Analyzing Program Behavior of SPECint2000 Benchmark Suite using Principal Components Analysis

Aashish Phansalkar Lizy Kurian John
Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin TX 78712
{aashish, ljohn}@ece.utexas.edu

***Abstract*—Reducing simulation time during the phase of design of a microprocessor has been one of the key issues discussed in the community lately. This report tries to throw light on program behavior of the whole SPECint 2000 benchmark suite including reference and train input sets using a statistical technique called Principal Components Analysis. Analyzing the results of Principal Components Analysis and the raw data collected for the program-input pairs together helps to identify the similarity between benchmarks. It also helps to choose few program-input pairs and hence reduce simulation time. The project also talks about how we can associate program-input pairs to characteristics that are closely related to the performance of a microprocessor.**

***Index Terms*—Performance Monitoring Counters, Principal Components Analysis, Program-Input pairs.**

I. INTRODUCTION

The question of composing a representative workload consists of two parts: (i) which benchmarks to choose (ii) which input data sets to select [1]. High-level architectural simulations are extremely slow. One of the solutions to this problem is that the total number of benchmarks and input data sets should be reduced without affecting the evaluation process of the design. SPECint 2000 is a very commonly used benchmark suite by many computer architects and others to demonstrate the gain in performance due to the modification in design. Many of them cannot simulate all the program-input pairs of the benchmark suite due to different reasons. Some of them face problems compiling all the programs. But majority of them want to reduce the size of the workload and avoid running long simulations. When it comes to making a choice between the different program-input pairs it should be fair and we should try to cover maximum workload space. If we know the relative position of each of the program-input pair with respect to the other in the workload space made up of a selected workload characteristics we can make a good choice.

Workload design space can be viewed as a n-dimensional space with n number of important program characteristics that affect performance, e.g., branch prediction accuracy, cache miss rates, etc. When we measure all the characteristics it is hard to read the data, since we have a large table with many program-input pairs and a value for each of the characteristics. Also the values do not lie in the same range. We cannot compare all these benchmarks by looking at the values of their characteristics. Correlation exists between different variables, which make it even harder to find workload characteristics that affect the program behavior. One of the solutions to this problem is Principal Components Analysis.

Principal Components Analysis is a statistical technique that linearly transforms an original set of variables into a substantially smaller set of uncorrelated variables that represents most of the information in original set of variables. The goal is to reduce dimensionality of the original data set. A small set of uncorrelated variables is much easier to understand and use in the further analyses than a larger set of correlated variables. This method was first conceived by Pearson (1901) and independently developed by Hotelling (1933) [2]. We will discuss Principal Components Analysis in detail in the later section

of this report. The data that we have will be reduced to fewer variables called Principal Components derived from the original set of variables. This data can be plotted in two-dimensional space and can be observed for clusters. If we find a cluster of points in the plot of these Principal Components we can say that these program-input pairs show similar workload behavior. *Eeckhout et al* have verified this idea [3]. They also talk about how strong and weak clusters help us to select representative program-input pairs. In this report we will see some more interesting conclusions which we can draw from Principal Components Analysis.

This report is organized as follows. In section 2, the program characteristics used are listed and the methodology of the experiment is also discussed. In section 3, Principal components analysis, is discussed to give a clear idea about the technique. Section 4 talks about data analysis Section 5 discusses previous work related to this project. Section 6 concludes the discussion.

II. WORKLOAD CHARACTERISTICS AND METHODOLOGY

A. Workload Characteristics

We selected eighteen different workload characteristics to measure for each program-input pair. These characteristics are listed in Table 1. The workload characteristics are broadly classified into four types. (i) General Events (ii) Front End Events (iii) Branch Events and (iv) Memory Events. The names in the middle column of Table 1 are used to represent these characteristics in the rest of the paper. We used Pentium 4 performance monitoring counters to measure all these events. We used Brink and Abyss tools [4]. Brink and Abyss tools provide high-level interface to the Pentium 4 performance counters on Linux Systems. This is much faster approach than running simulations because the programs run on the actual hardware. There are some disadvantages to it. One of the disadvantages is that the parameters e.g. cache size or branch predictor cannot be changed and we cannot measure the change in behavior of the program for change in parameters of the microprocessor.

