Characterizing Microprocessor Benchmarks

Towards Understanding the Workload Design Space

by

Michael Arunkumar, B.E.

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2003

Characterizing Microprocessor Benchmarks

Towards Understanding the Workload Design Space

APPROVED BY
SUPERVISING COMMITTEE

_____

_____

Dedication

To my family and all my teachers who have always shown me the right way, due to whom, I am where I am now.

Acknowledgements

I would like to thank Dr. Lizy Kurian John for her invaluable guidance and counseling from my entry into the University of Texas at Austin till the completion of this report. I would also like to thank Dr. Anthony Ambler, for accepting to be the second reader. I would also like to thank members of the Laboratory of Computer Architecture for their help during the work towards completion of this report.

December 3, 2003

Characterizing Microprocessor Benchmarks Towards Understanding the Workload Design Space

by

Michael Arunkumar, M.S.E

The University of Texas at Austin, 2003

SUPERVISOR: Lizy Kurian John

Reducing the time spent during the initial exploration of the design space of a microprocessor is being increasingly important especially from time to market perspectives. Benchmarks running on simulation models of the microprocessor take substantial amount of time, and the time it takes to run increases with the complexity of the microprocessor modeling. Most of the benchmarks used nowadays are enormous and take substantial amounts of time to run on a detailed microprocessor simulation. The idea of the report is to identify redundancies within a benchmark or redundancies among benchmarks, which stress on the same characteristic of the microprocessor thereby identifying for elimination, which will contribute towards time saving while simulating microprocessors in the early design phase. The SPEC-int95 benchmarks were

characterized, to understand how spread out or clustered they are in the workload design space.

Table of Contents

1. Introduction

Time to market has and will always be an important criterion directing microprocessor design and the institution designing it. Reduction of the time to market has direct significance towards an institution's success. One way to contribute to this is in the early design phase of the microprocessor.

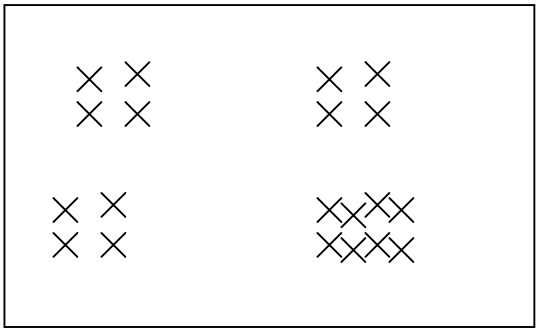
Simulation models of various levels of accuracy are created during this design phase using a particular software language. The models represent both the structure and behavior of the microprocessor in various ways. The more detailed the simulation is, the longer it takes to simulate a cycle. Computer system design is an experimental procedure that is usually a result of measuring or estimating the running time of workload or the result that it produces. A workload could be considered as a benchmark program given certain particular inputs.

Benchmark programs, like the SPECint-95 or the SPECint-2000 are given as inputs to the simulations of the microprocessor, in an attempt to explore the behavior of the system under real workload conditions. Running simulations with benchmarks are very time consuming, especially since each benchmark contains numerous input sets testing for various conditions. According to John

et al. [1] certain benchmark-input pairs result in testing the same area in the potential workload space.

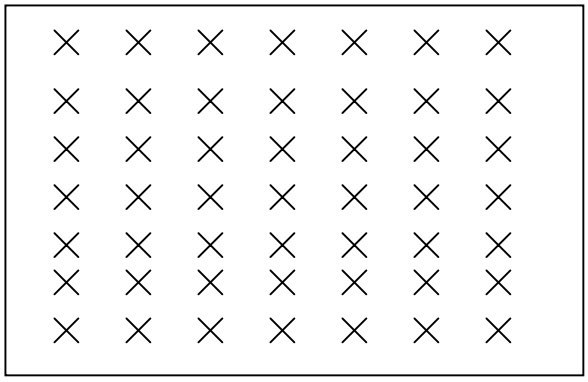
Consider the complete search space during the design phase of a microprocessor as a multidimensional space where each dimension is representative of each characteristic. The results each run of the benchmark would permeate into this space depending on how it stressed the benchmark. By identifying portions of this space, which are stressed redundantly, by either the same benchmark, or over various benchmarks, it would give a chance to think towards elimination of these redundancies. As an illustration, consider the multi-dimensional space compressed into a two-dimensional space.

Fig. 1 Potential Span of existing benchmarks in a multi-dimensional space



In reality, the present benchmarks do not span the entire possible design space and end up stressing the same characteristic several times and leave certain characteristics untouched. However, when running the simulations it would be preferred not to leave any stone unturned. By characterizing the benchmarks the program-input pairs stressing upon a portion of the application space already tested by another program-input pair, can be identified, thereby attempting to eliminate redundancy. Ideally, benchmarks should stress all locations of the search space thereby stressing the microprocessor in all aspect as shown below.

Fig. 2 Optimal Span of ideal benchmarks



This study initially attempts to identify a set of performance metrics of a computer system based on upon the results from hardware performance

counters monitoring the impact of SPECint-95 benchmarks on the computer system. The study then attempts to extract the dependence of the performance metrics on the Cycles per Instruction for various benchmarks and their inputs pairs. A further attempt is made to obtain a principal component, which is preferably a combination of one or very few benchmarks, thereby reducing the dimensionality of the set of performance metrics, thereby paving the way for further study.

It would be very difficult to display the enormous workload space in a comprehendible way. Initially after collecting the data using the hardware performance counters, regression analysis is performed in order to understand the impact of the benchmarks on the performance metrics in the computer system. The system's performance for a limited number of inputs is measured and used to produce a mathematical model to describe the behavior of the system over a range of values. The model obtained will be called a regression model. After the linear regression model is developed, it is needed to know how well the equation that was formed, actually models the measured data. In other words, it is necessary to know how the strength of the linear correlation between the input and output is. A study of the correlation of various performance metrics with the CPI is obtained.

The same data is then analyzed using principal component analysis. In this analysis, the multi-dimensional workspace (say, n dimensions) is reduced to a lesser dimensional workspace (say, m, where n>>m) thereby facilitating visualization of enormous workspace without the loss of valid information. All these analyses are carried out on the SPECint-95 benchmarks as described in the methodology. As a result of these analyses, an attempt is made to realize the impact of the benchmarks on the computer system.

2. Background and Motivation

A few decades ago, micro-architects for general-purpose microprocessors, designed microprocessors based on their intuitions and past experiences, and the experiences of others. Since the mid-1980s, microarchitecture had a face-lift by following a more systematic process that proceeded through architectural simulations tools and their results. Though the different tools used varied in the level of details, and hence accuracy, they provided a means to come to more scientific and reproducible conclusions while making micro-architectural decisions. However, due to the intense and increasing complexity of the microarchitecture, the applications that run on them and the hardware it is implemented with, the simulators become very time consuming and there arises a need to reduce the time spent during the simulation process.

In [2], Bhandarkar and Ding characterized the performance of several business and technical benchmarks on the Pentium® Pro processor based system. They were able to show that the Pentium Pro processor achieved significantly lower cycles per instruction that the Pentium® processor due it to its out of order and speculative execution and non-blocking cache and memory system. Using and describing the on-chip hardware performance counters, they analyzed the

performance of the machine by analyzing most of the individual performance characteristics of the processor.

In [4] and [5] Eeckhout et al, proposed the use of principal component analysis and cluster analysis to efficiently explore the workload space, in which the benchmark-input pairs could be plotted to give an idea of the behavioral difference between the benchmark-input pairs. They selected a limited set of representative program-input pairs with small dynamic instruction counts. They were able to substantiate their claims by showing that the program-input pairs that are close to each other in the principal components space, indeed exhibit similar behavior as a function of micro-architectural changes.

Though the paper by Bhandarkar et al., does analyses the performance ratios and the individual performance characteristics, it does not take into account the relation between the benchmarks and the way their behavior. In the work done by Eeckhout et al., the principal component analysis and cluster analysis was performed on the simple scalar simulator. The purpose of this report is to combine the best of both reports and provide a new set of results with a different objective, to understand the microprocessor benchmarks towards understand the workload design space. This report initially uses the on-chip

hardware performance counters to collect the data. The data is used and is initially analyzed using regression analysis to get a better understanding of the benchmarks and the effect of the different performance characteristics on CPI, after which principal component analysis is performed on the data to analyze the behavior of the benchmarks.

3. Methodology

In this section, the methodology followed to characterize the benchmarks, the tools that were used, the different benchmarks that were used and the different analysis carried out on them will be discussed. The methodology can be divided into two portions.

i) Data Collection

ii) Data Analysis

Data Collection was done with the help of in-built hardware performance counters present in the Pentium-III® processor. The SPECint-95 benchmarks were run, while the hardware performance counters collected data in the background. It will be discussed in more detail in the next section.

Data Analysis can be divided into two further portions

i) Regression Analysis

ii) Principal Component Analysis

Each one of the above will be discussed in depth in the following sections.

3.1 Data Collection:

Prior to the study of any data analysis technique is the data collection. A Pentium III processor on a Linux 2.4.7-10 was chosen. The processor was a 1 GHz processor with a 256KB cache size.

Nearly all of the microprocessors that are in the market today internally incorporate on-chip hardware performance monitors, which can be used to understand microprocessor performance while they run complex, real time workloads. Due to this complex systems can be monitored and evaluated very closely. Although they are not considered part of the architecture, microprocessor vendors release information about these hardware performance-monitoring counters.

The Intel Microprocessors contain two performance-monitoring counters. These counters can be read with special instructions on the processor. These counters can be made to measure user and kernel activity in combination or isolation. A variety of performance events can be measured using the counters. Though 200 events can be measured on the processor, only 2 can be measured at the same time. Usually the numbers of events that can be measured at the

same time are limited to 4 or 5 to maintain simplicity, because these events registering also contributes towards increased hardware complexity. If these performance counters are not carefully implemented, they can impact the processor cycle time and power consumed too, which is becoming more prominent as sub-micron levels are exploited.

In [3], Bhandarkar and Ding talk about the hardware performance counters present in the Pentium Pro ® architecture. Two performance counters are implemented. Each performance counter has an associated event select register that selects what is to be monitored. The RDMSR and WRMSR instructions are used internally to access the counters. While a program is running on the machine, various performance metrics can be measured in the background.

There are various tools available to monitor these hardware performance counters. PMON is a software driver used to monitor these counters, which is written by the Laboratory for Computer Architecture at the University of Texas at Austin.

The SPEC-int95 benchmarks were chosen for the sole reason that the time they take to run on a machine is lesser compared to other and latest benchmarks. Among the SPEC suite, the benchmark programs chosen were gcc, perl, li, go. Only the reference input sets were chosen to make it uniform over benchmarks. Pperf is the version of PMON compatible with Linux. A total of 18 performance metrics were calculated. The data obtained was used in the analyses that were carried out. The table below, taken from [3] shows the various events that were used to calculate the performance metrics.

