Using Wavelet Domain Workload Execution Characteristics to Improve Accuracy, Scalability and Robustness in Program Phase Analysis

Chang-Burn Chø and Tao Li

Intelligent Design of Efficient Architecture Lab (IDEAL)
Department of ECE, University of Florida
choreno@ufl.edu; taoli@ece.ufl.edu

Abstract
Program phase analysis has many applications in computer architecture design and optimization. Recently, there has been a growing interest in employing wavelets as a tool for phase analysis. Nevertheless, the examined scope of workload characteristics and the explored benefits due to wavelet-based analysis are quite limited. This work further extends prior research by applying wavelets analysis to abundant types of program execution statistics and quantifying the benefits of wavelet analysis in terms of accuracy, scalability and robustness in phase classification.

Experimental results on SPEC CPU 2000 benchmarks show that compared with methods that work in the time domain, wavelet domain phase analysis achieves higher accuracy and exhibits superior scalability and robustness. We examine and contrast the effectiveness of applying wavelets to a wide range of runtime workload execution characteristics. We find that wavelet transform significantly reduces temporal dependence in the sampled workload statistics and therefore simple models which are insufficient in the time domain become quite accurate in the wavelet domain. More attractively, we show that different types of workload execution characteristics in wavelet domain can be assembled together to further improve phase classification accuracy. For long-running, complex and real-world workloads, a scalable phase analysis technique is essential to capture the manifested large-scale program behavior. In this study, we show that such scalability can be achieved by applying wavelet analysis of high dimension sampled workload statistics to alleviate the counter overflow problem which can negatively affect phase classification accuracy. By exploiting the wavelet denoising capability, we show in this paper that phase classification can be performed robustly under program execution variability. To our knowledge, this work presents the first effort on using wavelets to improve scalability and robustness in phase analysis.

1. Introduction
As computer architectures become more complex, their efficiency increasingly depends on the runtime, dynamic behavior of the workloads. Program execution exhibits time varying phase behavior: workload execution manifests similar behavior within each phase while showing distinct characteristics between different phases. Many challenges related to the design, analysis and optimization of computer architectures (e.g. multithread scheduling, feedback-directed optimizations, power management and design space exploration) can be efficiently solved by exploiting program phases [1, 6-9]. For this reason, there is a growing interest in studying program phase behavior [2-4, 7, 10-19].

Several phase analysis techniques have been proposed [1, 7, 8, 13, 15]. Among these, phase classification techniques that use architecture independent structures (e.g. basic block vector [1], program working set [7], and subroutine call boundary [9]) have been shown to have the capability of capturing workload characteristics across different architecture configurations. This feature is valuable in architecture design space exploration where a large amount of configurations need to be quickly examined.

A complementary method to approaches that use architecture independent structures for phase classification is to use workload execution statistics that a program generates at runtime to infer its phase behavior. In this work, we focus on workload-statistics-based phase analysis since on a given machine configuration and environment, it is more suitable to identify how the targeted architecture features vary during program execution. In contrast, phase classification using program code structures lacks the capability of informing how workloads behave architecturally [13, 29]. Therefore, phase analysis using specified workload characteristics allows one to explicitly link the targeted architecture features to the classified phases. For example, if phases are used to optimize cache efficiency, the workload characteristics that reflect cache behavior can be used to explicitly classify program execution into cache performance/power/reliability oriented phases. Program code structure based phase analysis identifies similar phases only if they have similar code flow. There can be cases where two sections of code can have different code flow, but exhibit similar architectural behavior [13]. Code flow based phase analysis would then classify them as different phases. Another advantage of workload-statistics-based phase analysis is that when multiple threads share the same resource (e.g. pipeline, cache), using workload execution information to classify phases allows the capability of capturing program dynamic behavior due to the interactions between threads.

