Pruning Hardware Evaluation Space via Correlation-Driven Application Similarity Analysis

Rosario Cammarota † Arun Kejariwal‡ Paolo D'Alberto ‡ Sapan Panigrahi‡
Alexander V. Veidenbaum† Alexandru Nicolau†

†Department of Computer Science
University of California at Irvine
Irvine, CA, USA

‡Yahoo! Inc.
Sunnyvale, CA, USA

ABSTRACT

System evaluation is routinely performed in industry to select one amongst a set of different systems to improve performance of proprietary applications. However, a wide range of system configurations is available every year on the market. This makes an exhaustive system evaluation progressively challenging and expensive.

In this paper we propose a novel similarity-based methodology for system selection. Our methodology prunes the set of candidate systems by eliminating those systems that are likely to reduce performance of a given proprietary application. The pruning process relies on applications that are similar to a given application of interest whose performance on the candidte systems is known. This obviates the need to install and run the given application on each and every candidate system.

The concept of similarity we introduce is performance centric. For a given application, we compute the Pearson's correlation between different types of resource stall and cycles per instruction. We refer to the vector of Pearson's correlation coefficients as an application signature. Next, we assess similarity between two applications as Spearman's correlation between their respective signature. We use the former type of correlation to quantify the association between pipeline stalls and cycles per instruction, whereas we use the latter type of correlation to quantify the association of two signatures, hence to assess similarity, based on the difference in terms of rank ordering of their components.

We evaluate the proposed methodology on three different micro-architectures, viz., Intel's Harpertown, Nehalem and Westmere, using industry-standard SPEC CINT2006. We assess performance centric similarity among applications in SPEC CINT2006. We show how our methodology clusters applications with common performance issues. Finally, we show how to use the notion of similarity among applications to compare the three architectures with respect to a given Yahoo! property.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: [Measurement techniques, Performance attributes]; D.4.8 [**Software Engineering**]: [Measurements, Modeling and prediction, Stochastic analysis]

CF May 03-05 2011, Ischia, Italy

General Terms

Measurements, Performance

Keywords

Performance analysis, Measurements, Application similarity

1. INTRODUCTION

In the Internet space, one of the key metrics associated with user satisfaction, is low response time. Likewise, it is critical to support high throughput to service a multitude of users. For example, given an application such as Yahoo! Finance (referred to as Y! subsequently) determining the best hardware, from performance standpoint, is imperative to achieve the aforementioned objectives. However, the gamut of candidate micro-architecture configurations is very large owing to frequent roll-out by hardware vendors such as Intel [1], making system evaluation progressively more challenging and expensive.

In this paper we propose a novel similarity-based methodology for system selection. Our methodology prunes the set of candidate systems by eliminating those systems that are likely to reduce performance of a given proprietary application. The pruning process relies on applications that are similar to a given application of interest whose performance on the candidte systems is known. This obviates the need to install and run the given application on each and every candidate system.

Although there has been a large body of work in the context of quantitative assessment of similarity among applications, we believe that the technique and the applications we propose in this paper are complementary to those of prior work. Existing techniques in this regard have been geared towards workload design and characterization [2, 3, 4, 5, 6] to classification of benchmarks [7, 8]. Several techniques, using micro-architecture dependent [9] as well as micro-architecture independent program features [10, 11], have been proposed to assess application similarity for several purposes such as reducing simulation time and predicting application performance. The underlying similarity metric used in these techniques does not capture the relation between performance, as measured by CPI (cycles per instruction), and resource stalls, which are directly responsible for performance degradation. This relation allows to gain insight into software/hardware interactions. In fact, decomposing micro-architecture performance into individual CPI components enables differentiation of cycles consumed in active computation from those spent in handling various miss events. Capturing the relation between CPI and resource stalls is key to determine similarity between two applications with respect to causality of application performance

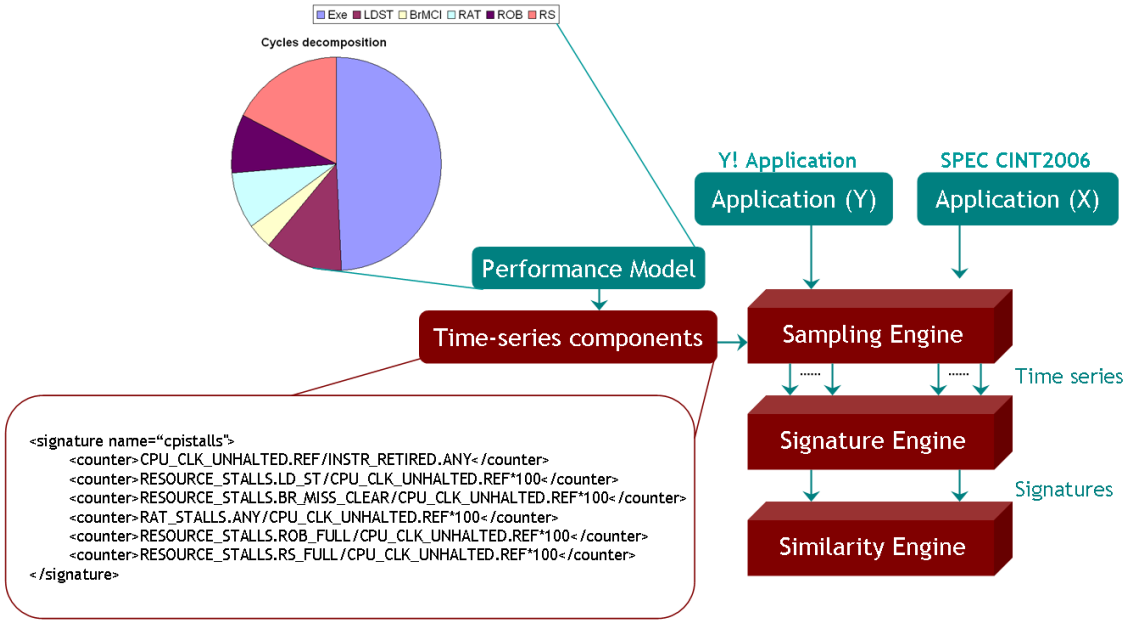


Figure 1: Overview of the methodology

due to the stalls in the processor pipeline. Subject to the above relation, applications in, say, SPEC CPU2006 similar to a given reference application can be used to prune the evaluation space [1] of candidate micro-architecture configurations.

To the best of our knowledge, this is the first work to evaluate application similarity based on correlation between CPI and resource stalls. The main contributions of the paper are as follows:

- We define a signature of an application as a vector of Pearson's correlation coefficients [13, 14] between CPI (performance) and different resource stalls (application features). We selected resource stalls as they directly affect CPI. Each component of the signature falls in the range $[-1, 1]$. A correlation coefficient of 1 implies that an increase in the corresponding resource stalls corresponds to an increase in CPI.

  We use Pearson's correlation to quantify the association between each type pipeline stall and CPI, hence capturing in a single number the effect of such stall to the dynamic behavior of performance. The existence of a strong correlation, in our context, is a sufficient condition for causality between each type of resource stall and CPI.
- Next, we propose a technique to compute similarity between two applications using their respective signatures. Specifically, similarity between two applications is assessed as the Spearman's correlation [15, 16], ρ, between their respective signatures.