Pentium® Pro Processor Counter based Performance Metrics		
Performance Metric	Numerator Event	Denominator Event
Data References per instruction	DATA_MEM_REFS	INST_RETIRED
L1 Dcache misses per instruction	DCU_LINES_IN	INST_RETIRED
L1 Icache misses per instruction	L2-IFETCH	INST_RETIRED
ITLB misses per instruction	ITLB_MISS	INST_RETIRED
Istalls cycles per instruction	IFU_MEM_STALL	INST_RETIRED
L1 Cache misses per instruction	L2_RQSTS	INST_RETIRED
L2 cache misses per instruction	L2_LINES_IN	INST_RETIRED
L2 miss ratio	L2_LINES_IN	L2_RQSTS
Memory transactions per instruction	BUS_TRANS_MEM	INST_RETIRED
FLOPS per instruction	FLOPS	INST_RETIRED
UOPS per instruction	UOPS_RETIRED	INST_RETIRED
Speculative Execution Factor	INST_DECODED	INST_RETIRED
Branch Frequency	BR_INST_RETIRED	INST_RETIRED
Branch mispredict Ratio	BR_MISS_PRED_RETIRED	BR_INST_RETIRED
Branch Taken Ratio	BR_TAKEN_RETIRED	BR_INST_RETIRED
BTB miss ratio	BTB_MISSES	BR_INST_DECODED
Branch Speculation Factor	BR_INST_DECODED	BR_INST_RETIRED
Resource Stalls per instruction	RESOURCE_STALLS	INST_RETIRED
Cycles per instruction	CPU_CLK_UNHALTED	INST_RETIRED

3.2 Data Analysis:

As mentioned earlier, data Analysis can be divided into two different parts:

1. Regression Analysis
2. Principal Component Analysis

3.2.1 Regression Analysis:

Measuring the performance of a computer system for all possible values would allow us to have a clear understanding of a system's performance under any possible condition. However, that would be prohibitively expensive, to undertake such an endeavor. Instead, the system's performance for a limited number of inputs can be measured and these measured values can be used to produce a mathematical model to describe the behavior of the system over a range of values. This model that obtained will be called a Regression Model, and will be used to predict how the system will perform when given an input value that was not actually measure in the first place.

The formation of this linear regression model begins with a system that has one continuous input factor whose value can be controlled. If the measured response values were plotted as a function of the input values, it is found that a linear relationship appears between the input and the output. The least-squares minimization is then used to produce a linear regression equation to model the system. This will help predict as to how the system could react to a response that was not measured before.

3.2.1.a. Least Squares minimization:

A simple linear regression model is of the form

$$Y = a + bx,$$

Where x is the input variable, y is the predicted output response, and 'a' and 'b' are the regression parameters that should be estimated from the set of measurements. If $y_{i,}$ is the value actually measured when the input is x_i, then each of these variable can be written as,

$$y_i = a + bx_i + e_{i,}$$

where e_i is the difference between the measure value for y_i, and the value that would have been predicted for y_i, from the regression model.

The analysis can be complete after computing 'a' and 'b'. This 'a' and 'b' will form a line that most closely fits n measured data points. At the same time there is a need to minimize the sum of squares of these residuals, denoted by SSE. And shown below as:

$$\text{SSE} = \sum_{i=1}^{n} ei2 = \sum_{i=1}^{n} (yi - a - bxi)2$$

3.2.1. b. Correlation:

After the linear regression model is developed, it is needed to know how well the equation that was formed, actually models the measured data. In other words, it is necessary to know how the strength of the linear correlation between the input and output is. The coefficient of determination and its square root, called the correlation coefficient help us determine the correlation of this linearity.

The fraction of the total variation explained by the regression model is called the coefficient of determination. In the case where a perfect linear relationship exists between the input and output, all of the variations in the measured outputs will explained by the regression model. In this case, all of the measured data points would fall directly on the regression line so that all of the residuals, i.e. e_i are zero, giving SSE =0. Then, the coefficient of determination r^2=1. On the contrary, if none of the variations is explained by the regression model, then $r^2 = 0$.

Thus, the coefficient of determination provides an indication of how well the regression model fits the data. Values of $r^2 = 1$, signify a close linear relationship between the input and output values. Of course on the flip side, r^2=0 signify that there is little or absolutely no relationship between the input and output values. This would occur if there was a horizontal line, so that knowing any of the input values would be of absolutely no help in predicting the output values. Also, if there existed a functional relationship between the inputs and outputs, which however was not linear, then r^2 would be near zero.

The square root of the coefficient of determination is called the correlation coefficient, denoted by r. The value of r ranges from -1 to +1. A value of +1 signifies a perfect positive linear correlation between the input and output values. In this case, an increase in the magnitude of the input will reflect scaled appropriately in the output. Conversely, a value of r = -1 means that any change in the input will produce an appropriate change in the output, however in the reverse direction. In simple words, an increase in the input will cause a decrease in the output. Values of +1 and -1 indicate different levels of correlation.

It is important to understand the difference between the coefficient of determination and the correlation coefficient. Consider two systems, for which linear regression models have been developed. If for the first system, the correlation coefficient was calculated to be 0.5 and for the other system if it was -0.91, it cannot concluded that the linear relationship for the second system is better than for the first. All that can be concluded is that the linear relationship in the second systems appears stronger.

Analyzing this further, if the coefficient of determination for the first system is 0.25 and the second system around 0.81, only 25% of the variation is explained by the regression model for the first system and 81% for the second system.

Thus it can be concluded that the linear relationship between the inputs and outputs for the second system is much stronger when compared to the first system.

It is also worth the fact to mention the difference between correlation and causation. Causation means that an observed change in the output is the direct result of a change in the input. That is there is some process within the system that somehow links the input and output, which implies that if the process were linear, a larger coefficient of determination can be expected. A famous example is that reading a large file takes longer than reading a small file because more data needs to be transferred from the disk to the memory, thus existing a high correlation between the file reading time and the number of bytes read.

The converse however does not hold true. The output, could he highly correlated to the input without the input causing the output.

On the same lines, multiple input regressions also exist, which is just an extension of the one input regression. This helps in including the effects of several input variables linearly related to a single output variable.

3.2.2 Principal Component Analysis:

Principal component analysis is a statistical analysis techniques used mainly in situations where there are numerous inputs and analysis of all the data could be complicated. It linearly transforms an original set of variables into a substantially smaller set of uncorrelated variables called Principal Components, which represent most of the information in the original set.

Thus as mentioned earlier, the goal is reduce the dimensionality of the original data set. In general PCA helps us understand the structure of a multivariate data set.

The p original variables X_i, i = 1 to p are linearly transformed into p principal components Z_i, i = 1 to p. The principal components are constructed such that Z1 has maximum variance and Z2 is chosen such that it has maximum variance under the constraint that it is not correlated to Z1. The same procedure is followed to form the other principal components. Automatically, the principal components are arranged in decreasing variance and are uncorrelated, that is the covariance between one principal component and another is equal to zero,

where Covariance is a measure of how much the deviations of two variables match.

Thus the dimensionality of the data set is brought down by keeping only those principal components which have the highest variance. The number of retained principal components can be traded-off against the fraction of the variance in the data set that is explained. In other words, it is up to the user of the PCA to decide how many principal components are to be retained.

First of all, before applying PCA, it is advisable to standardize the variable, that is, rescale them such that they have zero mean and unit variance. Otherwise, the variable with higher variance will have higher impact in the first principal components.

The idea behind principal component analysis is that it is easier to understand differences between benchmarks when there are only q = 5 (for example) types of possible differences when the benchmarks may differ in p = 50 difference ways. When the user feels q is sufficiently small, the reduced space can be visualized with a scatter plot, showing the position of each benchmark with respect to the principal components. The eccentricity of the benchmarks with respect to the analyzed benchmark suite determines their position on the scatter

plot. Benchmarks that are close to the origin of the q-dimensional space are average benchmarks, that is, when one of the parameters is changed, the benchmark will also see a change similar to the average over the entire suite. Benchmarks that are far from the origin either see an unusually large or small effect when one of the parameters is changed.

Only a few parameters play an important role in each principal component. This can be determined from the factor loadings. The factor loadings are the coefficients a_{ij} in the linear combination, $Z_i = \sum_{j=1}^{p} aijXj$. The larger a_{ij} is in magnitude, the stronger it influences the principal component, while the closer it is to zero, the lesser or nil impact it has on the principal component. Thus the benchmark with large values on X_{ij}, will score positively on Zi when aij is positive, while those that have small values for Xj will score negatively.

When the value for a variable Xj is large for a benchmark, then that benchmark is insensitive to the parameter that is changed in that variable. The benchmark will have a large and positive value for Zi, assuming that the factor loading on aij is large and positive. On the contrary, Zi will be negative when the benchmark is very sensitive to the parameter change in Xj, when aij is still large and positive. Thus, when the factor loading aij is positive, the benchmarks that

are sensitive to the parameter changed in Xj have a negative value for Zi. Those that are not sensitive have a positive value for Zi.

Principal components analysis can be used to judge the impact of an input on a program also. The inputs usually have a small impact when their workload characteristics are not that different. Consequently, these program-input points will be near each other in terms of original p-dimensional space and also in the q-dimensional space of principal components. It is possible to find clusters of benchmarks, (groups of benchmarks that are internally close, but externally distant to other clusters). It can be said that the input has little effect on the behavior of the program if all the instances of the same program run on different inputs in the same cluster.

Hence, benchmark suites can be compared more easily in the reduced space of the principal components. The benchmark suites occupy disjoint areas in the space of principal components. In reality, it can be expected that the benchmarks to overlap, thereby having a few benchmarks exhibiting similar behavior. When a region of space contains only benchmarks from one suite, then those benchmarks contain characteristics that are not present in the other suite.

4. Results

The results will be discussed in the following sections:

1. Regression Analysis Results
2. Principal Component Analysis Results

4.1 Regression Analysis Results:

As mentioned earlier, a Pentium III processor on a Linux 2.4.7-10 was the chosen system of operation with SPECint-95 benchmarks, the benchmarks in consideration installed.

PPerf was the utility compatible with unix, which was nothing but a software driver able to access the hardware performance counters. Also mentioned earlier, is the ability of PPerf to access only two counters simultaneously.

For each benchmark and its input value, PPerf measured the data from every counter and subsequently the data was recorded. Data was obtained from the hardware performance counters from which 18 different performance metrics were calculated. After this was done, single-input regression analysis was

performed on the data and multiple-input regression was performed for a few metrics. Microsoft Excel® was used for the regression analysis.