The key goal of workload execution based phase analysis is to accurately and reliably discern and recover phase behavior from various program runtime statistics represented as large-volume, high-dimension and noisy data. To effectively achieve this objective, recent work
[29, 30] proposes using wavelets as a tool to assist phase analysis. The basic idea is to transform workload time domain behavior into the wavelet domain. The generated wavelet coefficients which extract compact yet informative program runtime feature are then assembled together to facilitate phase classification. Nevertheless, in current work, the examined scope of workload characteristics and the explored benefits due to wavelet transform are quite limited. In this paper, we extend prior research by applying wavelets to abundant types of program execution statistics and quantifying the benefits of using wavelets for improving accuracy, scalability and robustness in phase classification. We conclude that wavelet domain phase analysis has the following advantages: 1) accuracy: the wavelet transform significantly reduces temporal dependence in the sampled workload statistics. As a result, simple models which are insufficient in the time domain become quite accurate in the wavelet domain. More attractively, wavelet coefficients transformed from various dimensions of program execution characteristics can be dynamically assembled together to further improve phase classification accuracy; 2) scalability: phase classification using wavelet analysis of high-dimension sampled workload statistics can alleviate the counter overflow problem which has a negative impact on phase detection. Therefore, it is much more scalable to analyze large-scale phases exhibited on long-running, real-world programs; and 3) robustness: wavelets offer denoising capabilities which allows phase classification to be performed robustly in the presence of workload execution variability.

The rest of this paper is organized as follows. Section 2 introduces the wavelet transform and describes how wavelet analysis can be applied to represent program behavior. Section 3 describes the experimental setup including the simulated machine configuration, studied workloads and evaluated metrics. Section 4 explores wavelet domain phase analysis using a wide range of workload execution statistics and compares its accuracy with methods that work in the time domain. Additionally, Section 4 investigates the use of wavelets to address scalability and workload variability in phase analysis. Section 5 discusses related work. Section 6 summarizes the paper and outlines our future work.

2. Background

In this work, we use wavelets as an efficient tool for capturing workload behavior. To familiarize the reader with general methods used in this paper, we provide a brief overview on wavelet analysis and show how program execution characteristics can be represented using wavelet analysis in this section. To learn more mathematical details of wavelets, a reader is encouraged to read [20].

2.1 Discrete Wavelet Transform

Wavelets are mathematical tools that use a prototype function (called the analyzing or mother wavelet) to transform data of interest into different frequency components, and then analyze each component with a resolution matched to its scale. Therefore, the wavelet transform is capable of providing a compact and effective mathematical representation of data. In contrast to Fourier transforms which only offer frequency representations, wavelets transforms provide time and frequency localizations simultaneously [21]. Wavelet analysis allows one to choose wavelet functions from numerous functions. In this section, we provide a quick primer on wavelet analysis using the Haar wavelet, which is the simplest form of wavelets [20].

Consider a data series $X_{n,k}, k = 0,1,2,...$, at the finest time scale resolution level $2^{-n}$ . This time series might represent a specific program characteristic (e.g., number of executed instructions, branch mispredictions and cache misses) measured at a given time scale. We can coarsen this event series by averaging (with a slightly different normalization factor) over non-overlapping blocks of size two

$$X_{n-1,k} = \frac{1}{\sqrt{2}} (X_{n,2k} + X_{n,2k+1}) \quad (1)$$

and generate a new time series $X_{n-1}$, which is a coarser granularity representation of the original series $X_n$. The difference between the two representations, known as details, is

$$D_{n-1,k} = \frac{1}{\sqrt{2}} (X_{n,2k} - X_{n,2k+1}) \quad (2)$$

Note that the original time series $X_n$ can be reconstructed from its coarser representation $X_{n-1}$ by simply adding in the details $D_{n-1}$, i.e.,

$$X_n = 2^{-1/2} (X_{n-1} + D_{n-1}) \quad .$$

We can repeat this process (i.e., write $X_{n-1}$ as the sum of yet another coarser version $X_{n-2}$ of $X_n$ and the details $D_{n-2}$, and iterate) for many scales in the original time series, i.e.,

$$X_n = 2^{-n/2} X_0 + 2^{-n/2} D_0 + ... + 2^{-1/2} D_{n-1}$$
This feature can be exploited to create concise yet informative fingerprints to capture program execution behavior. Wavelet coefficients usually contain the important trend. Although the DWT operations can produce as many wavelet coefficients as the original input data, the first few wavelet coefficients make up the so-called discrete wavelet transform (DWT). As can be seen, the DWT offers a natural hierarchy structure to represent data behavior at multiresolution levels: the first few wavelet coefficients contain an overall, coarser approximation of the data; additional coefficients illustrate high detail. This property can be used to capture workload execution behavior.

