

Residual Plot:

Text(0, 0.5, 'Residual')



6. 511. povray_r SPEC FP Benchmark

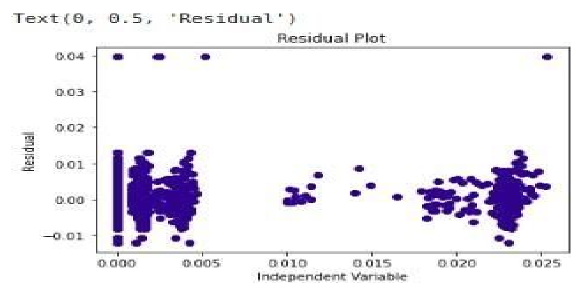


Mean CPI : 0.4262694100800861 Calculated CPI : 0.4261276795882872

	O/p Coefficients of model	P value
Intercept	0.3341	0.000
dTLB load misses	1701.2948	0.005
L2-misses	1022.21176	0.000
L1 dcache load misses	0.5957	0.000
Branch misses	27.7637	0.000
iTLB load misses	383.8243	0.502
dTLB store misses	0.000	0.000
L1 icache load misses	7.091796	0.000
Branch load misses	3.6575	0.003

RMSE: 0.00814116
R^2 Value: 0.928
Adjusted R^2 Value: 0.927
F statistic: 1805.0

Residuals Plot:



Key Observations:

- From CPI stack we can easily see that which architectural event is taking more time for a particular benchmark.If we can improve the largest time taking event ,our performance would be better. we can also easily compare the event rates in these benchmarks.
- 557.xz_r and 531.deepjeng INT benchmarks takes more CPI for branch misses than any other microarchitectural events which means they have more number of branch Instrucution.In contrast to 507.cactuBSSN_ and 511.povray_r FP benchmarks have less branch misses CPI which means they have less number of branch instructions.
- 520.Omnetpp_r takes a large amount of time in servicing L2 cache misses and also it has the largest total CPI among all 6 benchmarks.
- 511. povray_r benchmark suffers from high L1 instruction cache miss rate and has more frontend stalls.
- 520 omnetpp_r and 507 cactuBSSN_r suffer from higher data cache miss rates which means that they have more number of memory instructions(load/store). They have more backend stalls.
- Benchmarks like 511 povray_r benchmarks suffer from true data dependencies(RAW) because they have the highest base CPI and contribution of other miss events in CPI is very less.
  Therefore, more instructions are waiting in the reservation table because of the dependencies and it's the main cause of pipeline stalls.