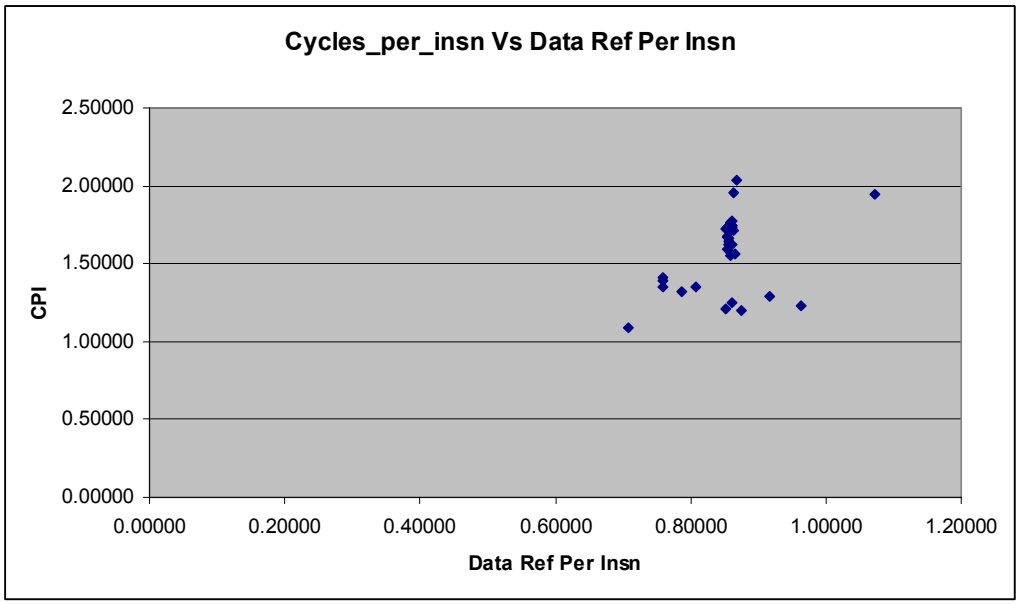
All of the 18 performance metrics will be plotted against the Cycle per instructions (CPI). Thus the dependence of the CPI on all the other 18 performance metrics will be discussed below. A high correlation factor would imply a high dependence of CPI on the performance metric.

4.1.1 CPI Vs Data References Per Instruction

Table 2. CPI Vs Data References Per Instruction

Equation : y=a+bx	a	b	r
CPI Vs data_ref_per_insn	0.234164216	1.573628565	0.401497719

Graph 1. CPI Vs Data References Per Instruction



As can be seen from the graph, CPI does not seem to be correlated too well with the Data References Per Instruction. This is further confirmed by the regression results shown.

This could be due to the fact that data references are a combination of hits and misses in the L1 cache, L2 cache and also misses in the L2 cache which would mean that the main memory is accessed. If the main memory was accessed too often then the CPI would have a chance to be affected more by the data

references per instruction, since it would contribute to more cycles being spent for that particular instruction and if the data references consisted of a large percentage of unique memory addesses then CPI could have depended more on data references per instruction, which is not that case as shown in the graph.

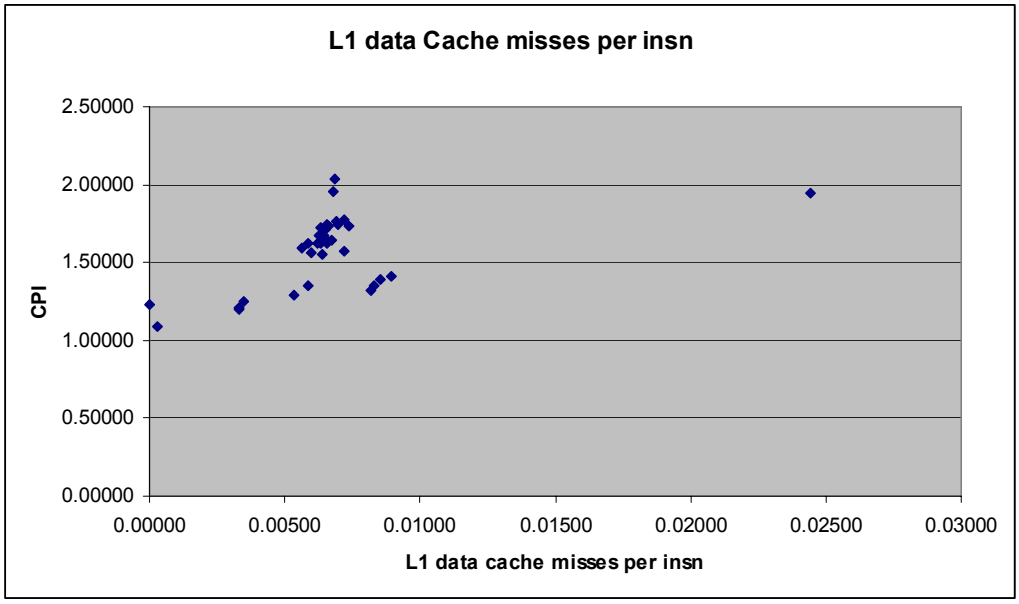
It is also found that the same benchmarks with different inputs seem to affect the CPI in the same way, thus not showing a clear distribution. Thus a benchmark mixed with memory intensive sections and non-memory intensive sections would have stressed it uniformly.

4.1.2 CPI Vs L1 Data Cache Misses Per Insn:

Table.3. CPI Vs L1 Data Cache Misses Per Instruction

Equation : y=a+bx	a	b	r
L1_data_cache_misses_per_insn	1.36316598	1.36316598	0.510789062

Graph 2. CPI Vs L1 Data Cache Misses per Instruction



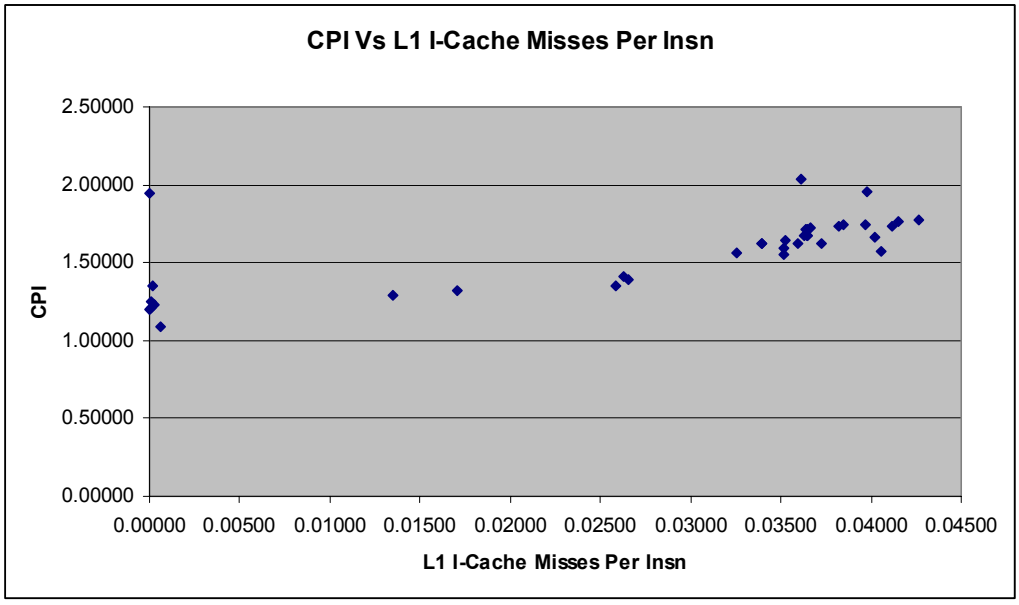
The argument for Result 2 is very similar to that of the first result. The CPI does not have a high correlation with the L1 data Cache miss. However, it does have a marginally higher correlation coefficient. Though it does not give a clear idea it gives an indication that the total number of memory transactions which are also part of the L1 cache misses are a greater portion of the L1 Misses than the Data References Per Instruction. If there was a benchmark which had more transactions to memory, that is, a more memory intensive benchmark, then it would have more correlation with the CPI. It is also found that there is clustering of inputs for the same type of benchmark showing that the same benchmark has the same effect on CPI irrespective of the input.

4.1.3 CPI Vs L1 instruction cache misses Per Instruction:

Table 4. CPI Vs L1 Instruction Cache Misses Per Instruction

Equation : y=a+bx	a	b	r
L1_I_cache_misses_per_insn	1.274649485	10.81724493	0.714008245

Graph 3. CPI Vs L1 Instruction Cache Misses per Instruction



Though the L1 I-cache misses have a greater correlation that the L1 data cache misses, this still cannot definitively confirm that it has a high correlation. This

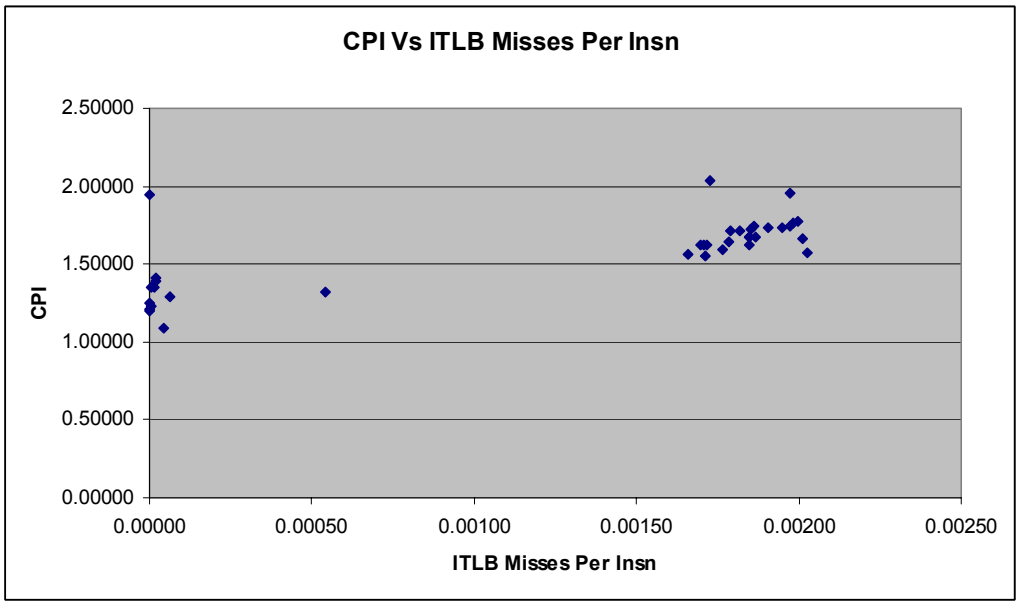
could mean that the instruction cache could not have as many levels as the data cache and a miss in the L1 Cache would need a access to memory, however not significant enough to cause enough stalls to have a great enough effect on CPI. Also it can be seen that different inputs for the same benchmark have varied effects on the CPI, which is closer to what is preferable.