  Spearman's correlation is a type of rank correlation, which assigns a rank to each component of a given signature with respect to their impact to CPI, then computes the correlation between two signatures on the basis of such ranking. A value of ρ close to one signifies that the order with which pipeline stalls are ranked in both signatures is the same or almost the same. Hence, both the applications are affected with the same performance issues, in the same proportion, i.e. the two applications are similar from a micro-architectural standpoint. Because a given type of stall can have a stronger association to CPI than others, the use of Spearman's correlation quantifies the association between two signatures, hence the similarity between two applications.
- We address the problem of finding similarity among a set of applications proposing two clustering algorithm based on minimum spanning tree (MST). The resulting clusters provide a performance driven classification of the applications. We use clustering algorithms based on MST as a means to visualize the relation between a set of applications based on our similarity approach.

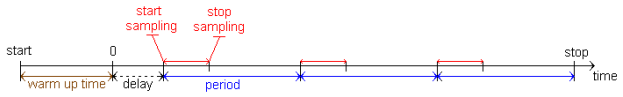


Figure 2: Sampling methodology

We validate our approach with a set of experimental results using applications in SPEC CINT2006 and a given Yahoo! application on three different Intel Xeon micro-architectures, viz., Harpertown, Nehalem and Westmere. We believe that the architectures we selected represent the state-of-the-art of micro-architectures for server applications. We apply our methodology on a set composed of SPEC CINT2006, and a given Yahoo! application. Experimental results, presented in Section 4, illustrte how to identify subsets of SPEC CINT2006, and how to isolate a subset of SPEC CINT2006 that is similar to one Yahoo! application. Eventually we show how to use the methodology proposed to compare the three micro-architectures in use, and select the best among them to run a given Yahoo! application.

The rest of the paper is organized as follows: the methodology is detailed in Section 2. Experimental results are presented in Section 4. Related work is discussed in Section 5. Finally, the conclusions are presented in Section 6.

[1] Note that the performance of SPEC CPU2006 applications for all the hardware configurations in the market is available at the official SPEC CPU website [12].

2. METHODOLOGY

In this section we present our methodology, the notation we use, and mathematical concepts and tools used. An overview of the methodology is shown in Figure 1. First, in subsection 2.1, we present our performance model. Second, in subsection 2.2, we detail the sampling engine used for collecting data, as a time-series, for CPI and stalls. The sampling engine uses hardware performance counters for data collection. Third, in subsection 2.3, we define an application signature subject to the performance model presented in subsection 2.1. In subsection 2.4, we present a novel approach to compute similarity between two applications using their respective signatures. Fifth, in subsection 3.1, we describe how to cluster applications in a suite such as SPEC CINT2006 based on their signatures. Finally, in subsection 3.2, we present a technique to find similar applications in SPEC CINT2006 to a given (reference) Y! application.

2.1 Performance model

From a micro-architecture perspective, the target of performance optimization is the minimization of the cycles per instruction (CPI). One can divide the cycles into useful cycles, where the pipeline is not stalled and stall cycles, where the pipeline is stalled. This dissection of CPU cycles forms the basis of our performance model, captured by Equation 1. From the equation we note that the achieved CPI can be dissected into two components: optimal CPI that can be achieved and CPI lost due to pipeline stalls. Optimal CPI corresponds to the best achievable CPI on a given micro-architecture. For example, on Intel Nehalem, optimal CPI is 0.25. CPI lost due to stalls can be ascribed, in part, to the cycles lost due to resource stalls. Therefore we model the overall loss of CPI due to resource stalls as a sum of cycles lost due to each resource stall:

$$CPI_{achieved} = CPI_{optimal} + CPI_{pipeline\ stalls} \quad (1)$$

Stall cycles owing to resource stalls adversely affect performance [17, 18, 19]. The different type of resource stalls we address are enumerated as follow: branch misprediction stalls; load/store queue(s) stalls; reorder buffer full stalls; reservation stations stalls; register renaming table saturated stalls.

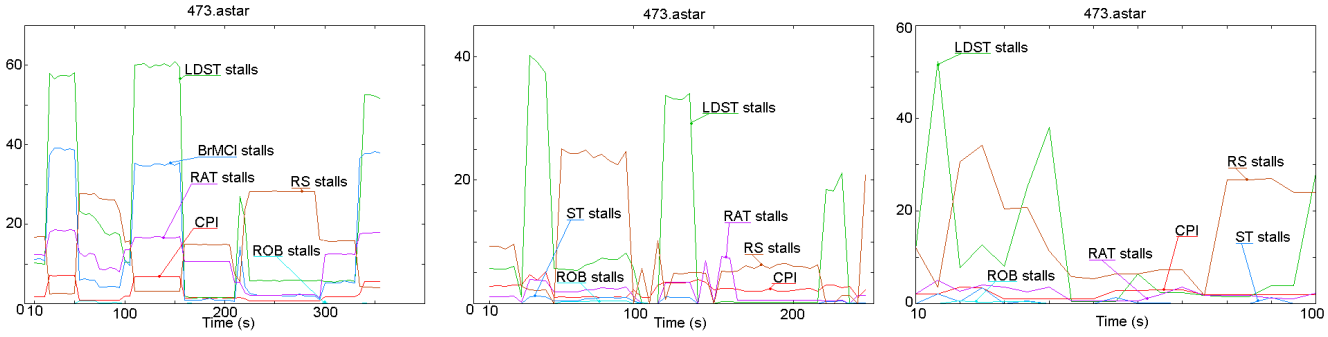
Hardware performance counters are used to measure resource stalls. We selected counters corresponding to resource stalls as they directly affect CPI. Counters related to other kind of events do not always shed light on cycles lost due to the particular event. For instance, out-of-order architectures, as those considered in this work, cache misses affect CPI only when the latency to access to memory cannot be hidden. Admittedly, resource stalls as captured by the hardware performance counters, are not mutully exclusive, i.e., a stall may overlap with one or more other resource stalls [17]. However, in [18], Azimi et al. showed that the effect of the overlap between the different type of stalls on performance is minimal.

2.2 Sampling Engine

The availability of dedicated performance monitoring unit on modern micro-architectures [20, 21] enables the collection of run-time application characteristics [22], with negligible overhead. The Sampling Engine collects the data for CPI and resource stalls as per the schema shown in Figure 2. The time-line illustrates the execution of an application. During the *warm up* phase, the application is deployed and prepared to be run on the target system. The application execution corresponds to the duration between *start* and *stop* time stamps in Figure 2. After a *sampling delay* (s_d), which is configurable, the counters are collected periodically for a *sampling interval* (s_i). The purpose of the sampling delay is to ensure that hardware performance counters are sampled only after the application has reached a steady state. For example, in Java-based applications, class loading occurs at the start of the application. This does not constitute the steady state; hence, in order to avoid tainting of performance data (collected via hardware performance counters), the Sampling Engine introduces a sampling delay. Subsequently, at the beginning of every *sampling period* (s_p), which is configurable, a list of counters to be sampled is set. The counters are sampled for a duration specified by *sampling interval*, which is configurable as well.