B. Methodology

All the experiments are run on a Pentium 4 Xeon processor. Most modern, high-performance processors have special, on-chip hardware that can be used to monitor the performance of the processor. Data collected by the on-chip performance monitoring hardware can be used to understand how applications, the operating system, and the processor are performing [5]. This data can then be used to guide efforts to improve performance by tuning the algorithms used by the application and the operating system and by tuning the code sequences used to implement those algorithms. Performance monitoring hardware is typically composed of two components: performance event detectors and event counters. By properly configuring the event detectors and counters, one can obtain counts of a variety of performance events under various conditions. Performance event detectors can be configured to detect any one of a large number of performance events (e.g., cache misses or branch mispredictions). Often, event detectors have an event mask field that allows further qualification of the event to be configured. For example, the Pentium 4 event to count load accesses to the level 2 cache (L2_LD) has an event mask that allows further qualification by the state of the cache line being accessed (i.e., modified, shared, exclusive, or invalid). The configuration of the event detector also allows qualification by the current privilege level of the processor. The tool Brink and Abyss performs all the tasks of setting the fields for the event detectors and counters. The data was collected for all the characteristics listed in Table 1. To avoid the effect of the length of benchmark and to get all the program-input pairs to a comparable level, all the characteristics were measured per instruction for a particular program-input pair. Then the data was normalized to zero mean and unit standard deviation for each characteristic. This is done to give equal weight to all the characteristics. The difference in the values of different characteristics can be of the order of 100. Normalization gets all the characteristics to a same level. We feed this normalized data to a ‘Matlab’ program, which runs the Principal Components Analysis algorithm. The result is in the form of three matrices: (1) The scores of Principal

Components for each program-input pair (2) The factor loadings of each Principal Component (3) The Eigen Values of each Principal Component which helps us to calculate the contribution of each of them towards the total variance in the data.

We did not consider test input sets of SPECint 2000 benchmark suite in our experiment because the number of instructions for each input is very small. The performance counter measurement error is significant for test input sets. This might lead to misleading results.

III. PRINCIPAL COMPONENTS ANALYSIS

Principal components analysis (PCA) is a statistical data analysis technique that presents a different view on the measured data. It builds on the assumption that many variables (in our case, program characteristics) are correlated and hence, they measure the same or similar properties of the program-input pairs. PCA computes new variables, called *principal components*, which are *linear combinations* of the original variables, such that all principal components are uncorrelated. PCA transforms the p variables $X_1, X_2, ..., X_p$ into $Z_1, Z_2, ..., Z_p$ with $Z_i = \sum_{j=1}^{p} a_{ij} X_j$. These are the p principal components. This transformation has the property *Var[Z1] > Var[Z2]>*..... which means that Z_1 contains the most information and Z_p the least which means that there is no information overlap between the principal components. Note that the total variance in the data remains the same before and after the transformation.

As stated in the property in the previous paragraph, some of the principal components will have a high variance while others will have a small variance. By removing the components with the lowest variance from the analysis, we can reduce the number of program characteristics while controlling the amount of information that is thrown away. We retain q principal components which are a significant information reduction since in most cases, q is typically about 2 to 4. In this study the original variables are the program characteristics mentioned in section 2A. By examining the most important principal components, which are linear combinations of the original program characteristics, meaningful interpretations can be given to these principal components in terms of the original program characteristics.

The next step in the analysis is to display the various benchmarks as points in the q dimensional space built up by the principal components. Computing the values of the q principal components for each program-input pair can do this.

IV. DATA ANALYSIS

As discussed in section 2B the results of Principal Components Analysis have three matrices. The first matrix gives the Eigen values, which shows the variance covered by each Principal Component. Principal Component 1(PC1) covers 36.83% of total variance. PC2 covers 17.81% of total variance, PC3 covers 14.34% of total variance and PC4 covers 11.45% of total variance. The total percentage of variance covered by PC1-PC4 is 80.42%.

Type of event	Name of the event	Description
General	num_cycles	Number of cycles
	uops_retired	Number of retired u-ops
	stores_retired	Number of retired stores
	loads_retired	Number of retired loads
Front End Events	Tcmiss	Number of trace cache misses
	itlb_reference	Number of ITLB references
	itlb_reference_hits	Number of ITLB hits
Branch events	branch_retired	Number of branches retired
	mpred_branch_retired	Number of mis-predicted branches retired
	pred_branch_retired	Number of predicted branches retired
	mpred_nt_branch_retired	Number of NT mis-predicted branches retired
	mpred_t_branch_retired	Number of T mis-predicted branches retired
	pred_nt_branch_retired	Number of NT predicted branches retired
	predicted_t_branch_retired	Number of T predicted branches retired
Memory access events	ld_miss_L1	Number of L1 cache load misses
	ld_miss_L2	Number of L2 cache load misses
	dtlb_miss	Number of DTLB misses
	mem_retired	Number of memory operations retired

Table 1. Different types of events used in the analysis, which form a set workload characteristics. All the events were measured using Pentium 4 Performance Monitoring Counters.