4.1.4 CPI Vs ITLB Misses Per Instruction:

Table.5. CPI Vs ITLB Misses Per Instruction

Equation : y=a+bx	a	b	r
ITLB_misses_per_insn	1.326095446	1.326095446	0.755605461

Graph 4. CPI Vs ITLB Misses Per Instruction



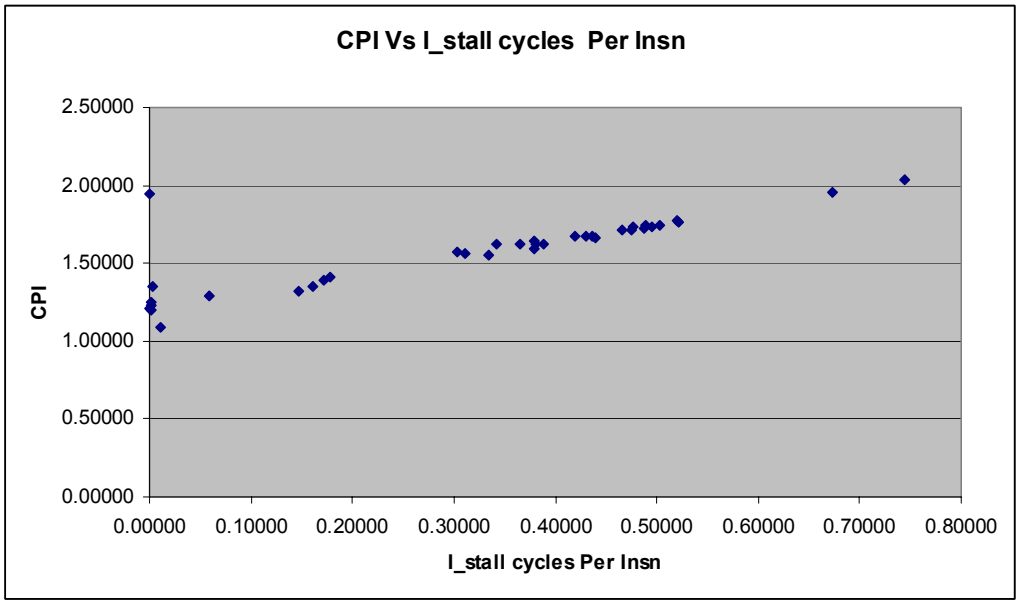
The ITLB Misses do not fall onto a line of best fit as ideally desired. The regression coefficient too is not high enough to mention a high correlation with CPI. It is also seen that the same benchmarks with different input sets seem to have nearly the same effect on CPI thereby bringing all the points into a cluster.

4.1.5 CPI Vs Instruction Stall Cycles Per Instruction:

Table 6. CPI Vs Instruction Stall Cycles Per Instruction

Equation : y=a+bx	a	b	r
I_stalls_cycles_per_insn	1.281516406	0.926023062	0.840292315

Graph 5. CPI Vs Instruction Stall Cycles per Instruction



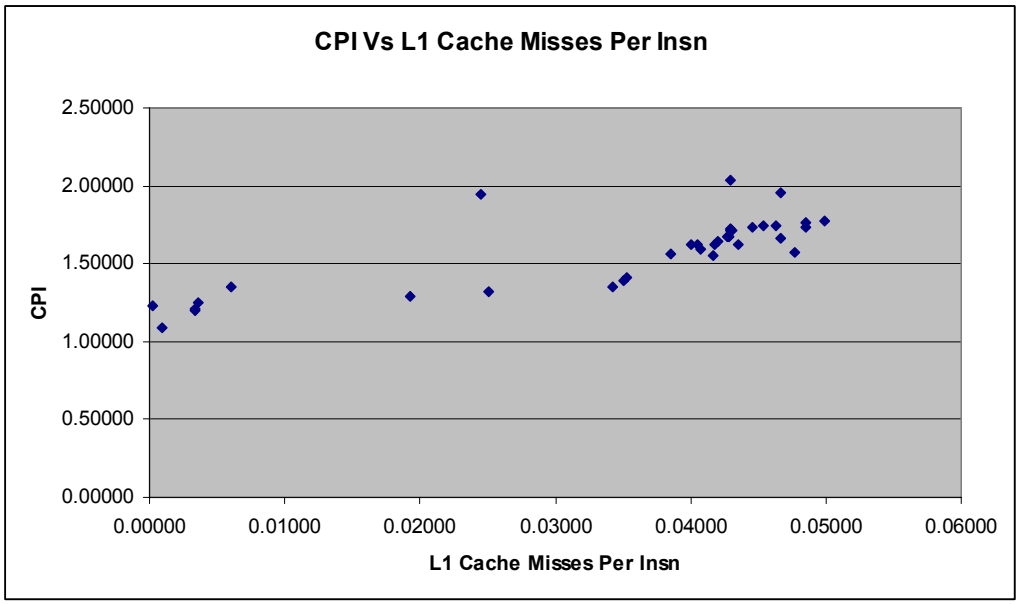
The Instruction stall cycles per instruction have a higher correlation coefficient when compared to many other performance metrics. This could be due to the fact that Instruction stalls could be a result of the most of the stages ahead of it in the pipeline stalling, since the Instruction fetch occurs at the start of the pipeline. It can be seen that a better line of best fit is obtained compared to some of the other performance metrics that seen earlier. Hence, the same benchmark with different inputs stresses the system differently, resulting in different CPIs.

4.1.6. CPI Vs L1 Cache Misses Per Instruction:

Table 7. CPI Vs L1 Cache Misses Per Instruction

Equation : y=a+bx	a	b	r
L1_cache_misses_per_insn	1.176480636	11.59879379	0.799498704

Graph 6. CPI Vs L1 Cache Misses Per Instruction



This L1 cache misses are nothing but a combination of the L1 data and L1 instruction cache misses. It can be inferred that, though both of them did not have a high correlation with CPI, I-cache misses had a line, which fit better than the D-Cache graph. At first glance of the graph above, it could be said that there seems to be a nice line to which the data falls into. But looking at the correlation coefficient, it is only 0.8, which is not enough to say that L1 cache misses have a high correlation with CPI. One interesting point to note is that within a particular benchmark, in this case gcc, there seems to be good fit into a

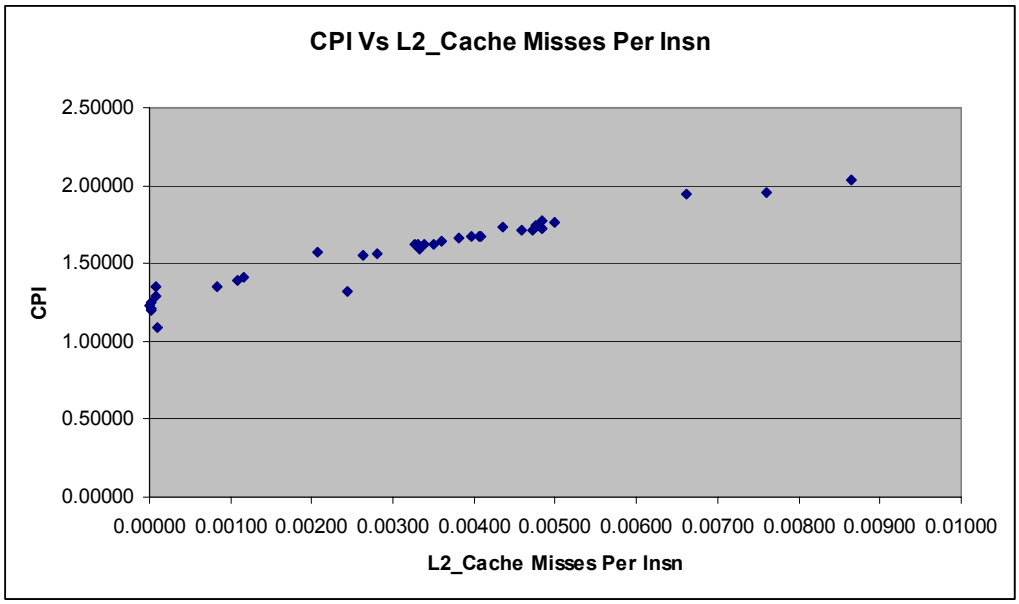
line. This could be due to the fact that the gcc program might be more memory intensive thereby a L1 cache miss would have subsequently been a L2 cache miss too thereby indicating the existence of memory traffic.

4.1.7 CPI Vs L2 Cache Misses per Instruction:

Table 8. CPI Vs L2 Cache Misses Per Instruction

Equation : y=a+bx	a	b	r
L2_cache_misses_per_insn	1.256416465	100.27	0.967588459

Graph 7. CPI Vs L2 Cache Misses Per Instruction



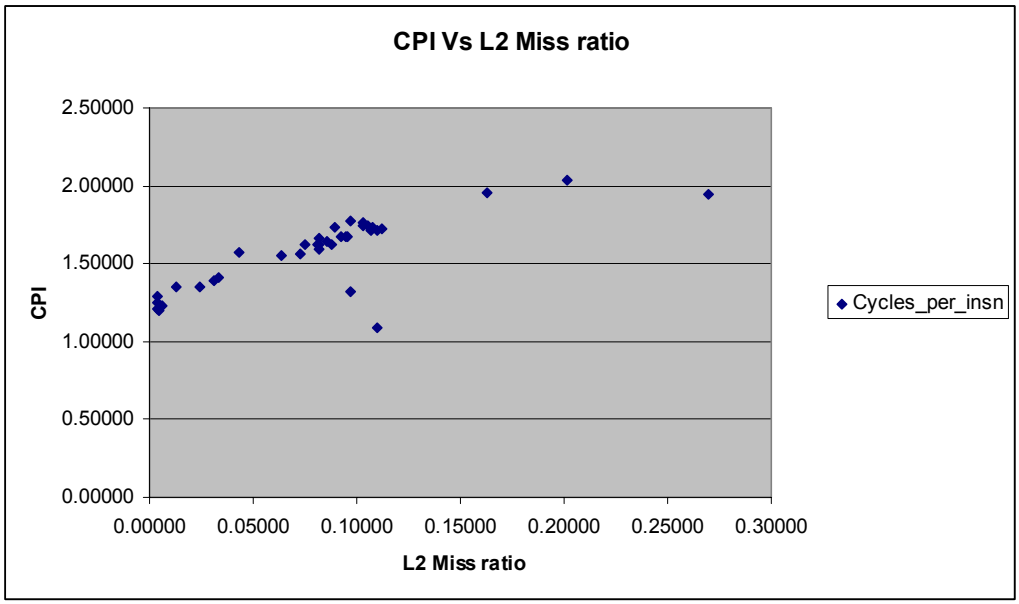
This result is very interesting. From first glance not only can it be seen that the data falls into a line of very good fit, but the correlation coefficient too is high, indicating that the CPI depends a lot on the L2 cache Misses Per Instruction. This could be attributed the one main reason. The fact that there are only two levels of hierarchy, means that whenever there is a miss in the L1 cache which misses in the L2 cache too, this is going to generate a reference to the main memory which is not preferred since this usually takes significantly more clock cycles when compared to a cache hit. These extra cycles add up to the CPI. Hence the high correlation between the CPI and L2 cache misses per instruction.