The data collected by the Sampling Engine yields a time series of values, one series for each counter. For example, Figure 3 illustrates the time series for the `473.astar` – an application in SPEC CINT2006 – on three different Intel architectures (refer to Table 2 for their detailed configuration). The time series were obtained with $s_d = 10sec$, $s_p = 5sec$ and $s_i = 3sec$. Each point in the CPI time series is the ratio of CPU cycles and the number of instructions retired during the corresponding sampling interval. Each point in the other time series is the ratio of the stall cycles and the CPU cycles in the corresponding sampling interval.

At the end of sampling, we obtain a multi-dimensional time series $\mathbf{c}_t$ comprising of a time series of CPI, and as many series as the number of hardware counters available to measure resource stalls: $\mathbf{c}_t = (c^0, c^2, \ldots, c^{M-1})_t$. In the current case, $M = 5$.



(a) Harpertown (b) Nehalem (c) Westmere

Figure 3: Time-series of 473.astar on three micro-architectures ($s_i = 3$, $s_p = 5$)

	Harpertown					Nehalem					Westmere				
	LDST	BrMCl	RAT	ROB	RS	LD	ST	RAT	ROB	RS	LD	ST	RAT	ROB	RS
400.perlbench	0.30	0.71	0.56	0.42	-0.61	0.29	-0.38	-0.14	0.28	-0.46	0.14	0.14	0.23	0.15	-0.06
483.xalancbmk	0.71	0.37	0.15	0.45	0.87	-0.07	0.50	0.39	-0.05	0.79	-0.16	0.31	0.32	-0.03	0.37
403.gcc	-0.50	0.37	-0.42	-0.12	0.60	0.07	-0.09	0.02	0.09	0.73	-0.07	-0.07	-0.12	-0.16	-0.19
429.mcf	0.44	0.65	0.40	0.03	-0.39	0.26	-0.15	-0.03	-0.25	0.26	-0.01	-0.02	0.05	0.14	-0.02
401.bzip2	-0.01	0.23	0.01	-0.10	0.09	-0.14	-0.22	-0.23	-0.24	0.10	0.08	0.12	0.08	-0.07	-0.05
445.gobmk	0.48	-0.20	0.16	-0.03	-0.22	0.00	0.03	0.04	-0.03	-0.03	0.15	0.11	0.16	0.25	0.24
456.hmmer	0.41	0.24	0.57	0.17	0.99	-0.02	0.08	-0.07	0.01	-0.04	0.23	0.11	0.22	0.34	0.36
458.sjeng	0.81	0.28	0.49	0.58	0.27	0.06	-0.13	0.01	0.06	-0.18	0.08	0.15	0.17	0.20	0.23
462.libquantum	0.99	0.08	0.38	-0.93	-0.75	0.05	-0.09	-0.03	-0.09	-0.07	-0.01	-0.01	-0.02	-0.04	-0.04
464.h264ref	0.77	0.25	-0.08	0.01	-0.08	0.62	0.19	0.10	0.17	0.60	0.27	0.16	0.26	0.30	0.32
471.omnetpp	0.57	0.56	0.56	0.55	0.55	0.49	0.51	0.50	0.54	0.51	0.21	0.18	0.21	0.31	0.31
473.astar	0.94	0.96	0.80	-0.24	-0.88	0.51	0.15	0.16	-0.38	-0.66	0.03	0.41	0.33	0.21	0.34

Table 1: Signatures for applications in SPEC CINT2006

2.3 Signature Engine

We use Pearson's correlation to quantify the causality relation between CPI and each type of resource stall. Let M be the number of hardware counters available to measure resource stalls. We define an application signature as a *M-dimensional* vector (in this paper $M = 5$) wherein each element of the vector is the Pearson's correlation between the time series of CPI and corresponding type of resource stall, see Figure 4.

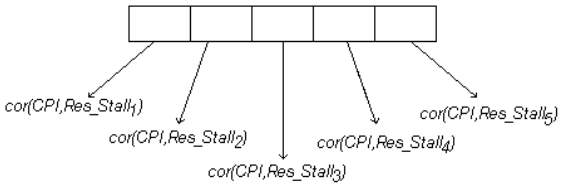


Figure 4: Illustration of an Application Signature

The value of Pearson's coefficient coefficient ranges from -1 to +1 and it is independent of the units of measurement (the series ranges). Given two n-samples series, $\{x_i\}$ and $\{y_i\}$, the Pearson's coefficient of correlation is defined as follows:

$$r(x,y) = \frac{1}{N-1}\sum_{i=0}^{N-1}\frac{(x_i-\mu_x)}{\sigma_x}\frac{(y_i-\mu_y)}{\sigma y} \tag{2}$$

where

$$\mu_x = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$
$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu_x)^2 \tag{3}$$

A coefficient of correlation of +1 indicates that the two series move in the same direction (i.e., both increase or both decrease). On the other hand, a correlation coefficient of -1 indicates that two series move in the opposite directions at all times (one increases the other decreases). A correlation coefficient of 0 indicates that the series do not have a detectable relation. In the current context, a high correlation between CPI and a given type of resource stall would suggest that the increase in CPI is induced, in part, by an increase in the given type of resource stall.

The signatures for all the applications in SPEC CINT2006 are reported in Table 1. Each row in the table contains the signature of the corresponding application on Intel's Harpertown, Nehalem and Westmere. Each component of a given application signature represents the correlation between a given pipeline stall and CPI, and it is labeled accordingly. In particular, in the case of Harperton, the first component of the signature indicates is associated to load/store queue stalls (LDST); the second component is associated branch to misprediction stalls (BrMCl); the third component is associated to register alias table stalls (RAT); the fourth component is associated to reorder buffer stalls (ROB); the last component is associated to reservation stations stalls (RS). Nehalem and Westmere is different from Harpertown in at least two aspects: the pipeline does not stall in presence of branch misprediction; the load/store queue is split in two queues. One queue serves load instructions waiting for data, the other queue serves store instructions waiting to write back. Therefore, the signature contains components associated to load queue stalls (LD) and to store queue stalls (ST).

2.4 Similarity Engine

The Similarity Engine determines the similarity between two applications using their respective signatures. For this, the Similarity Engine computes a rank correlation measure, as Spearman's ρ [15, 16, 23], between the application signatures.

For each application signature, a rank is assigned to each element of a signature. In case two elements have the same rank, an arithmetic average of the rank is assigned to the corresponding elements of the signature. Let $\pi()$ denote the ranking procedure. Given two signatures $\mathbf{s}$ and $\mathbf{v}$, we obtain the vectors $\mathbf{s}_\pi = \pi(\mathbf{s})$ and $\mathbf{v}_\pi = \pi(\mathbf{v})$. Then, Spearman's ρ correlation coefficient is computed as follows:

$$\rho(\mathbf{v}_\pi, \mathbf{s}_\pi) = 1 - \frac{6\sum_{i=0}^{N-1}(\pi(\mathbf{s})_i - \pi(\mathbf{v})_i)^2}{N^3 - N} \tag{4}$$

A Spearman's correlation of 1 between two application signatures implies that resource stalls affect CPI in the same order.

In alternative, the Kendall's τ [24], which is another type of rank correlation, can also been used to assess application similarity. However Spearman's ρ and Kendall's τ are equivalent as it is shown by Diaconis and Graham in [25].