The second set of information we get from PCA is the factor loading matrix. The factor loadings are the coefficients of each of these Principal Components. E.g. the factor lodaings of PC1, PC2, PC3 and PC4 are basically the coefficients of the linear equation of PC1, PC2, PC3 and PC4 respectively. The factor loadings of first four Principal Components are shown in Figure 1. Looking at the factor loadings we can conclude that PC1 is positively dominated by the branch events and memory reference events. The vertical names suggest the events each of the set of four principal components show. For example the branch retired event has a factor loading of 2.7 and pred_nt_branch_retired event has a factor loading of close to 3 for the first Principal Component. Similarly PC1 is negatively dominated by the front end events; refer to Table 1 and the general events. Thus PC1 shows a contrast of branch, memory events and front-end events. Similarly PC2 and PC3 can be analyzed with reduced importance in this order.

Another set of information we collect from PCA is that about the scores of each program-input pair. The scores for each of the retained Principal Components can be plotted on a two-dimensional plot considering two Principal Components at a time. Figure 2 shows a plot of PC2 Vs PC1 for all the program-input pairs. From the factor loadings we can conclude that the

program-input pairs, which have a higher value for PC1, have higher counts for branch prediction events and memory access events. So the program behavior of these program-input pairs is dominated by this characteristic. E.g. mcf shows very high value for PC1 as compared to others, the gzip and the twolf. All the rest of program-input pairs are either near zero or show a negative value for PC1. This means that their program behavior is either equally dominated by branch, memory and front events or more by the front-end events. Same conclusions can be drawn from the factor loadings of PC2 PC3 and PC4.