4.1.8 CPI Vs L2 Miss Ratio:

Table 9. CPI Vs L2 Miss Ratio

Equation : y=a+bx	a	b	r
L2_miss_ratio	1.309744765	3.278480091	0.792151428

Graph 8. CPI Vs L2 Miss Ratio



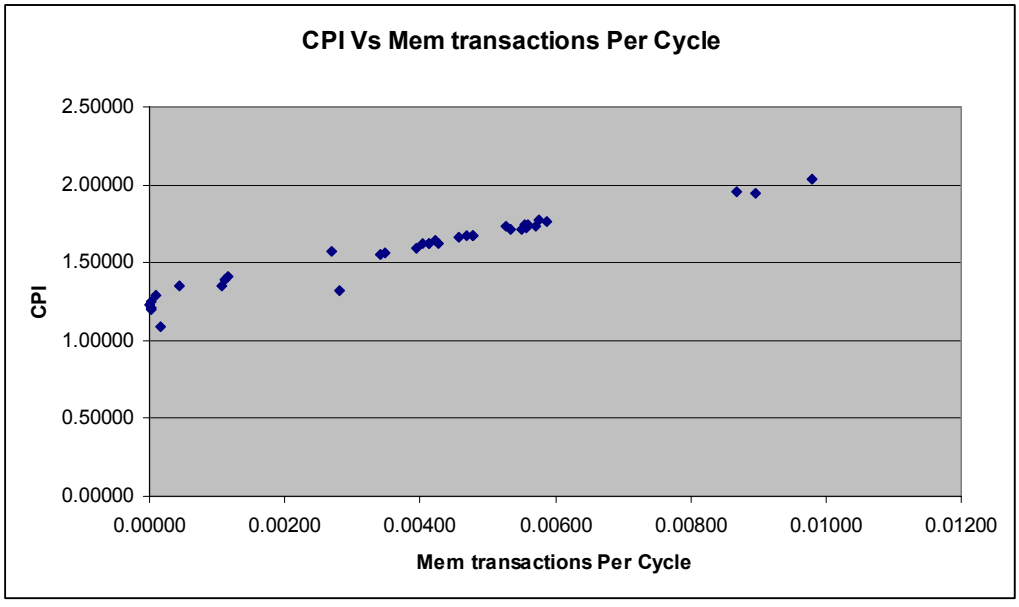
Even though the L2 misses had a high correlation, the L2 miss ratio does not seem to have a high correlation. One possibility could be that, the number of misses was not significant enough to impact the CPI. Also, it can be seen that certain benchmarks and their input sets seem to form a line of good fit. So it could also be that certain outliers, possibly could have been detrimental to a high correlation with CPI.

4.1.9 CPI Vs Memory Transactions per Cycle:

Table 10. CPI Vs Memory Transaction Per Instruction

Equation : y=a+bx	a	b	r
Mem_trans_per_insn	1.250804631	85.36869916	0.972288404

Graph 9. CPI Vs Memory Transaction Per Instruction



The Memory transactions per cycle is nothing but an extension of the many of the above discussed performance metrics. The L2 cache misses contribute to

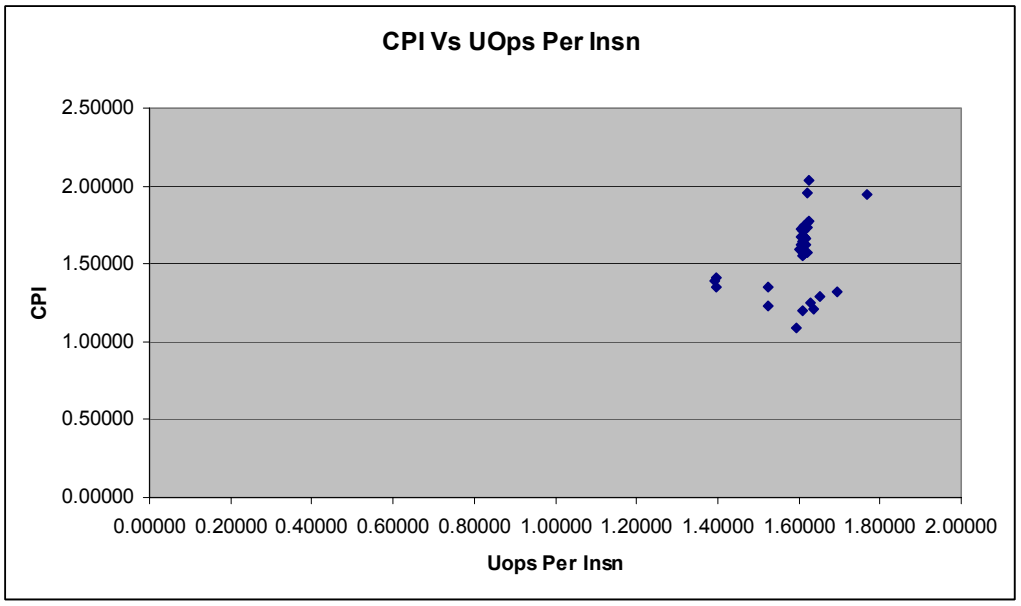
memory transactions per cycle and memory transactions are expensive on CPI. Hence, t\he 0.97 correlation coefficient, indicating a high correlation coefficient between CPI and Memory Transactions per Cycle.

4.1.10 CPI Vs Micro-operations per instruction

Table 11. CPI Vs Micro-operations Per Instruction

Equation : y=a+bx	a	b	r
UOPS_per_insn	-0.238805567	1.136245489	0.360553351

Graph 10. CPI Vs Micro-operations Per Instruction



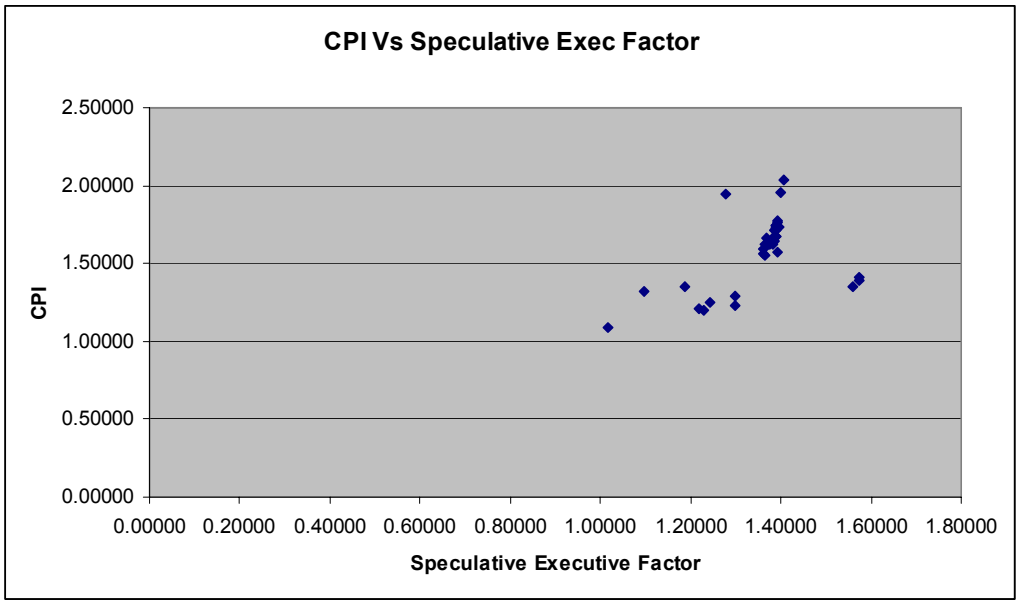
The Micro-operations per instruction do not have a high correlation with CPI. This can be seen from the fact that there hardly seems to be a line of best fit. This is kind of intuitive, since for the same program, irrespective of the inputs applied, the number of micro-operations seems to be the same. The point lying outside mostly come from non-gcc points.

4.1.11 CPI Vs Speculative Execution Factor:

Table 12. CPI Vs Speculative Execution Factor

Equation : y=a+bx	a	b	r
Speculative_exec_factor	0.329871221	0.920393757	0.444809942

Graph 11. CPI Vs Speculative Execution Factor



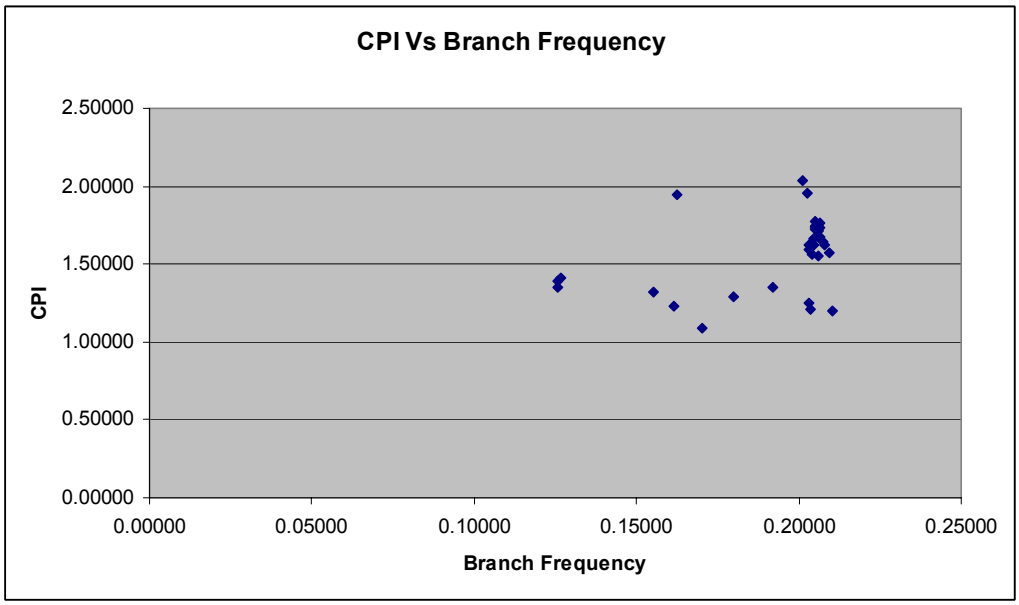
The Speculative execution factor is a ratio of the total number of instructions decoded to the total number of instructions retired. The fact that many points seem to be clustered together could mean that for a particular benchmark (most of those point belong to the gcc benchmark), the change in inputs bring about only a small change in the Speculative Execution factor and hence not much correlation with the CPI.

4.1.12 CPI Vs Branch Frequency:

Table 13. CPI Vs Branch Frequency

Equation : y=a+bx	a	b	r
Branch_frequency	0.838513361	3.829141845	0.416675437

Table 12. CPI Vs Branch Frequency



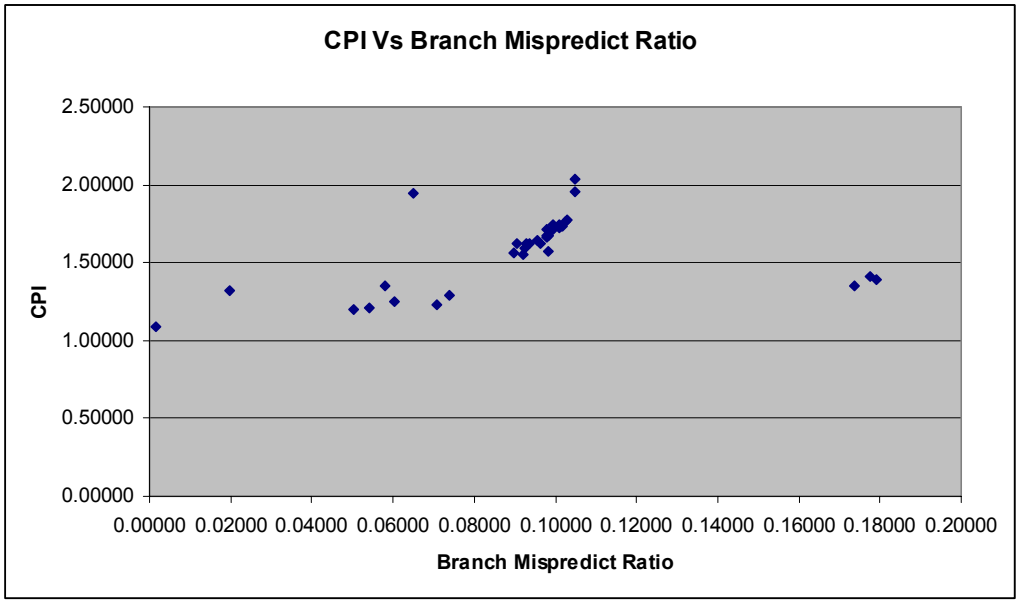
The branch frequency is a measure of the occurrence of branch instructions among the total instructions. It is a ratio of the total branch instructions to the total number of instructions retired. The number of branches taken would have had more significance to the CPI if most of them caused a branch mispredict, thereby stalling the pipeline for a number of cycles. But this was not the case observed as the correlation coefficient is not a significant number.