3. APPLICATION CLUSTERING

In the next two subsections we present two clustering algorithms (Algorithm 1 and Algorithm 2), based on minimum spanning tree (MST) to derive clusters of similar applications. Algorithm 1 is implemented in an analysis tool. Given a set of applications and a parametric threshold, Algorithm 1 returns clusters of applications sharing performance issues on a given micro-architecture. Algorithm 2 is implemented in a search tool to search for a single cluster of applications that are similar to an application of interest.

Algorithm 1 Performance centric clustering of a set of applications

Input: $\mathbf{S} = \{$**set of signatures**$\}$ and a threshold $g \in (0, 2)$.
Output: Set of clusters containing edges with weights smaller than g.

procedure $recursive\_max\_edge\_cut(g, \mathbf{S})$
$\mathbf{G} =$ **compute_similarity_graph**$(\mathbf{S})$
if $(g > edges(G))$ **then**
  print S
  exit
else
  $\mathbf{MST} =$ **minimum_spanning_tree**$(\mathbf{G})$
  $\{\mathbf{MST_j}\} =$ **cut_max_edges**$(\mathbf{MST})$
  for all $\mathbf{m} \in \{\mathbf{MST_j}\}$ **do**
    $\mathbf{mm} =$ **vertex**$(\mathbf{m})$
    $recursive\_max\_edge\_cut(g, \mathbf{mm})$
  end for
end if

Algorithm 2 Performance centric search of one cluster containing an application of interest

Input: $\mathbf{S} = \{$**set of signatures**$\}$, $\mathbf{v_{ref}}$, is the signature of a reference application.
Output: Minimum cluster containing the reference application.

procedure $recursive\_max\_edge\_cut\_reference(\mathbf{v_{ref}}, \mathbf{S})$
$\mathbf{G} =$ **compute_similarity_graph**$(\mathbf{S} \cup \{\mathbf{v_{ref}}\})$
if (weight edges node $\mathbf{v_{ref}}$ = max weight edges $\mathbf{G}$) **then**
  print S
  exit
else
  $\mathbf{MST} =$ **minimum_spanning_tree**$(\mathbf{G})$
  $\{\mathbf{MST_j}\} =$ **cut_max_edges**$(\mathbf{MST})$
  for all $\mathbf{m} \in \{\mathbf{MST_j}\}$ **do**
    if $(\mathbf{v_{ref}} \in \mathbf{m})$ **then**
      $\mathbf{mm} =$ **vertex**$(\mathbf{m})$
      $recursive\_max\_edge\_cut\_reference(\mathbf{v_{ref}}, \mathbf{mm})$
    end if
  end for
end if

3.1 Performance centric clustering of a set of applications

Algorithm 1 is described with the following steps. First, we compute pairwise application similarity (as discussed earlier in this section) between all the applications in a given benchmark suite. This yields an application similarity matrix. Second, we "normalize" the similarity matrix so that each cell in the matrix has a positive entry. Specifically, each element in the matrix is negated and value 1 is added to it. The similarity matrix can be interpreted as an adjacency matrix of an application similarity graph wherein the nodes in the graph correspond to the different applications and the edge weights correspond to the entries in the similarity matrix. It should be noted that a similarity graph is a fully connected graph. Third, we use Prim's algorithm [26] to determine the MST of the similarity graph. Note that if each edge has a distinct weight then there will only be one, unique minimum spanning tree [27, 28]. The edges in the similarity graphs obtained in our experiments have distinct weights. Similar applications are group of nodes of the MST that are close to each other, in terms of weighted hops on the MST, i.e., arcs with maximum weight divide the MST into clusters of applications. Applications belonging on the same cluster are similar, whereas applications belonging on different clusters are dissimilar subject to the similarity metric defined in the previous section.

Algorithm 1 recursively partitions the minimum spanning tree of the similarity graph. At each step of the recursion, clusters obtained in the previous iteration are partitioned by cutting the edges with maximum weight. Unrestricted iterative partitioning would yield partitions comprising of one application each. Therefore, we use a threshold as input parameter to Algorithm 1. A small value of the threshold would result in a small clusters, since differences on the ordering of the ranks of the signatures would be highlighted more. On the opposite, a high threshold would result in large clusters. Clusters of similar applications in SPEC CINT2006, obtained using Algorithm 1, are presented in Section 4.

3.2 Performance centric search of one cluster containing an application of interest

Algorithm 2 formally describes how, given a reference application and a set of benchmarks, to search for a subset of benchmarks similar to the reference application. First, signatures of the reference application and the applications in the benchmark suite are measured. Second, Algorithm 2 computes pairwise application similarity between all the applications. This yields an application similarity matrix which contains the reference applications and its relation with the benchmarks in terms of similarity. Third, the MST of the similarity graph corresponding to the similarity matrix is computed.

Algorithm 2 starts with one cluster containing all the applications. Differently from the previous algorithm, this algorithm searches for one cluster containing the reference application. That is, Algorithm 2 recursively prunes the MST of all the clusters that not contain the reference application. Experimental results obtained using Algorithm 2 are presented in Section 4.

4. EXPERIMENTAL RESULTS

We evaluate our techniques on three different architectures and we give the system configurations in Table 2. We use applications taken from the industry-standard SPEC CINT2006 [29], and one Yahoo! proprietary application. Note that our methodology is independent of any specific application or benchmark suite, and reference application.

Micro-architectures			
Vendor, chipest	Intel(R), Xeon(R)		
Model	L5420 (Harpertown) @ 2.50GHz	E5520 (Nehalem) @ 2.26GHz	X5680 (westmere) @ 3.33GHz
L1 I/D cache [KB]	32	32	32
L2 cache	6MB	256KB	256KB
LLC	None	8MB	12MB
Main memory			
Memory [GB]	16	24	24
System level configuration			
Compilers, optimization level	Intel icc v11.1, -fast		
Operating system	Linux Red Hat AS 4 update 7		

Table 2: System configurations

	400.perlbench	483.xalancbmk	403.gcc	429.mcf	401.bzip2	445.gobmk	456.hmmer	458.sjeng	462.libquantum	464.h264ref	471.omnetpp	473.astar
400.perlbench	1											
483.xalancbmk	-0.8	1										
403.gcc	-0.1	-0.1	1									
429.mcf	0.7	-0.2	-0.4	1								
401.bzip2	0.3	-0.3	0.6	0.3	1							
445.gobmk	0.1	0.1	-1	0.4	-0.6	1						
456.hmmer	-0.5	0.1	0.2	-0.4	0.4	-0.2	1					
458.sjeng	0	0.3	-0.9	0.3	-0.8	0.9	-0.5	1				
462.libquantum	0.1	0.1	-0.7	0.6	0.1	0.7	0.3	0.4	1			
464.h264ref	0.3	0.3	-0.6	0.8	-0.2	0.6	-0.6	0.7	0.5	1		
471.omnetpp	0.2	0.2	-0.5	0.8	0.3	0.5	0.1	0.3	0.9	0.7	1	
473.astar	0.7	-0.2	-0.4	1	0.3	0.4	-0.4	0.3	0.6	0.8	0.8	1