### 2.2 Apply DWT to Capture Workload Execution Behavior

Workload-statistics based phase classification often involves a process of automatic extraction of repeatable patterns from a large collection of data. Since variation of program characteristics over time can be viewed as signals, we apply discrete wavelet analysis to capture program execution behavior.

To obtain time domain workload execution characteristics, we break down entire program execution into intervals and then sample multiple data points within each interval. Therefore, at the finest resolution level, program time domain behavior is represented by a data series within each interval. Note that the sampled data can be any runtime program characteristics of interest. We then apply discrete wavelet transform (DWT) to each interval. As described in Section 2.1, the result of DWT is a set of wavelet coefficients which represent the behavior of the sampled time series in the wavelet domain. Although the DWT operations can produce as many wavelet coefficients as the original input data, the first few wavelet coefficients usually contain the important trend. This feature can be exploited to create concise yet informative fingerprints to capture program execution behavior.

One advantage of using wavelet coefficients to fingerprint program execution is that program time domain behavior can be reconstructed from these wavelet coefficients. Figure 1 shows that the time domain workload characteristics can be recovered using the inverse discrete wavelet transforms. In Figure 1 (a)-(e), the first 1, 2, 4, 8, and 16 wavelet coefficients were used to restore program time domain behavior with increasing fidelity. As shown in Figure 1 (f), when all (e.g., 1024) wavelet coefficients are used for recovery, the original signal can be completely restored. However, this could involve storing and processing a large number of wavelet coefficients. Using a wavelet transform gives time-frequency localization of the original data. As a result, most of the energy of the input data can be represented by only a few wavelet coefficients. As can be seen, using 16 wavelet coefficients can recover program time domain behavior with sufficient accuracy. Therefore, in the rest of this paper, we choose 16 as the maximum number of wavelet coefficients that we can use.

### 3. Experimental Setup

Using the above described wavelet-based method, we explore program phase analysis on a high-performance, out-of-order execution superscalar processor coupled with a multi-level memory hierarchy. We use Daubechies wavelet [20] with an order of 8 for the rest of the experiments due to its high accuracy and low computation overhead. This section describes our experimental methodologies, the simulated machine configuration, experimented benchmarks and evaluated metrics.

We performed our analysis using twelve SPEC CPU 2000 integer benchmarks: bzip2, crafty, eon, gap, gcc, gzip, mcf, parser, perlbmk, twolf, vortex and vpr. All programs were run with the reference input to completion. The runtime workload execution statistics were measured on the SimpleScalar 3.0 [22] sim-outorder simulator for the Alpha ISA. The baseline microarchitecture model we used is detailed in Table 1.
We use IPC (instruction per cycle) as the metric to evaluate the similarity of program execution within each classified phase. To quantify phase classification accuracy, we use the weighted COV metric proposed by Calder et al. [15]. After classifying all program execution intervals into phases, we examine each phase and compute the IPC for all the intervals in that phase. We then calculate the standard deviation in IPC within each phase, and we divide the standard deviation by the average to get the Coefficient of Variation (COV). We then calculate an overall COV metric for a phase classification method by taking the COV of each phase and weighting it by the percentage of execution that the phase accounts for. This produces an overall metric (i.e. weighted COVs) used to compare different phase classifications for a given program. Since COV measures standard deviations as a percentage of the average, a lower COV value means a better phase classification technique.

4. Exploring Wavelet Domain Phase Analysis

In this section, we first evaluate the efficiency of wavelet analysis on a wide range of program execution characteristics by comparing its phase classification accuracy with methods that use information in the time domain. Next, we explore methods to further improve phase classification accuracy in the wavelet domain. The advantages of using wavelets to enhance scalability and robustness in phase analysis are discussed in subsections 4.2 and 4.3 respectively.