4.1.13 CPI Vs Branch Mispredict ratio:

Table 14. CPI Vs Branch Mispredict Ratio

Equation : y=a+bx	a	b	r
Branch_mispred_ratio	1.382971731	2.097102083	0.320703921

Graph 13. CPI Vs Branch Mispredict Ratio



The microprocessors nowadays have very efficient branch predictors which predict a branch to a very high level of accuracy. From the graph and table

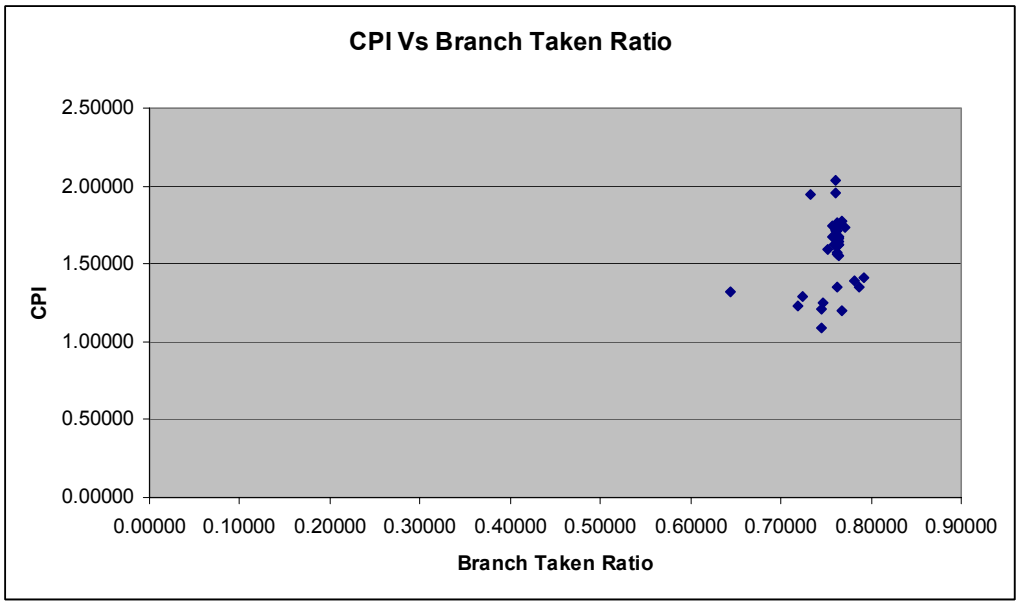
above it can be seen that the dependence of CPI on the branch mispredict ratio is very low. Like mentioned earlier, this could be a result of there being very few branch mispredictions compared to the total number of branch instructions.

4.1.14 CPI Vs Branch Taken Ratio:

Table 15. CPI Vs Branch Taken Ratio

Equation : y=a+bx	a	b	r
Branch_taken_ratio	-0.309135784	2.492986457	0.259514779

Graph 14. CPI Vs Branch Taken Ratio



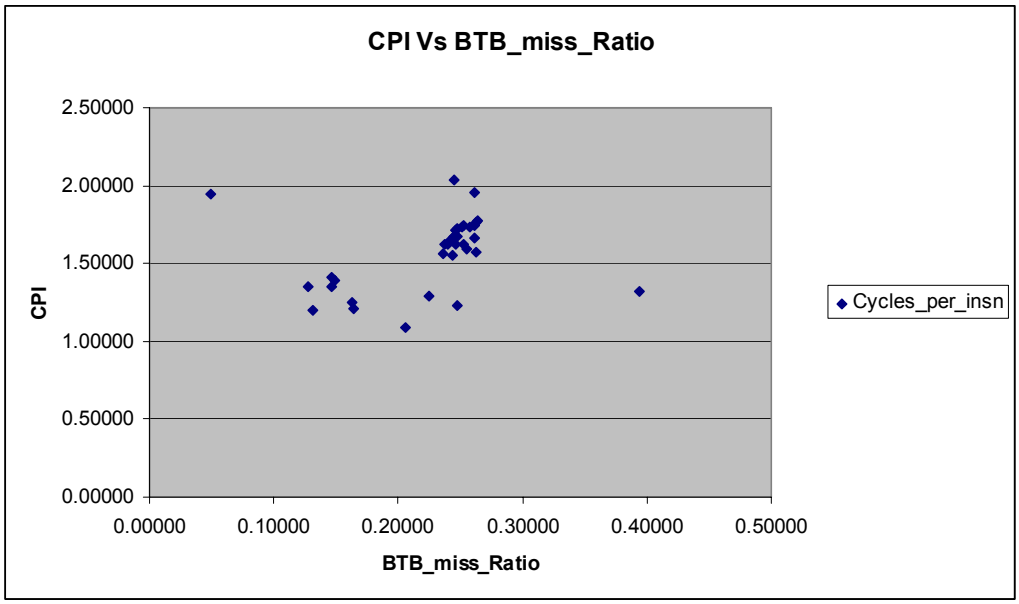
The branch taken ratio as expected hardly had any correlation with CPI. This is because that these taken branches which were predicted could have hardly ended up in mispredictions and pipeline stalling. The only way this could have had any significance on the CPI is if the Taken branches were a result of mispredictions.

4.1.15 CPI Vs BTB miss ratio:

Table 16. CPI Vs BTB Miss Ratio

Equation : y=a+bx	a	b	r
BTB_miss_ratio	1.354687829	0.981848193	0.252081632

Graph 15. CPI Vs BTB Miss Ratio



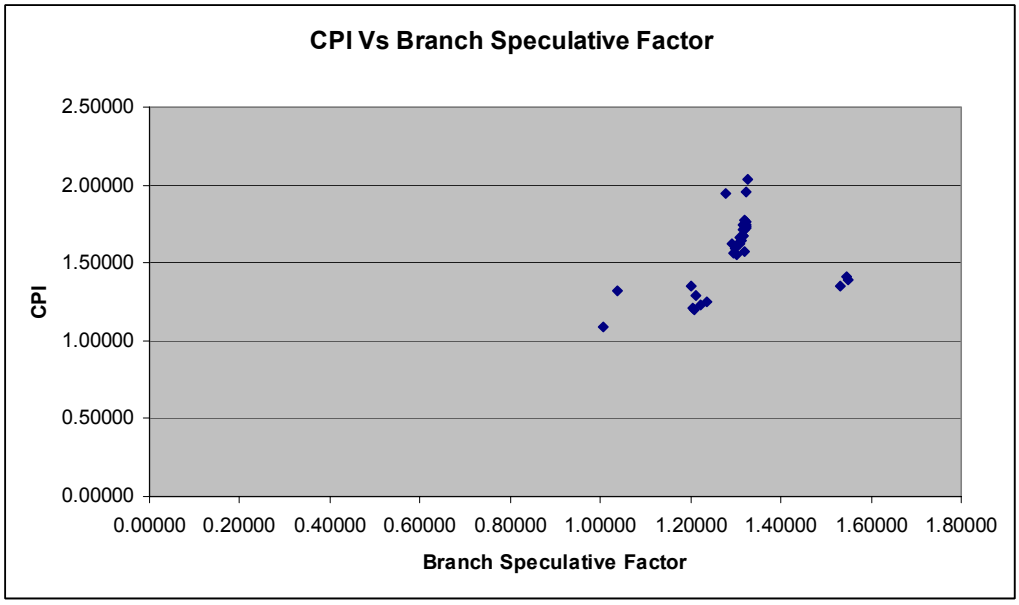
The Branch target buffer miss ratio had any significance on the CPI and this is actually somewhat an expect behavior.

4.1.16 CPI Vs Branch Speculative Factor:

Table 17. CPI Vs Branch Speculative Factor

Equation : y=a+bx	a	b	r
Branch_spec_factor	0.606962806	0.747077822	0.341327378

Graph 16. CPI Vs Branch Speculative Factor



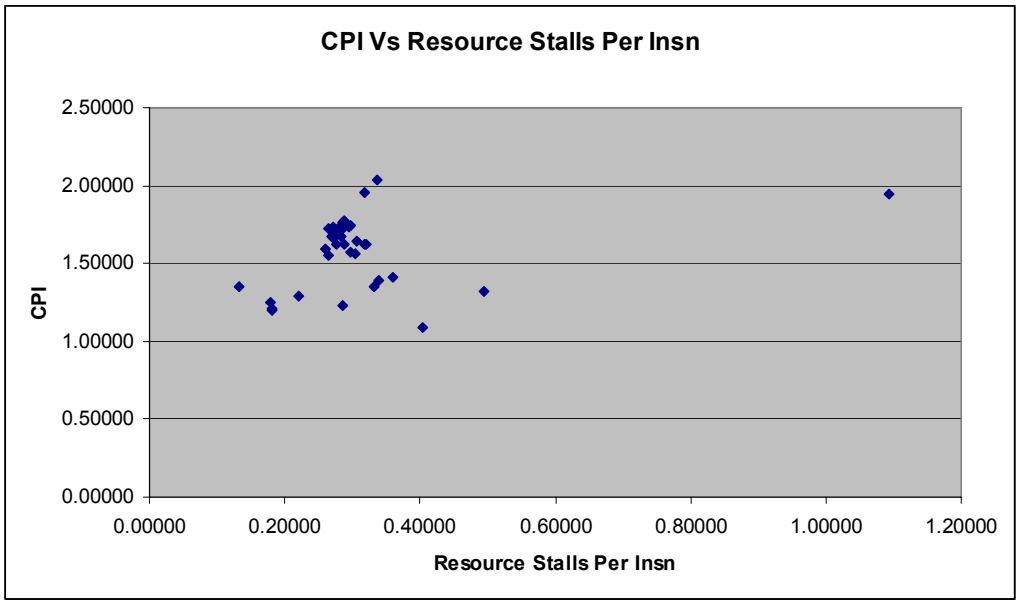
The branch speculative factor is nothing but the ratio of total branch instructions decoded and the total branch instructions retired. From the graph and table it can be clearly deduced that it CPI has a very low correlation on the branch speculative factor.