Table 5: Correlation matrix for SPEC CINT2006 on Harpertown

	400.perlbench	483.xalancbmk	403.gcc	429.mcf	401.bzip2	445.gobmk	456.hmmer	458.sjeng	462.libquantum	464.h264ref	471.omnetpp	473.astar
400.perlbench	1											
483.xalancbmk	-1	1										
403.gcc	-0.1	0.1	1									
429.mcf	-0.3	0.3	0.4	1								
401.bzip2	-0.4	0.4	0.3	0.9	1							
445.gobmk	-0.1	0.1	-0.8	0.1	0	1						
456.hmmer	0.1	-0.1	-0.3	-0.6	-0.2	-0.3	1					
458.sjeng	1	-1	-0.1	-0.3	-0.4	-0.1	0.1	1				
462.libquantum	0.3	-0.3	-0.1	0.7	0.5	0.5	-0.6	0.3	1			
464.h264ref	0.1	-0.1	0.3	0.6	0.8	-0.3	0.2	0.1	0.4	1		
471.omnetpp	-0.3	0.3	0.1	-0.7	-0.5	-0.5	0.6	-0.3	-1	-0.4	1	
473.astar	0.7	-0.7	-0.6	0	-0.1	0.6	-0.1	0.7	0.7	0.1	-0.7	1

Table 6: Correlation matrix for SPEC CINT2006 on Nehalem

	400.perlbench	483.xalancbmk	403.gcc	429.mcf	401.bzip2	445.gobmk	456.hmmer	458.sjeng	462.libquantum	464.h264ref	471.omnetpp	473.astar
400.perlbench	1											
483.xalancbmk	-0.3	1										
403.gcc	-0.2	0.3	1									
429.mcf	0.8	-0.3	0.2	1								
401.bzip2	0.5	-0.1	-0.9	0.1	1							
445.gobmk	0.1	-0.1	0.9	0.5	-0.8	1						
456.hmmer	-0.5	0.1	0.9	-0.1	-1.0	0.8	1					
458.sjeng	-0.1	0.4	0.9	0.4	-0.7	0.8	0.7	1				
462.libquantum	0.2	0.0	0.8	0.7	-0.6	0.9	0.6	0.9	1			
464.h264ref	-0.5	0.1	0.9	-0.1	-1.0	0.8	1.0	0.7	0.6	1		
471.omnetpp	-0.2	0.3	1.0	0.2	-0.9	0.9	0.9	0.9	0.8	0.9	1	
473.astar	-0.1	0.6	-0.1	0.1	0.3	-0.3	-0.3	0.3	0.1	-0.3	-0.1	1

Table 7: Correlation matrix for SPEC CINT2006 on Westmere

	Harpertown	Nehalem, Westmere
CPI	INST_RETIRED.ANY	INST_RETIRED.ANY
	CPU_CLK_UNHALTED.REF	CPU_CLK_UNHALTED.REF
Stalls	RESOURCE_STALLS.LD_ST	RESOURCE_STALLS.LOAD
	RESOURCE_STALLS.BR_MISS_CLEAR	RESOURCE_STALLS.STORE
	RAT_STALLS.ANY	RAT_STALLS.ANY
	RESOURCE_STALLS.ROB_FULL	RESOURCE_STALLS.ROB_FULL
	RESOURCE_STALLS.RS_FULL	RESOURCE_STALLS.RS_FULL

Table 4: List of hardware counters sampled

For this work, we used the Intel Performance Tuning Utility (PTU) for the Harpertown and Nehalem architecture; we used the Intel VTune for Westmere architecture. Table 4 lists the set of counters (sampled) for each architecture. To assure the maximum performance, we disabled the dynamic voltage frequency scaling for all architectures and all experiments, so as not to influence the results. We selected counters corresponding to resource stalls as they directly affect CPI (refer to Equation 1 for the performance model).

We compiled all applications from SPEC CINT2006 with Intel C/C++ compiler using the `-fast` option. We ran the SPEC CINT2006 applications with the reference input data set.

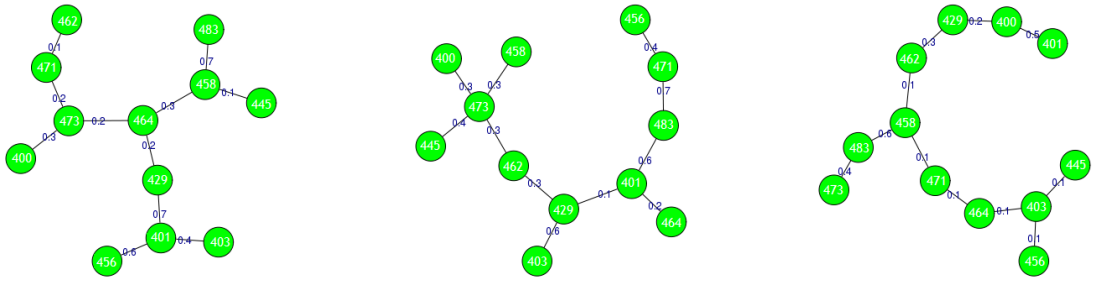
4.1 Effect of sampling interval and sampling period

In order to select the sampling interval, s_i, and the sampling period, s_p, we performed sensitivity analysis with different ratios s_i/s_p (refer to Figure 2). The ratios considered in this analysis are: $60\% = s_i/s_p(\%) = 3/5 \times 100$, $50\% = 5/10 \times 100$, and $20\% = 30/60 \times 100$. We compute the sample mean and sample variance of the CPI for each ratio $s_i/s_p \times 100$ (from the sample time-series). The variation in sample mean between the first and the second ratio (60% and 50% respectively) is 6% on an average and the variation in sample mean between the second and the third ratio (50% and 20% respectively) is 9% on an average. This suggests that the bias due to sampling artifacts is low on an average. We selected the first sampling ratio so as to obtain a larger number of elements per sample.

4.2 Similarity analysis

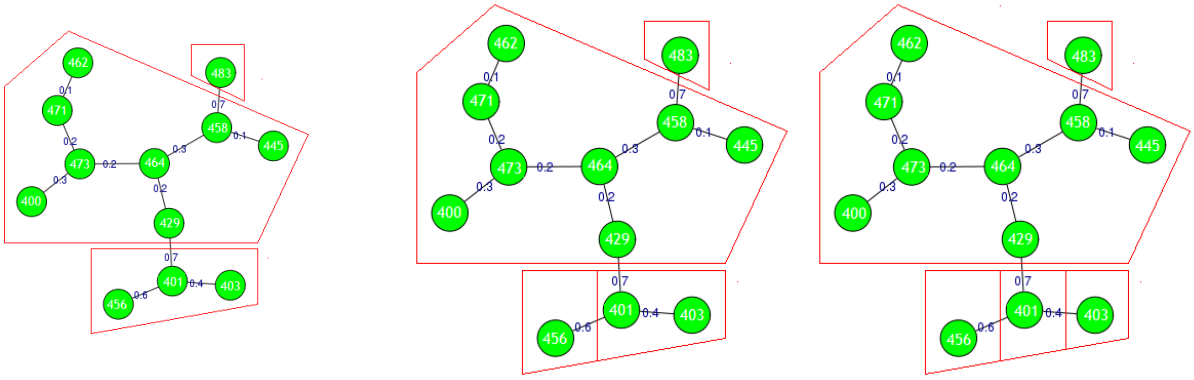
Table 1 reports the application signatures for the applications in SPEC CINT2006, for the three micro-architectures listed in Table 2.