4.1 Phase Classification: Time Domain vs. Wavelet Domain

The wavelet analysis method described in Section 2 provides a cost-effective representation of program behavior. Since wavelet coefficients are generally decorrelated, we can transform the original data into the wavelet domain and then carry out the phase classification task. The generated wavelet coefficients can be used as signatures to classify program execution intervals into phases: if two program execution intervals show similar fingerprints (represented as a set of wavelet coefficients), they can be classified into the same phase. With our time domain phase analysis method, each program execution interval is represented by a time series which consists of 1024 sampled program execution statistics. We first apply random projection to reduce the data dimensionality to 16. We then use the k-means clustering algorithm [23] to classify program intervals into phases. This is similar to the method used by the popular Simpoint tool [1] where the basic block vectors (BBVs) are used as input. For the wavelet domain method, the original time series are first transformed into the wavelet domain using DWT. The first 16 wavelet coefficients of each program execution interval are extracted and used as the input to the k-means clustering algorithms. Figure 2 illustrates the above described procedure.

![Figure 2. Phase analysis methods that use time domain and wavelet domain information](image)

We investigated the efficiency of applying wavelet domain analysis on 10 different workload execution characteristics, namely, the numbers of executed loads (load), stores (store), branches (branch), the number of cycles a processor spends on executing a fixed amount of instructions (cycles), the number of branch misprediction (branch.miss), the number of L1 instruction cache, L1 data cache and L2 cache hits (il1_hit, dl1_hit and ul2_hit), and the number of instruction and data TLB hits (itlb_hit and dtlb_hit). Figure 3 shows the COVs of phase classifications in time and wavelet domains when each type of workload execution characteristic is used as an input. As can be seen, compared with using raw, time domain workload data, the wavelet domain analysis significantly improves phase classification accuracy and this observation holds for all the investigated workload characteristics across all the examined benchmarks.

This is because in the time domain, collected program runtime statistics are treated as high-dimension time series data. By transforming program runtime statistics into the wavelet domain, workload behavior can be represented by a series of wavelet coefficients which are much more compact and efficient than its counterpart in the time domain. The wavelet transform significantly reduces temporal dependence and therefore simple models which are insufficient in the time domain become quite accurate in the wavelet domain.
Our objective is to improve wavelet domain phase classification accuracy across different program characteristics. As shown in Figure 4, a DWT is applied to each type of workload characteristic. The generated wavelet coefficients from different categories can be assembled together to form a signature for a data clustering algorithm.

Figure 3 shows that in the wavelet domain, the efficiency of using a single type of program characteristic to classify program phases can vary significantly across different benchmarks. For example, while ul2_hit achieves accurate phase classification on the benchmark vortex, it results in a high phase classification COV on the benchmark gcc. To overcome the above disadvantages and to build phase classification methods that can achieve high accuracy across a wide range of applications, we explore using wavelet coefficients derived from different types of workload characteristics. As shown in Figure 4, a DWT is used to generate wavelet coefficients from different categories. The generated wavelet coefficients from different categories can be assembled together to form a signature for a data clustering algorithm.

Figure 4. Phase classification using hybrid wavelet coefficients from different workload statistics

Our objective is to improve wavelet domain phase classification accuracy across different programs while using an equivalent amount of information to represent program behavior. As mentioned in Section 2, we choose a set of 16 wavelet coefficients as the phase signature since it provides sufficient precision in capturing program dynamics when a single type of program characteristic is used. If a phase signature can be composed using multiple workload characteristics, there are many ways to form a 16-dimension phase signature. For example, a phase signature can be generated using one wavelet coefficient from 16 different workload characteristics (16×1), or it can be composed using 8 workload characteristics with 2 wavelet coefficients from each type of workload characteristic (8×2). Alternatively, a phase signature can be formed using 4 workload characteristics with 4 wavelet coefficients each and 2 workload characteristics with 8 wavelet coefficients each (4×4, and 2×8) respectively. We extend the 10 workload execution characteristics (shown in Figure 5) to 16 by adding the following events: the number of accesses to instruction cache (il1_access), data cache (dl1_access), L2 cache (ul2_access), instruction TLB (itlb_access) and data TLB (dtlb_access). To understand the trade-offs in choosing different methods to generate hybrid signatures, we did an exhaustive search using the above 4 schemes on all benchmarks to identify the best COVs that each scheme can achieve. The results (their ranks in terms of phase classification accuracy and the COVs of phase analysis) are shown in Table 2. As can be seen, statistically, hybrid wavelet signatures generated using 16 (16×1) and 8 (8×2) workload characteristics achieve higher accuracy. This suggests that combining multiple dimension wavelet domain workload characteristics to form a phase signature is beneficial in phase analysis.