4.1.17 CPI Vs Resource Stalls Per instruction:

Table 18. CPI Vs Resource Stalls Per Instruction

Equation : y=a+bx	a	b	r
Resource_stalls_per_insn	1.435559721	0.458279559	0.296454012

Graph 17. CPI Vs Resource Stalls Per Instruction



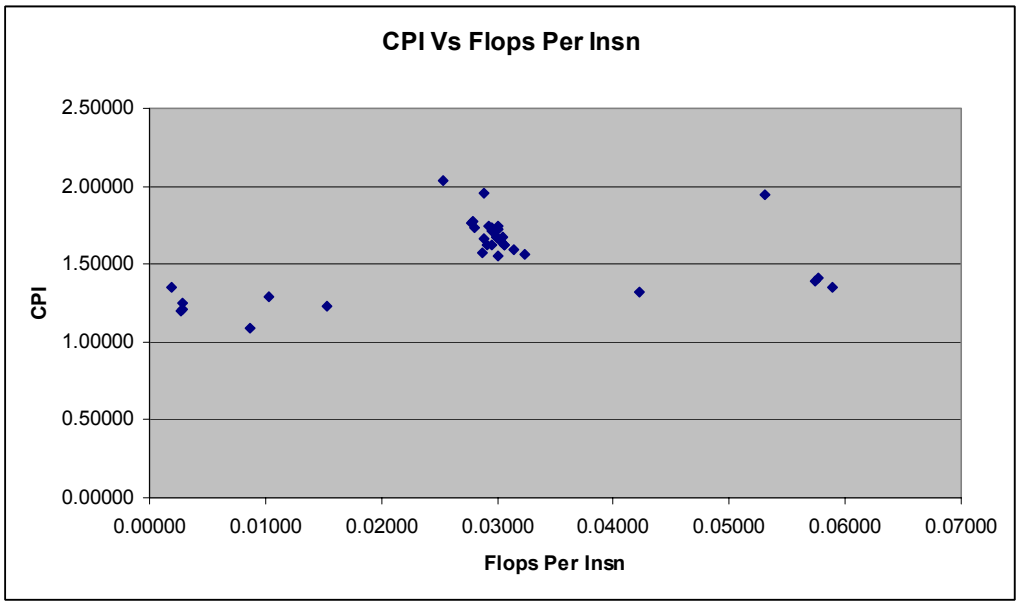
This is probably one of the surprising results obtained. Usually a resource stall should cause an increase in the CPI. One possible explanation for this value could be that there could be multiple resources all operating in parallel and even though one resource stalled due to a dependency, for example, the other resources could have been able to continue executing. However, the cycles it stalled for could have been included in the counter.

4.1.18 CPI Vs Flops per instruction:

Graph 19. CPI Vs Flops per instruction

Equation : y=a+bx	a	b	r
Flops_per_insn	1.409575225	5.907193981	0.365794466

Graph 18. CPI Vs Flops Per Instructions



The benchmark used was a SPEC int benchmark and hence the low correlation coefficient of CPI with the Flops Per instruction. The number of flop instructions were low and not significant enough to have an impact on the CPI. However, if there were extensive flop calculations, then you could expect it to have a higher impact on the CPI, that is assuming that the flop instructions would have been taxing on the resources.

4.2 Principal Component Analysis Results:

As mentioned in detail earlier, Principal component analysis is a statistical technique used to reduce the dimensionality of the original set. Ideally it would be ideal, to end up into a set of uncorrelated variables called the Principal components from our original set of variables, which represent most of the information of the original set. It would be preferable to retain only those principal components that have the highest variance.

The 19 performance metrics were initially obtained as mentioned during the Regression analysis. After data acquisition, the data was normalized. For normalization, initially all the data for a performance metric was taken and its mean was calculated. After calculating the mean, the standard deviation was calculated. The normalized data was obtained from :

$$NormalizedData = \frac{Value - MeanOfMetric}{S\tan dardDeviation}$$

The normalized data which can be considered as a huge matrix (19 x 19) in our case becomes the input matrix for our PCA analysis. The analysis was done by a MATLAB program, which will analyze the data for its principal components. The table below shows the mean and the standard deviation of all the performance metrics.

Table 20. Mean and Standard Deviation of all performance metrics – Principal component Analysis

Performance Metric	Mean	Standard Deviation
data_ref_per_insn	0.853467746	0.058510186
L1_data_cache_misses_per_insn	0.006581106	0.003601615
L1_I_cache_misses_per_insn	0.027969779	0.015136909
ITLB_misses_per_insn	0.001251023	0.00086327
I_stalls_cycles_per_insn	0.319310659	0.208093835
L1_cache_misses_per_insn	0.034548834	0.015807222
L2_cache_misses_per_insn	0.003199252	0.002212943
L2_miss_ratio	0.081580692	0.055409762
Mem_trans_per_insn	0.003823425	0.002611843
UOPS_per_insn	1.598255856	0.072769265
Speculative_exec_factor	1.355218035	0.110828498
Branch_frequency	0.192913219	0.024954396
Branch_mispred_ratio	0.092620054	0.035069964
Branch_taken_ratio	0.756659234	0.02387222

BTB_miss_ratio	0.22663138	0.058877245
Branch_spec_factor	1.298716955	0.104774574
Resource_stalls_per_insn	0.309081468	0.148346557
Cycles_per_insn	1.57720544	0.22932459
flops_per_insn	0.0283773	0.014200594

From these values and using the formula above, the data was normalized. This data was used for PCA analysis and through the analysis, the principal components were identified. During these calculations, the eigen values and fraction of variance contained in the first q principal components will be obtained.

19 principal components were obtained but only the significant ones are shown below. For simplicity of explanation, only 3 principal components will be shown the analysis.

Table 21 shows the Eigen values or variance and the eigen vectors or factor loadings on the 3 principal components, PC1, PC2, PC3.

Table 21. Factor loadings on the first three principal components

Performance Characteristic	PC1	PC2	PC3
data_ref_per_insn	0.071	**-0.224**	**0.217**
L1_data_cache_misses_per_insn	**0.152**	-0.018	**0.450**
L1_I_cache_misses_per_insn	**0.312**	0.061	**-0.208**
ITLB_misses_per_insn	**0.292**	-0.108	**-0.252**
I_stalls_cycles_per_insn	**0.320**	-0.041	**-0.198**
L1_cache_misses_per_insn	**0.333**	0.055	-0.097
L2_cache_misses_per_insn	**0.318**	**-0.153**	0.068
L2_miss_ratio	**0.235**	**-0.215**	**0.246**
Mem_trans_per_insn	**0.316**	**-0.164**	0.098
UOPS_per_insn	0.058	**-0.428**	0.048
Speculative_exec_factor	**0.225**	**0.334**	0.005
Branch_frequency	0.116	**-0.262**	**-0.306**
Branch_mispred_ratio	**0.185**	**0.383**	0.030
Branch_taken_ratio	0.131	**0.287**	-0.099
BTB_miss_ratio	0.124	**-0.165**	**-0.307**
Branch_spec_factor	**0.183**	**0.374**	0.068
Resource_stalls_per_insn	0.067	-0.112	**0.489**
Cycles_per_insn	**0.335**	-0.103	0.063
flops_per_insn	**0.189**	**0.224**	**0.268**

The table 6 below show the amount of variance explained by the 3 important principal components.

Table 22. Variance explained by the first three principal components

Property	PC1	PC2	PC3
Eigen Value	7.742	4.622	3.378
% Variance	41.914	25.020	18.289
Cumulative %	41.914	66.933	85.223

The Eigen value of a principal component is the amount of variance it captures. Hence, the first principal component has a larger percentage variance than any of the other principal components. This corresponds to ~42% of the total variance. Thus with three principal components ~85% of the variance present in the 19 original performance metrics, can be explained. As the principal components progress (that is, PC4, PC5 and so on), they become harder to interpret since they progressively contain lesser information.

The factor loadings depict the performance metrics that a particular principal component can measure. The first principal component loads all performance

metrics positively, but by using a threshold of 0.15, it can be said that PC1 depends upon the following performance metrics:

a) L1 Data Cache Misses Per instruction
b) L1 Instruction Cache Misses per instruction
c) ITLB misses per instruction
d) Instruction stall cycles per instruction
e) L1 cache misses per instruction
f) L2 cache misses per instruction
g) L2 miss ratio
h) Memory transactions per Instruction
i) Speculative Execution factor
j) Branch misprediction ratio
k) Branch Speculative factor
l) Cycles per instruction
m) Flops per Instruction

This means that PC1 does not depend on a few or one particular performance metric. It is a combination of many performance metrics and it seems tough to uniquely identify by separating out as to what it exactly depends upon.

Similarly, it can be seen that even PC2 and PC3 depend on many other performance characteristics and hence, it cannot be said that the principal component depends upon a particular performance metric.

Since the factor loadings on PC2 contain both negative and positive numbers, it can be said that PC2 is a contrast between the performance metrics with positive factor loadings and performance metrics with negative factor loadings. What makes it more difficult to analyze is the fact that the factor loadings are very near each other in magnitude.

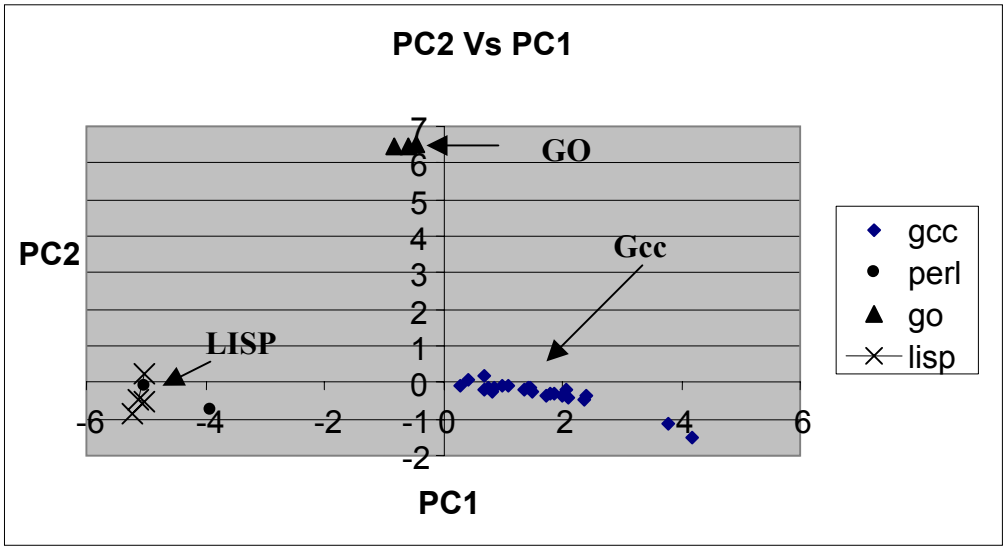
After obtaining the PCA output, the scores matrix can be calculated, which is a relationship with the benchmark-input sets and the influence they have the principal components. This is obtained by multiplying the PCA output matrix by the PCA input matrix. From there they can be plotted in scatter plots.

PCA also allows us to judge the impact of the input on a program too. The inputs have a small impact when their workloads do not differ much. As a result, these program-input pairs end up being close to each other in terms of the original dimension space with all the variables. They will also be near each other in the reduce dimension space. As will be shown below, it can be found

benchmarks which are close to each other but are far apart from other benchmark clusters. When the same benchmark with different inputs ends up in the same cluster, then it can be said that the input has very little effect on the behavior of the program.