Tables 5, 6 and 7 report the pairwise similarity between SPEC CINT2006 applications on Harpertown, Nehalem and Westemere. We carried out clustering using both Algo-

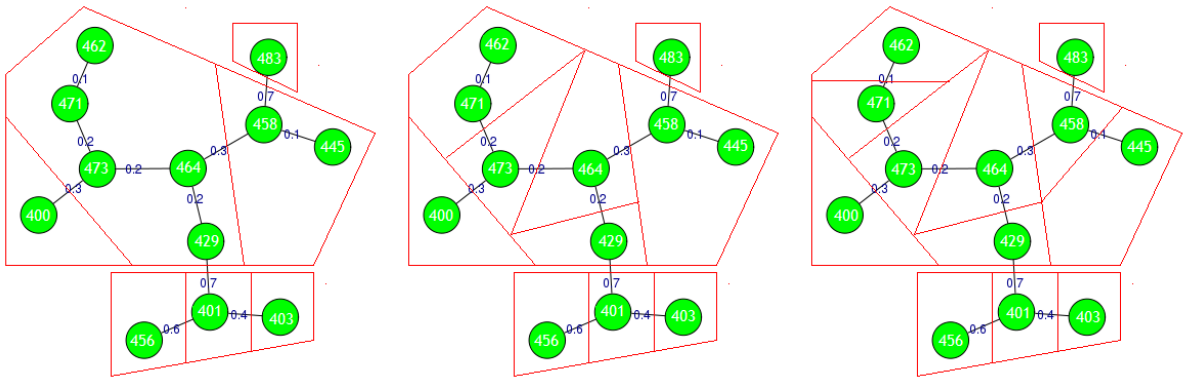


(a) Harpertown (b) Nehalem (c) Westmere

Figure 5: Minimum spanning tree for SPEC CINT2006



(a) Step 1 (b) Step 2 (c) Step 3



(d) Step 4 (e) Step 5 (f) Step 6

Figure 6: Harpertown. Recursive maximum length cut

rithm 1 and Algorithm 2. Intuitively, clusters are group of nodes of the MST that are similar to each other.

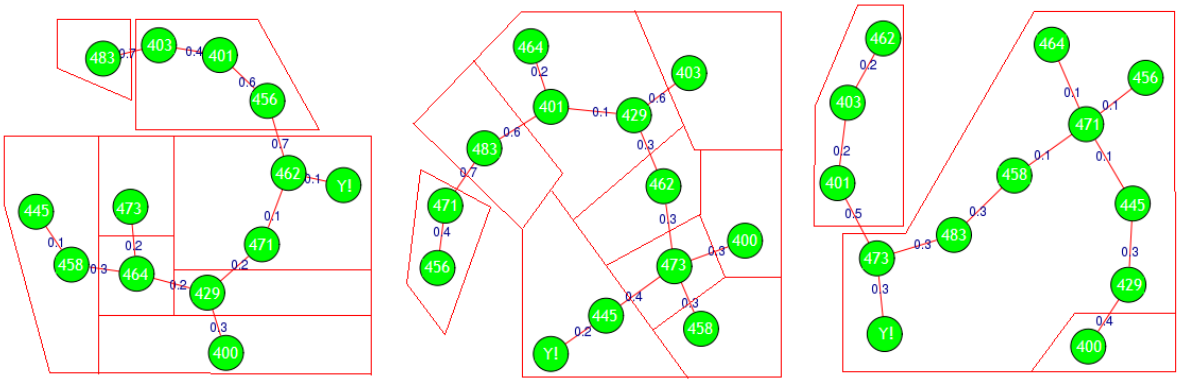
In Figure 5, we show the spanning trees associated to the correlation matrices in Tables 5, 6, and 7. We associate every node with the application number as in SPEC CINT2006.

In Figure 6, we show step by step application of Algorithm 1 to the MST shown in Figure 5(a). After the first call of the procedure *recursive_max_edge_cut* in Algorithm 1, two edges are cut thereby yielding two clusters. One cluster contains only one application – `483.xalancbmk`. Notice that `483.xalancbmk` is the only application of SPEC CINT2006 for which CPI is highly correlated with reservation station and load/store queue utilization. We can see this by its rank vector:

$$\pi(\mathbf{v}_{483}) = (4, 2, 1, 3, 5)$$

where the components of the vector, from left to right, correspond to: reservation station stalls, load store queue stalls, reorder buffer stalls, branch misprediction stalls, and register alias table stalls. Another cluster contains `456.hmmer`, `401.bzip2`, and `403.gcc`. CPI of these three applications is highly correlated with branches misprediction and reservation station stalls, as indicated with the rank vectors:

$$\pi(\mathbf{v}_{401}) = (3, 5, 2, 1, 4)$$



(a) Harpertown (b) Nehalem (c) Westmere

Figure 7: Search of a minimum cluster containing a reference application Y

$$\pi(\mathbf{v}_{403}) = (1, 4, 2, 3, 5)$$

$$\pi(\mathbf{v}_{456}) = (3, 2, 4, 1, 5)$$

However, the correlation among applications in this cluster is low. The low correlation between them drives the subsequent two iterations of *recursive_max_edge_cut*. The cluster is partitioned in two and three sets respectively. After the first step, the biggest cluster is composed of applications for which CPI performance is highly correlated with load/store queue, branch misprediction, and register alias table stalls. For instance, the following rank vectors

$$\pi(\mathbf{v}_{471}) = (5, 3.5, 3.5, 1.5, 1.5)$$

$$\pi(\mathbf{v}_{473}) = (5, 4, 3, 2, 1)$$

$$\pi(\mathbf{v}_{462}) = (5, 3, 4, 1, 2)$$

$$\pi(\mathbf{v}_{400}) = (2, 5, 4, 3, 1)$$

are in the same cluster after the first step, because their CPI is correlated with stalls due to branch misprediction. After the fourth step, `400.perlbench` is isolated into a separate cluster. After the fifth step, `471.astar` and `462.libquantum` with the rank vectors

$$\pi(\mathbf{v}_{471}) = (5, 3.5, 3.5, 1.5, 1.5)$$

$$\pi(\mathbf{v}_{462}) = (5, 3, 4, 1, 2)$$

fall in the same cluster and `458.sjeng` and `445.gobmk` with the rank vectors

$$\pi(\mathbf{v}_{458}) = (5, 2, 3, 4, 1)$$

$$\pi(\mathbf{v}_{445}) = (5, 2, 4, 3, 1)$$

fall in the same cluster.

For the Intel's Harpertown architecture, applications in the former cluster are load/store and control intensive; applications in the latter group are load/store intensive and with high register pressure.