We further compare the efficiency of using the 16×1 hybrid scheme (Hybrid), the best case that a single type workload characteristic can achieve (Individual Best) and the Simpoint based phase classification that uses basic block vector (BBV). The results of the 12 SPEC integer benchmarks are shown in Figure 5. As can be seen, the Hybrid outperforms the Individual Best on 10 out of the 12 benchmarks. The Hybrid also outperforms the BBV based Simpoint method on 10 out of the 12 cases.
above procedure. Figure 6 (a) describes the dimension accumulative counter values to perform phase analysis and calculate the COVs. Figure 6 (b) shows the n-bit sampling counter.

### Table 2. Efficiency of Different Hybrid Wavelet Signatures in Phase Classification

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Hybrid Wavelet Signature and its Phase Classification COV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rank 1</td>
</tr>
<tr>
<td>bzip2</td>
<td>16×1 6.5%</td>
</tr>
<tr>
<td>crafty</td>
<td>4×4 1.2%</td>
</tr>
<tr>
<td>eon</td>
<td>8×2 2.1%</td>
</tr>
<tr>
<td>gap</td>
<td>4×4 4.2%</td>
</tr>
<tr>
<td>gcc</td>
<td>8×2 2.7%</td>
</tr>
<tr>
<td>gzip</td>
<td>16×1 2.5%</td>
</tr>
<tr>
<td>mcf</td>
<td>16×1 9.5%</td>
</tr>
<tr>
<td>parser</td>
<td>16×1 4.7%</td>
</tr>
<tr>
<td>perlbmk</td>
<td>8×2 0.7%</td>
</tr>
<tr>
<td>twolf</td>
<td>16×1 0.2%</td>
</tr>
<tr>
<td>vortex</td>
<td>16×1 4.4%</td>
</tr>
<tr>
<td>vpr</td>
<td>16×1 1.3%</td>
</tr>
</tbody>
</table>

Figure 5. Phase classification accuracy of using 16×1 hybrid scheme (Hybrid), the best case that uses single type workload characteristic (Individual_Best) and the Simpoint based phase classification that uses basic block vector (BBV).

### 4.2 Scalability

Above we can see that wavelet domain phase analysis can achieve higher accuracy. In this subsection, we address another important issue in phase analysis using workload execution characteristics: scalability. Counters are usually used to collect workload statistics during program execution. The counters may overflow if they are used to track large scale phase behavior on long running workloads. Today, many large and real world workloads can run days, weeks or even months before completion and this trend is likely to continue in the future. To perform phase analysis on the next generation of computer workloads and systems, phase classification methods should be capable of scaling with the increasing program execution time.

To understand the impact of counter overflow on phase analysis accuracy, we use 16 accumulative counters to record the 16-dimension workload characteristic described in subsection 4.1. The values of the 16 accumulative counters are then used as a signature to perform phase classification. We gradually reduce the number of bits in the accumulative counters. As a result, counter overflows start to occur. We use two schemes to handle a counter overflow. In our first method, a counter saturates at its maximum value once it overflows. In our second method, the counter is reset to zero after an overflow occurs. After all counter overflows are handled, we then use the 16-dimension accumulative counter values to perform phase analysis and calculate the COVs. Figure 6 (a) describes the above procedure.

Our counter overflow analysis results are shown in Figure 7. Figure 7 also shows the counter overflow rate (e.g. percentage of the overflowed counters) when counters with different sizes are used to collect workload statistics within program execution intervals. For example, on the benchmark crafty, when the number of bits used in counters is reduced to 20, 100% of the counters overflow.

![Large Scale Phase Interval](image)

Figure 6. Different methods to handle counter overflows.

For the purpose of clarity, we only show a region within which the counter overflow rate is greater than zero and
less than or equal to one. Since each program has different execution time, the region varies from one program to another. As can be seen, counter overflows have negative impact on phase classification accuracy. In general, COVs increase with the counter overflow rate. Interestingly, as the overflow rate increases, there are cases that overflow handling can reduce the COVs. This is because overflow handling has the effect of normalizing and smoothing irregular peaks in the original statistics.