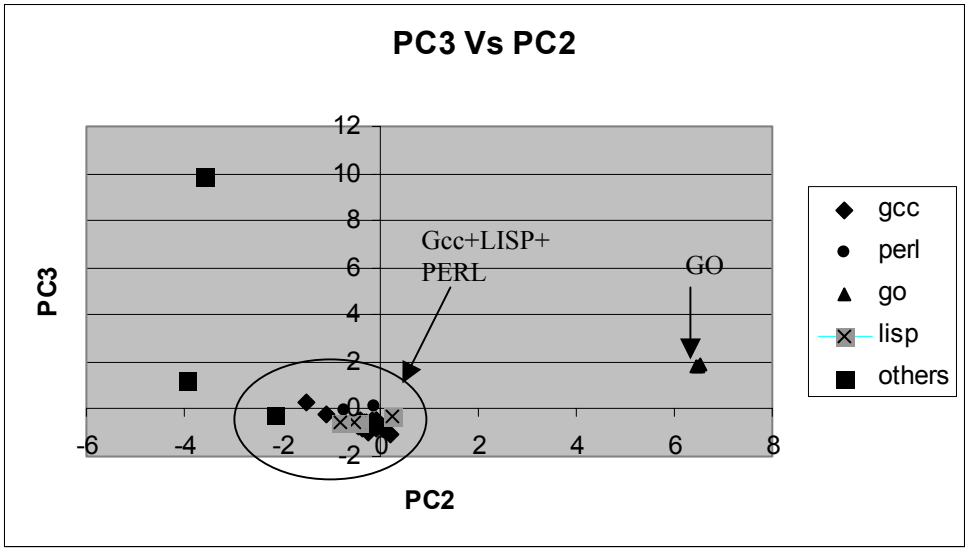
The workload space can be visualized by means of scatter plots. Three of them are shown below, each with one of the principal components, PC1, PC2 or PC3 on the axes.

Graph 19. Scatter plot of PC2 Vs PC1



The scatter plot between PC1 and PC2 is shown above. It can be seen that the Gcc inputs are all clustered together, thereby indicating that irrespective of the inputs used, the gcc benchmark seems to exhibit the same behavior. If the principal component had depended on lesser number of performance metrics it could have been mentioned as to how the input could have a better effect on the program. It should also be noted that PC1 and PC2 together contribute to ~67% of the variance. Also, the go benchmarks are clustered together suggesting that the behavior may not be too input dependent, or the inputs may be causing the benchmark to behave in the same way. Also shown below are the scatter plots for PC2 and PC3.

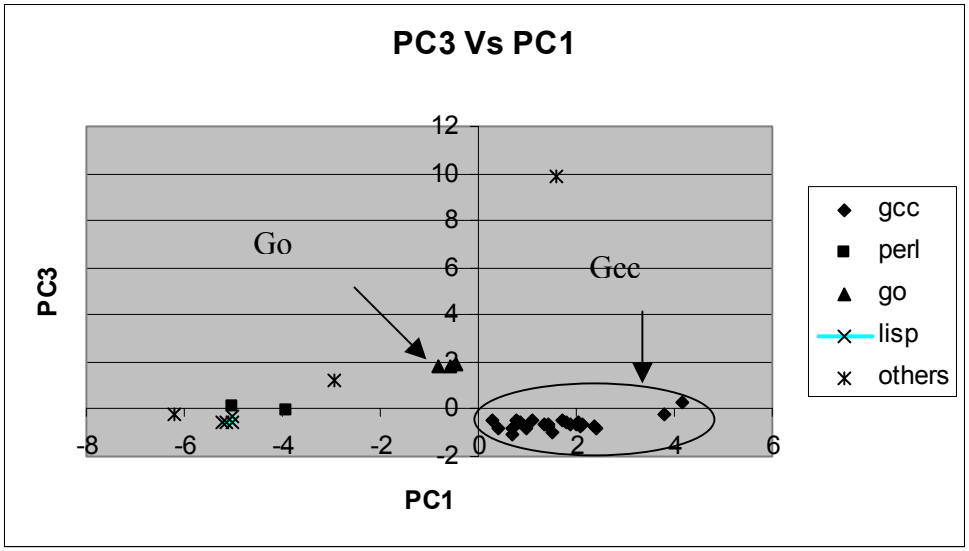
Graph 20. Scatter plot of PC3 Vs PC2



As it can be seen most of the benchmarks seem to be clustered together indicating that the inputs did not have an impact on most of the programs. This could be due to the fact that as other principal components are considered, they contain lesser information. There are a few outliers. The benchmark marked as "others" indicates, the benchmarks with only one inputs.

Also shown below, is the scatter plot between PC1 and PC3. Even in this plot it can be seen that all the gcc inputs do not cause the benchmark to behave differently and end up very near each other. It can also be seen that the different inputs of the go benchmarks exhibit the same behavior.

Graph 21. Scatter plot of PC3 Vs PC1



The three graphs above give us a clearer understanding of the behavior of the benchmarks. In graphs 19, 20 and 21, it can be clearly seen that the GCC and GO benchmarks exhibit totally different behaviors for their respective input pairs. All the points representing the input pairs of each of the benchmarks are clustered together and the two clusters themselves are far apart on the scatter plots. However, similar conclusive remarks could not be drawn based on their inconsistent behavior.

So, since the different inputs of Go exhibit similar behavior and similar inputs of the Gcc exhibit similar behavior, based on the plots, we can safely say that w could reduce the number of input sets on the benchmarks. This reduction of input sets on the SPEC-int95 should not impact any analysis drastically.

5. Conclusion

Reducing the time-to-market for microprocessors, without reducing the quality will pay off directly as dividends. Such a contribution can be made during the initial design phase. If it is possible to study the benchmarks to reduce redundancy, it can contribute to this cause. For such a reason, a clear understanding of the benchmarks is necessary. This report presents two different analyses which can be used to study benchmarks. The data collected was that of the SPECint-95 benchmarks, running on a Intel Pentium-III® processor. The data was collected using the in-built hardware performance monitoring counters.

The two analyses carried out were the regression analysis and the principal component analysis.

Through regression it was noted that there was a high correlation between the L2 cache misses per instruction and the memory transactions per instruction, for the data that was obtained. The data collected mainly belonged to the integer benchmarks, totaling to about 9 benchmarks and several input sets. An

extension to this would be to use a Pentium-IV® processor running many more benchmarks, which would be available through the SPECint-2000 benchmarks.

Also presented in this report, is the principal component analysis, which helps us study the structure of a multivariate set, by reducing the dimensionality of the data. Even though the data was reduced to a lesser dimensionality, I was unable to pinpoint the exact performance metrics that the principal components depended upon. One of the main points that has to be taken care of while conducting principal component analysis is that is makes no sense to conduct principal component analysis on variables that have low correlation between themselves, since it will take as many principal components as there were original data to account for a large portion of the variance in the variables that were present at the start of the study.

From the principal component analysis carried out, we could come to a conclusion that the different input sets of Gcc and Go end up stressing the computer in the same manner, that is, exhibiting similar behavior, and reducing the number of inputs would not affect analysis of the benchmarks in any drastic manner.

6. References

[1] L. John, "Performance Evaluation: Techniques, Tools and Benchmarks", The Computer Engineering Handbook, CRC Press, 2001.

[2] D. Bhandarkar and J. Ding, "Performance Characterization of the Pentium Pro Processor", Proceedings of the 3rd High Performance Computer Architecture Symposium, pp. 288-297, 1997.

[3] L. John, P. Vasudevan and J. Sabarinathan, "Workload Characterization: Motivation, Goals and methodology", "Workload Characterization: Methodology and Case Studies", IEEE Computer Society, 1999.

[4] Lieven Eeckhout, Hans Vandierendonck, and Koen De Bosschere, "Workload Design: Selecting Representative Program-Input Pairs", Proc. 2002 International Conference - Parallel Architectures and Compilation Techniques(PACT, 2002), IEEE CS Press, 2002, pp 83-94

[5] Designing Computer Architecture Research Workloads, IEEE Computer Society, Computer Magazine, pp. 65-71

[6] David L. Lilja “Measuring Computer Performance – A practitioner’s guide”, Cambridge University Press, 2000

[7] Reinhold P. Weicker, "An Overview of Common Benchmarks", IEEE Computer, pp. 65-75, December 1990.

[8] Reinhold Weicker, "On the Use of SPEC Benchmarks in Computer Architecture Research", Computer Architecture News, pp. 19-22, March 1997.

[9] P. Bose and T. M. Conte, Performance analysis and its impact on design. IEEE Computer, 31(5):41–49, May 1998

[10] K. Chow, A. Wright, and K. Lai, Characterization of Java workloads by principal components analysis and indirect branches. In Proceedings of the Workshop on Workload Characterization (WWC-1998), held in conjunction with the 31st Annual ACM/IEEE International Symposium on Micro-architecture (MICRO-31), pages 11–19, Nov. 1998

[11] W. C. Hsu, H. Chen, P. Y. Yew, and D.-Y. Chen. On the predictability of program behavior using different input data sets. In Proceedings of the Sixth Workshop on Interaction between Compilers and Computer Architectures (INTERACT 2002), held in conjunction with the Eighth International Symposium on High-Performance Computer Architecture (HPCA-8), pages 45–53, Feb. 2002

[12] A. S. Huang and J. P. Shen. The intrinsic bandwidth requirements of ordinary programs. In Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), pages 105–114, Oct. 1996

[13] B. F. J. Manly. Multivariate Statistical Methods: A primer. Chapman & Hall, second edition, 1994.

[14] R. H. Saavedra and A. J. Smith. Analysis of benchmark characteristics and benchmark performance prediction. ACM Transactions on Computer Systems, 14(4):344–384, Nov. 1996

[15] D. W. Wall. Predicting program behavior using real or estimated profiles. In Proceedings of the 1991 International Conference on Programming Language Design and Implementation (PLDI-1991), pages 59–70, 1991

[16] H. Cragon, Computer Architecture and Implementation, Cambridge University Press, 2000

[17] J. E. Smith, Characterizing Computer Performance with a Single Number, Communications of the ACM, October 1988

[18] Patterson and Hennessy, Computer Architecture: The Hardware/Software Approach, by Hennessy and Patterson, Morgan Kaufman Publishers, 2nd edition, 1998, ISBN 1558604286

[19] P6perf utility, http://developer.intel.com/vtune/p6perf/index.htm

[20] Perf-monitor for UltraSparc, http://www.sics.se/~mch/perf-monitor/index.html

[21] PMON http://www.ece.utexas.edu/projects/ece/lca/pmon

[22] B. Cmelik and D. Keppel, "Shade: A Fast instruction-set simulator for execution profiling", Chapter 2 in “Fast Simulation of Computer Architectures”, by T. M. Conte and C. E. Gimarc, Kluwer Academic Publishers, 1995

[23] SPEC Benchmarks, www.spec.org

[24] PC Benchmarks, www.pcbenchmarks.com

7. VITA

Michael Arunkumar, the son of Soundara Rajan and Angella Rosalind, was born in Chennai, India on September 26, 1978. After completing his work at Carmel Garden Matriculation Higher Secondary School, Coimbatore, in 1996, he entered into the Regional Engineering College (now National Institute of Technology), Trichy, India. He received the Bachelor of Engineering from Bharathidasan University in May 2000. After working for i2 Technologies for over a year, he joined the graduate school at the North Carolina state University, from where he transferred to the University of Texas at Austin in August 2002.

Permanent Address: 42, "Ave Maria",

I Cross, Elgi Nagar, Meena Estates,

Coimbatore – 641028,

Tamilnadu,

India.

This report was typed by Michael Arunkumar.