We repeated the same process for the other two micro-architectures. For Intel's Nehalem, `429.mcf, 401.bip2` and `464.h264ref` are clustered together as their CPI is highly correlated with load/store queue stalls and reservation station stalls. The application `445.gobmk`, is isolated after the third iteration of Algorithm 1. CPI of `445.gobmk` is highly correlated with load/store queue stalls and register alias table stalls. For Intel's Westmere, a large group of applications – `458.sjeng`, `471.omnetpp`, `464.h264ref`, `403.gcc`, `456.hmmer` and `445.gobmk` – – are clustered together as their CPI is highly correlated with the reorder buffer and reservation station stalls. This can be seen by comparing the following rank vectors:

$$\pi(\mathbf{v}_{471}) = (2.5, 1, 2.5, 4.5, 4.5)$$

and

$$\pi(\mathbf{v}_{448}) = (1, 2, 3, 4, 5)$$

In case of Westmere, `473.astar` is the only application wherein CPI is highly correlated with store-queue, and reservation-station stalls:

$$\pi(\mathbf{v}_{473}) = (1, 5, 3, 2, 4)$$

On Nehalem and Westmere, the effect of each resource stall on achieved performance is smaller than it is on Harpertown. This stems from the fact that the components of signature vector are lower in magnitude on Nehalem and Harpwertown as compared to Harpwertown. This is primarily due to the improvements in the memory subsystem from Harpertown to Nehalem and Westemere.

Remarks The application of our methodology using Algorithm 1 orchestrates the identification of applications that stress certain micro-architectural resources. Our methodology is unsupervised, i.e., it does not make any assumption on the nature of the applications under study. For example, in our experiments, we considered SPEC CINT2006 as black boxes whose dynamic behavior was captured within the signature by measuring and composing hardware performance counters.

4.2.1 *Similarity with a reference application*

In this section, we apply Algorithm 2 to find a subset of SPEC CINT2006 similar to one Yahoo! property, Y. This application is known to expose a large memory footprint and to be highly control intensive. Clusters obtained with Algorithm 2 are compared against clusters obtained with k-means, using $k = 3, 5$. K-means is a well known clustering algorithm often used in prior work in conjunction to principal component analysis (see [6, 10, 7]).

Figure 7 shows the clustering with the reference Y! application. The addition of the reference application to SPEC CINT2006, modifies the similarity graph which in turn modifies the topology of the MST. The application of Algorithm 2 classifies the application as similar to others on the basis of rank associated to signature vectors.

	Harpertown	Nehalem	Westmere
Algorithm 2	Y, 462.libquantum, 471.omnetpp	Y, 445.gobmk	Y, 473.astat, 483.xalancbmk 458.sjeng, 471.omnetpp, 464.h264ref 445.gobmk, 456.hmmer, 429.mcf
3-means	Y, 400.perlbench, 429.mcf 462.libquantum, 464.h264ref, 471.omnetpp 473.astar	Y, 400.perlbench, 462.libquantum, 445.gobmk, 458.sjeng, 473.astar	Y, 473.astar, 483.xalancbmk
5-means	Y, 462.libquantum, 471.omnetpp	Y, 445.gobmk	Y, 473.astar, 483.xalancbmk

Table 8: Comparison Algorithm 2 vs {3,5}-means

Note that for each micro-architecture, the reference application is placed in a cluster with applications from SPEC CINT2006 for which CPI is highly correlated with similar resource stalls – load/store queue stalls and branch misprediction stalls in the case of Harpertown, store queue stalls and reservation station stalls in the case of Nehalem and Westmere.

A comparison between our algorithm, 3-means and 5-means (we use Hartigan and Wong algorithm [30]) shows that 5-means and our algorithms provide similar partitioning (see Table 8), whereas 3-means yields larger clusters. The advantage of using our methodology is that our MST-based algorithm shows topological property of out similarity approach. In fact, applications experiencing common runtime issues when running on a given micro-architecture are clustered together; the application signatures reveal the type of issues; the rank-based similarity between multiple applications reveals the nearness of one application to others.

4.3 System Evaluation

In this section we first show how our methodology is suitable to identify and reject micro-architectures for which a decrease in performance of similar applications is likely to imply a decrease in performance of the reference application. Next, we prune the evaluation space – comprised of Harpertown, Nehalem and Westmere – with respect to a reference Y! application. The micro-architectures under evaluation are referred as *candidate systems*, whereas the micro-architecture currently in use is referred as *current system*.

For each architecture and for each benchmark, we apply Algorithm 2 to identify similar applications. For each benchmark, we compute the speedup of similar applications on the two *candidate systems*. Next, for each *candidate system*, we count the number of times when the speedup of similar applications is greater (less) than one and the speedup of the reference application greater (less) than one. We refer to this case as *correct prediction*. Next, we count the number of times when the speedup of similar applications is greater (less) than one and the speedup of the reference application that is less (greater) than one. We refer to this case as *incorrect prediction*. Finally, we count the number of clusters containing applications whose speedup is either greater or less than one. We refer to this case as *unpredictable*. The result for the aforementioned cases are presented in Table 9, where the count is expressed in percentage over the number of cases considered.

Current to *Candidate* system	% *correct prediction*	% *incorrect prediction*	% *unpredictable*
Harpertown to *Nehalem*	33.33	33.33	33.33
Harpertown to *Westmere*	100	0	0
Nehalem to *Harpertown*	50	16	34
Nehalem to *Westmere*	100	0	0
Westmere to *Harpertown*	100	0	0
Westmere to *Nehalem*	100	0	0

Table 9: Validation table

When Harpertown is the current architecture, and Nehalem is under evaluation, the same rate of *correct prediction*, *incorrect prediction*, and *unpredictable* is recorded. Westmere has the highest rate of *correct prediction* (100%), as all applications gain performance. Therefore Westemere a good candidate, how- ever we cannot exclude Nehalem. When Nehalem is the current architecture, and Harpertown is under evaluation, the method has a better rate of *correct prediction*. In particular, we record 50% of correct prediction, 20% of correct prediction, and 30% cannot be predicted with the current method. In each case, Westmere has the highest percentage of *correct prediction* (100%). When we apply the methodology proposed to predict speedup of the reference Y! application, using only the most similar application in SPEC CINT2006, the prediction resutls always correct. Finally, we notice that when Harpertown is the current architecture, the average improvement in performance of applications similar to Y! on both Nehalem (23%) and Wesmere (39%) is commensurate with the improvement in performance of our Y! application. In contrast, the performance improvement of the entire SPEC CINT2006 is 6% and 33% on Nehalem and Westmere respectively.

5. RELATED WORK

The assesment of program similarity is a fundamental problem for the design/selection of benchmarks and for performance measurement/prediction (see [6, 4, 31, 32, 33]). In prior work different single- and multi-dimentional features have been used to capture similarity between applications. Similarity has been subsequently used to create benchmark suites satisfying certain requirements; to reduce the number of benchmarks of a given benchmark suite and their the input, with the goal of reducing simulation time; to build models to predict application performance.

Dujmovic et al. [34] used normalized execution time as application signature and proposed a methodology to design benchmark suites from the SPEC CPU benchmark suite. Hoste et al. built a prediction model using a large set of application specific features as application signature. Instructions mix, instructions level parallelism, working set, data stream strides and branch predictor predictability [7]. In [35], Wunderlich et al. proposed a framework to predict performance metrics as cycles per instruction, energy per instruction with fast and cycle accurate simulations of out-of-order architectures, where a sample of the whole application is simulated. We adopt a similar sampling approach to extract samples of CPI and pipeline stalls the execution of SPEC CINT2006 on real hardware.