One solution to avoid counter overflows is to use sampling counters instead of accumulative counters, as shown in Figure 6 (b). However, when sampling counters are used, the collected statistics are represented as time series that have a large volume of data. The results shown in Figure 3 suggest that directly employing runtime samples in phase classification is less desirable. To address the scalability issue in characterizing large scale program phases using workload execution statistics, wavelet based dimensionality reduction techniques can be applied to extract the essential features of workload behavior from the collected statistics. The observations made in subsection 4.1 motivate the use of DWT to absorb large volume sampled raw data and produce highly efficient wavelet domain signatures for phase analysis, as shown in Figure 6 (b).

Figure 7 further shows phase analysis accuracy after applying wavelet techniques on the sampled workload statistics using sampling counters with different sizes. As can be seen, sampling enables using counters with limited size to study large program phases. In general, sampling can scale up naturally with the interval size as long as the sampled values do not overflow the counters. Therefore, with an increasing mismatch between phase interval and counter size, the sampling frequency is increased, resulting in an even higher volume sampled data. Using wavelet domain phase analysis can effectively infer program behavior from a large set of data collected over a long time span, resulting in low COVs in phase analysis.

![Figure 7. Impact of counter overflows on phase analysis accuracy (Two schemes – Sat and Reset are used to avoid counter overflows). The scheme that uses wavelet analysis of sampling counters is labeled as Wavelet.](image)

### 4.3 Workload Variability

As described earlier, our methods collect various program execution statistics and use them to classify program execution into different phases. Such phase classification generally relies on comparing the similarity of the collected statistics. Ideally, different runs of the same code segment should be classified into the same phase. Existing phase detection techniques assume that workloads have deterministic execution. On real systems, with operating system interventions and other threads, applications manifest behavior that is not the same from run to run. This variability can stem from changes in system state that alter cache, TLB or I/O behavior, system calls or interrupts, and can result in noticeably different timing and performance behavior [18, 24]. This cross-run variability can confuse similarity based phase detection. In order for a phase analysis technique to be applicable on real systems, it should be able to perform robustly under variability.

Program cross-run variability can be thought of as noise which is a random variance of a measured statistic. There are many possible reasons for noisy data, such as
measurement/instrument errors and interventions of the operating systems. Removing this variability from the collected runtime statistics can be considered as a process of denoising. In this paper, we explore using wavelets as an effective way to perform denoising. Due to the vanishing moment property of the wavelets, only some wavelet coefficients are significant in most cases. By retaining selective wavelet coefficients, a wavelet transform could be applied to reduce the noise. The main idea of wavelet denoising is to transform the data into the wavelet basis, where the large coefficients mainly contain the useful information and the smaller ones represent noise. By suitably modifying the coefficients in the new basis, noise can be directly removed from the data. The general de-noising procedure involves three steps: 1) decompose: compute the wavelet decomposition of the original data; 2) threshold wavelet coefficients: select a threshold and apply thresholding to the wavelet coefficients; and 3) reconstruct: compute wavelet reconstruction using the modified wavelet coefficients. More details on the wavelet-based denoising techniques can be found in [31].

To model workload runtime variability, we use additive noise models and randomly inject noise into the time series that represents workload execution behavior. We vary the SNR (signal-to-noise ratio) to simulate different degree of variability scenarios. To classify program execution into phases, we generate a 16 dimension feature vector where each element contains the average value of the collected program execution characteristic for each interval. The k-mean algorithm is then used for data clustering. Figure 8 illustrates the above described procedure.

Figure 8. Method for modeling workload variability

We use the Daubechies-8 wavelet with a global wavelet coefficients thresholding policy to perform denoising. We then compare the phase classification COVs of using the original data, the data with variability injected and the data after we perform denoising. Figure 9 shows our experimental results. The SNR=20 represents scenarios with a low degree of variability and the SNR=5 reflects situations with a high degree of variability. As can be seen, introducing variability in workload execution statistics reduces phase analysis accuracy. Wavelet denoising is capable of recovering phase behavior from the noised data, resulting in higher phase analysis accuracy. Interestingly, on some benchmarks (e.g. eon, mcf), the denoised data achieve better phase classification accuracy than the original data. This is because in phase classification, randomly occurring peaks in the gathered workload execution data could have a deleterious effect on the phase classification results. Wavelet denoising smoothes these irregular peaks and make the phase classification method more robust.