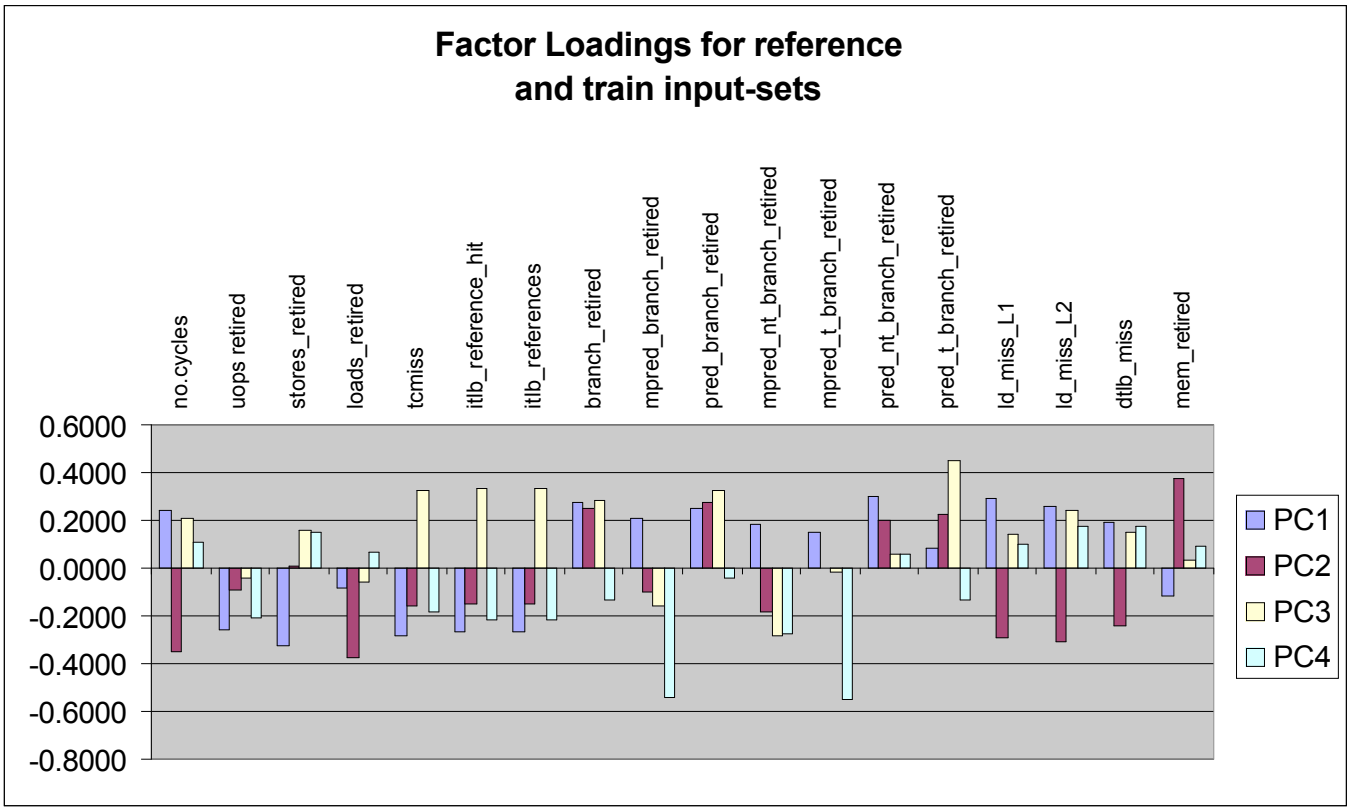

Figure 1. Factor loadings for reference and train input sets. These factor loadings were generated from the Principal Components Analysis of the data for both reference and train input sets.

The plots for scores of Principal Components can also be used to do clustering of the program-input pairs and show qualitative similarity between different program-input pairs. We can then assume that all the program-input pairs in a cluster show similar behavior and hence only one representative pair out of the cluster can be selected as a part of target workload. Figure 2 and Figure 3 show some obvious clusters. Rest of the clusters are not drawn to preserve the clarity of the plot. But many clusters can be drawn which significantly reduce the number of program-input pairs to be selected as a part of the target workload. One of the observations from Figure 2 and Figure 3 is that all the train program-input sets lie close to their respective reference program-input pairs. E.g. in Figure 2 mcf, eon, gzip and crafty and all others show the same behavior. Figure 3 shows twolf, perl.perfect and vpr.route. Again the rest of them are not shown to maintain the clarity of the plot. Also almost all the program-input pairs of the same program are clustered except for perl. Perl is scattered in three out of the four quadrants of the plot in Figure 2. This shows that the behavior of perl is dominated by the input set for the characteristics we measured in our experiments. For all other programs, the input set does not make a lot of difference.