In [10], Phansalkar et al. used micro-architecture independent features of an application such as instructions mix to predict performance. A similar approach to reduce the size of the input dataset is proposed by KleinOsowski et al. in [36].

In this paper we derive the feature selection from a simple performance model which focuses on pipeline stalls [19]. Arguably, more complex performance models, including other features such as cache miss count, working set, I/O and power consumption metrics, can be considered, but this is out of the scope of this paper.

In some prior work, e.g., [6, 4, 10, 36], Principal Components Analysis (PCA) is used to transform features of an application in principal components, which are statistically incorrelated. The largest (usually 2) principal components are retained, and are subsequently used to cluster applications. The use of PCA has several drawbacks, since each principal components is a linear combination of features. As such we cannot know how the resulting principal compents are composed.

In this paper we decide not to use PCA since we require our clustering strategy to be suitable as analysis tool, i.e., able to assign applications to clusters because of their performance issues. Estimation of CPI is also out of the scope of this paper. Since we only want to prune the hardware evaluation space, the quantitative evaluation of any degradation of performance is not relevant to us.

6. CONCLUSION AND FUTURE WORK

In this paper we propose a novel similarity-based methodology for system selection. Our methodology prunes the set of candidate systems by eliminating the systems that are likely to reduce performance of a given proprietary application. The key highlights of the proposed methodology are the novel definition of application signature and the way similarity between applications is assessed.

We defined an application signature as a vector of Pearson's correlation coefficients between cycles per instructions and different types of resource stalls. Each component of the signature captures the association between performance, as measured by cycles per instruction and resource stalls. Similarity between two applications is defined as the Spearman's correlation between their respective signatures. We use Spearman's correlation to quantify the association between two signatures, hence to assess similarity between applications. In fact Spearman's correlation is sensitive to the difference in terms of rank ordering of the components of two signatures.

Lastly, we proposed two algorithms based on minimum spanning tree to cluster similar applications. The first algorithm is used to identify clusters of similar applications in a given benchmark suite. The second algorithm is used to determine applications that are similar to a given application of interest.

The advantages of our approach are (1) similar applications can (yet do not need to) belong to standard benchmark suites such as SPEC CPU2006, whose performance is studied in detail for almost any relevant architecture and system configuration; (2) the pruning process relies only on similar applications, without the need to run the application of interest on all the new architectures.

We evaluated three different Intel micro-architectures targeted to run server applications. We demonstrated how the proposed methodology is able to prune the system evaluation space.

As future work, we plan to develop techniques to guide compiler optimization using the correlation-based application similarity analysis as defined and developed in this work.

7. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grant NSF CCF-0811882, and Yahoo! Inc.

8. REFERENCES

[1] Intel's Tick-Tock Model. http://www.intel.com/technology/tick-tock/index.htm.

[2] D. Ferrari. Workload charaterization and selection in computer performance measurement. *Computer*, 5:18–24, 1972.

[3] D. Ferrari. Characterizing a workload for the comparison of interactive services. *Managing Requirements Knowledge, International Workshop on*, page 789, 1979.

[4] M. Calzarossa and D. Ferrari. A sensitivity study of the clustering approach to workload modeling. In *SIGMETRICS*, pages 38–39, 1985.

[5] H. Eeckhout, L. Vandierendonck and K. De Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction Level Parallelism*, 5:1–33, 2003.

[6] D. Ferrari. On the foundations of artificial workload design. In *SIGMETRICS*, pages 8–14, 1984.

[7] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L.K. John, and K. De Bosschere. Performance prediction based on inherent program similarity. In *PACT*, pages 114–122, 2006.

[8] A. Phansalkar, A. Joshi, and L.K. John. Subsetting the SPEC CPU2006 benchmark suite. *SIGARCH Comput. Archit. News*, 35(1):69–76, 2007.

[9] H. Vandierendonck and K. De Bosschere. Many benchmarks stress the same bottlenecks. In *Proc. of the Workshop on Computer Architecture Evaluation using Commerical Workloads, pp. 57-71*, 2004.

[10] A. Phansalkar, A. Joshi, L. Eeckhout, and L.K. John. Measuring program similarity: Experiments with spec cpu benchmark suites. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, pages 10–20, Washington, DC, USA, 2005.

[11] A. Joshi, A. Phansalkar, L. Eeckhout, and L.K. John. Measuring benchmark similarity using inherent program characteristics. *IEEE Trans. Comput.*, 55(6):769–782, 2006.

[12] SPEC CPU2006. http://www.spec.org/cpu2006/.

[13] K. Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arised from random sampling. *Philosophical Magazine Series 5*, 50:157–175, 1900.

[14] J. Neyman and E. S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference. *Biometrika*, 20:175–240, 1928.

[15] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15:72–101, 1904.

[16] C. Spearman. 'footrule' for measuring correlation. *British Journal of Psychology*, 2(1):89–108, 1906.

[17] S. Eyerman, L. Eeckhout, T. Karkhanis, and J.E. Smith. A top-down approach to architecting cpi component performance counters. *IEEE Micro*, 27(1):84–93, 2007.

[18] R. Azimi, M. Stumm, and R.W. Wisniewski. Online performance analysis by statistical sampling of microprocessor performance counters. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 101–110, New York, NY, USA, 2005.

[19] D. Levinthal. Intel core i7 and Intel Xeon 5500 microarchitecture, optimization and performance analysis. Technical report, Intel Corporation, 2008-2009.

[20] B. Sprunt. The basics of performance-monitoring hardware. *IEEE Micro*, 22(4):64–71, 2002.

[21] V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J.C. Sexton, and R. Walkup. Next-generation performance counters: Towards monitoring over thousand concurrent events. In *ISPASS*, pages 139–146, 2008.

[22] K. Shen., M. Zhong, S. Dwarkadas, C. Li, C. Stewart, and X. Zhang. Hardware counter driven on-the-fly request signatures. In *ASPLOS*, pages 189–200, 2008.

[23] S. Siegel and N.J. Castellan. *Nonparametric statistics for the behavioral sciences*. Second edition, 1988.

[24] M. Kendall. A new measure of rank correlation. *Biometrika*, 30:81–89, 1938.

[25] R.L. Diaconis, P. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society*, 39:262–268, 1977.

[26] R.C. Prim. *Bell System Technology Journal*, 36:1389–1401, 1957.

[27] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

[28] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[29] SPEC CINT2006. http://www.spec.org/cpu2006/CINT2006/.

[30] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[31] J.L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *IEEE Computer*, 33(7):28–35, 2000.

[32] J.L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006.

[33] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[34] J.J. Dujmovic and I. Dujmovic. Evolution and evaluation of spec benchmarks. *SIGMETRICS Perform. Eval. Rev.*, 26(3):2–9, 1998.

[35] R.E. Wunderlich, T.F. Wenisch, B. Falsafi, and J.C. Hoe. Smarts: accelerating microarchitecture simulation via rigorous statistical sampling. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 84 – 95, 9-11 2003.

[36] A.J. KleinOsowski and D.J. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. *IEEE Comput. Archit. Lett.*, 1(1), 2002.