Various types of wavelet denoising can be performed by choosing different threshold selection rules (e.g. rigrsure, heursure, sqtwolog and minimaxi), by performing hard (h) or soft (s) thresholding, and by specifying multiplicative threshold rescaling model (e.g. one, sln, and mln). We compare the efficiency of different denoising techniques that have been implemented into the MATLAB tool [32]. Due to the space limitation, only the results on benchmarks bzip2, gcc and mcf are shown in Figure 10. As can be seen, different wavelet denoising schemes achieve comparable accuracy in phase classification.

Figure 9. Effect of using wavelet denoising to handle workload variability in phase classification (assuming variability affects 50% of program execution intervals). The Daubechies-8 wavelet with a global wavelet coefficients thresholding policy is used to perform denoising.

Figure 10. Efficiency of different denoising schemes (Wavelet denoising schemes are represented as X:Y:Z, where X is threshold selection rule, Y specifies hard or soft thresholding and Z is threshold rescaling model. More details can be found in [32]).
5. Related Work

Prior research has considered a range of phase analysis techniques. Sherwood and Calder proposed the use of Basic Block Vectors as a metric to capture a program’s phase behavior [1, 6, 10, 17]. In [7, 11], program working set changes are used to detect phase changes. Isci and Martonosi [5, 12, 13] showed that hardware performance counters can be exploited for phase classification and prediction. The work [9] tracks procedure calls via a call stack to dynamically identify phase changes. These studies, however, have focused almost exclusively on analyzing phase behavior in the time domain.

Wavelets have been widely used in numerous fields of study including computer science and engineering [25, 26]. In [27], Joseph and Martonosi used wavelets to analyze and predict the change of processor voltage over time. In an earlier work [28], they used Fourier analysis to characterize the power behavior of programs. Recently, using wavelets to assist program phase analysis has started to gain popularity. In [29], wavelets were explored to analyze the phase behavior of memory bus accesses on commercial workloads. However, only a single type of program execution information (e.g. addresses of memory accesses) is examined. Therefore, the effectiveness of wavelet analysis on a wide range of program execution statistics remains largely unknown. Moreover, the authors made no attempt to optimize wavelet domain phase classification accuracy. In [30], the multiresolution analysis capability of wavelets was exploited to analyze phase complexity. In [4], Shen and Ding used wavelets as a filter to remove the gradual changes in a reuse-distance trace to identify locality phase in programs. The wavelet-based filtering is used to accurately determine the best place for phase markers, but is not used as a method for comparing similarity under program variability. In [18], Isci and Martonosi developed glitch/gradient filtering to refine phase transitions from sampling effects and used near-neighbor blurring to handle observed moderate time dilations. To our knowledge, there has been no study on using wavelets to improve scalability and robustness in phase classification.

6. Conclusions and Future Work

Modeling and analyzing workload behavior play a critical role in the design, optimization and management of complex computer systems. For this reason, program phase classification is of growing interest in computer architecture research. Recently, several wavelet-based phase analysis techniques have been proposed. This study extends the scope of prior work by (1) exploring and contrasting the effectiveness of using wavelets on a wide range of program execution statistics for phase analysis; and (2) investigating techniques that can further optimize the accuracy of wavelet-based phase classification. More importantly, we identify additional benefits that wavelets can offer in the context of phase analysis. For example, wavelet transforms can provide efficient dimensionality reduction of large volume, high dimension raw program execution statistics from the time domain and hence can be integrated with a sampling mechanism to efficiently increase the scalability of phase analysis of large scale phase behavior on long-running workloads. To address workload variability issues in phase classification, wavelet-based denoising can be used to extract the essential features of workload behavior from their runtime non-deterministic (i.e., noisy) statistics. Our future research work includes the integration of the proposed wavelet approaches to runtime hardware and software monitoring systems. In this paper, selection of a wavelet basis is confined to within the Daubechies family of wavelets, which is widely used due to its simplicity of implementation. Other wavelets and their tradeoffs between efficiency and complexity need more investigation in the future.

References