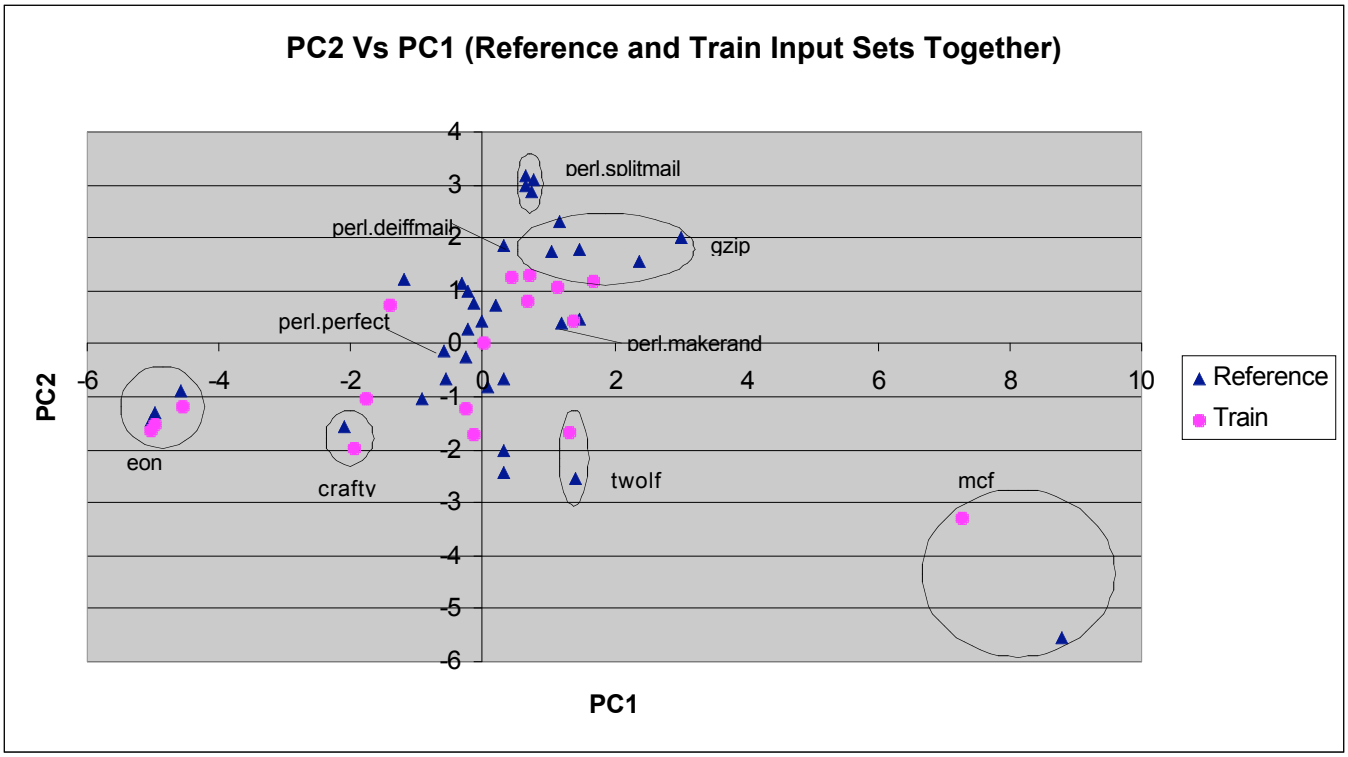

Figure 2. Plot for the scores of PC2 Vs PC1 for all the reference and train input sets.

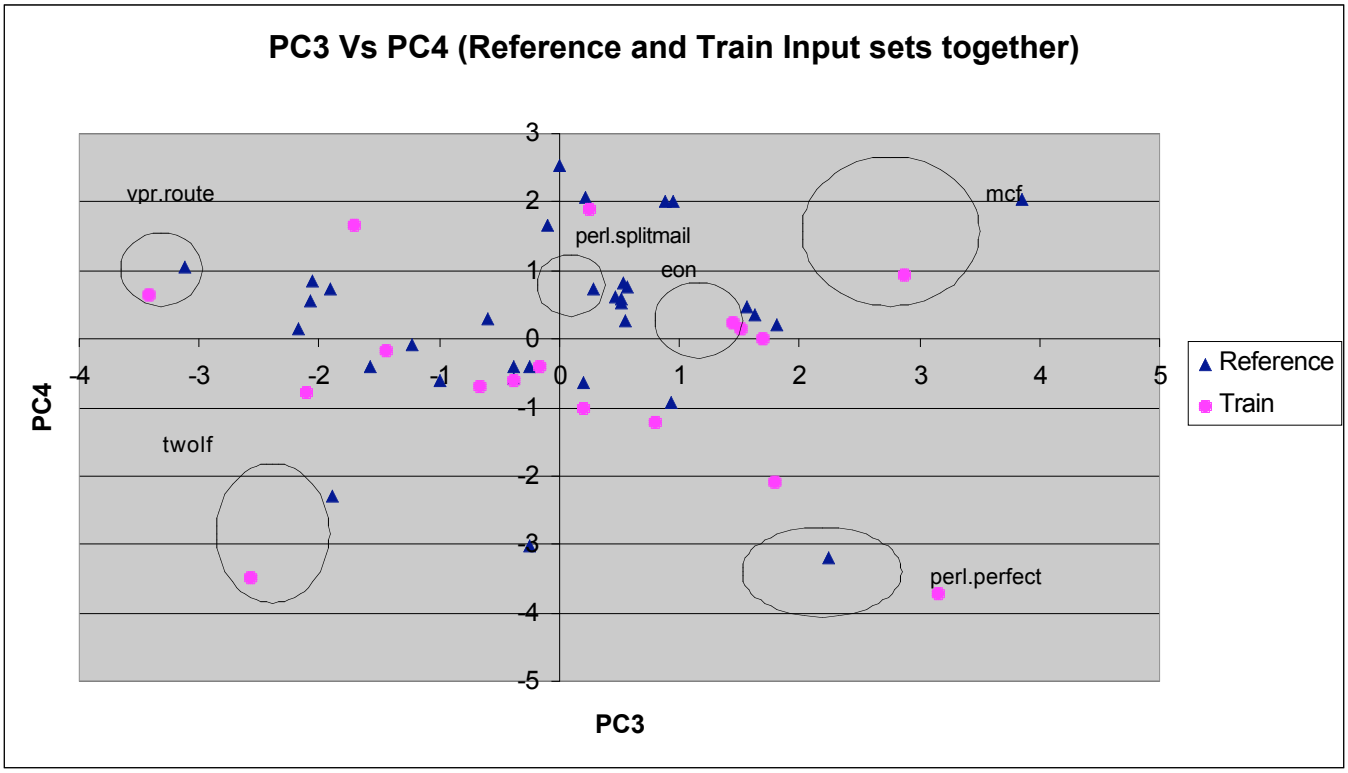

Figure 3. Plot for scores of PC4 Vs PC3 for all the reference and train input sets.

We can use this information very well to select representative program-input pairs. Since the train inputs lay very close to their respective reference program-input pairs we can very well choose train input sets because they have lesser number of instructions. If there are more than one train input sets we can choose the smallest one out of the cluster. This also applies to the reference input sets where their train input sets are not a very good representative of their behavior e.g. perl.splitmail is not clustered with any of the perl train input sets but all the four perl.splitmail reference input sets are clustered.

Another analysis we can make by looking at the raw data and the above plots is that the behavior of program-input pairs with higher values of PC1 is dominated by number of branch events and hence if we want to evaluate performance or measure accuracy of a branch predictor we should definitely consider choosing program-input pairs with higher values of PC1 rather choosing the ones with lower values of PC1 e.g. eon. Similarly, this is applicable to the study of memory behavior as well.

V. PREVIOUS WORK

There has been a significant work done by *Lieven Eeckhout et al*. in applying Principal Components Analysis to design target workload. [1] [3] [4] [6]. He has also verified the idea of using clusters to find similarity between benchmarks [1].
He has also used Cluster Analysis in his experiments to quantify the similarity between benchmarks, but George [2] suggests that Principal Components Analysis and Cluster Analysis show the same results and we do not need both the techniques together. However the methodology and the workload characteristics chosen by him for the experiment are different from our approach in this project. He collects data using cycle accurate simulator.

Eeckhout et al [4] recently discussed about behavior of Java programs at microarchitectural level and concluded that their behavior is dominated by the virtual machine used to run these programs. The program or the input set does not play a role in dominating program behavior for a given microarchitecture. Our methodology closely follows the methodology in this paper but has different set of characteristics. We also measured the data on per instruction basis, whereas he did it per cycle.

VI. CONCLUSION

This discussion throws light on the program behavior of SPECint 2000 benchmark suite. We used Principal Components Analysis to show similarity between benchmarks. We can conclude that all the program-input pairs except perl show that the train input sets can be treated as representative of their respective reference program-input pairs. For some reference input sets, which do not have, a representative train set are clustered and a shorter input set can be selected from the cluster as a part of the target workload. This will help to reduce simulation time. As discussed in the last part of the data analysis section we can also choose program-input pairs based on which component of the microprocessor we are trying to evaluate. Different components of a microprocessor can be simulated separately to find out their own accuracy or performance. While doing so Principal Components Analysis can be used to select few program-input pairs out of all and hence reduce simulation time.

REFERENCES

[1] Lieven Eeckhout, Hans Vandierendonck and Koen De Bosschere, “Workload Design:Selecting Representative Program-Input pairs” PACT 2002

[2] George Dunteman “Principal Components Analysis” Sage Publications.

[3] Lieven Eeckhout, Hans Vandierendonck and Koen De Bosschere “Quantifying the Impact of Input Data Sets on Program Behavior and its Applications.” Journal of Instruction-Level Parallelism (www.jilp.org), Volume 5, April 2003

[4] Lieven Eeckhout “How Java Programs Interact with Virtual Machines at Microarchitectural Level” OOPSLA 2003.

[5] Brinkley Sprunt “Performance Monitoring Features of Pentium 4 Processor” IEEE Micro Magazine 2002.

[6] Lieven Eeckhout, Hans Vandierndonck and Koen De Bosschere “Designing Computer Architecture Research Workloads” IEEE Computer magazine Vol 36