

Complexity-based Program Phase Analysis and Classification

Chang-Burm Cho

Intelligent Design of Efficient Architecture Lab (IDEAL)
Department of Electrical and Computer Engineering
University of Florida
choreno@ufl.edu

Tao Li

Intelligent Design of Efficient Architecture Lab (IDEAL)
Department of Electrical and Computer Engineering
University of Florida
taoli@ece.ufl.edu

ABSTRACT

Modeling and analysis of program behavior are at the foundation of computer system design and optimization. As computer systems become more adaptive, their efficiency increasingly depends on program dynamic characteristics. Previous studies have revealed that program runtime execution manifests phase behavior. Recently, methods and tools to analyze and classify program phases have also been developed. However, very few studies have been proposed so far to understand and evaluate program phases from their dynamics and complexity perspectives. In this work, we propose new methods, metrics and frameworks which aim to analyze, quantify, and classify the dynamics and complexity of program phases. Our methods use wavelet techniques to represent program phases at multiresolution scales. The cross-correlation coefficients between phase dynamics observed at different scales are then computed as metrics to quantify phase complexity. We propose to apply wavelet-based multiresolution analysis and data clustering to classify program execution into phases that exhibit similar degree of complexity. Experimental results on SPEC CPU 2000 benchmarks show that the proposed schemes classify complexity-based program phases better than currently used approaches.

Categories and Subject Descriptors

C [Computer Systems Organization]: General – *modeling of computer architecture*

General Terms

Measurement, Performance, Design

Keywords

Program Phase, Performance Modeling, Computer Architecture, Wavelet

1. INTRODUCTION

Modeling and analysis of program behavior are at the foundation of computer system design and optimization. By

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PACT'06, September 16-20, 2006, Seattle, Washington, USA.
Copyright 2006 ACM 1-59593-264-X/06/0009...\$5.00.

knowing program behavior, both hardware and software can be tuned to better suit the needs of applications. As computer systems become more adaptive, their efficiency increasingly depends on the dynamic behavior that programs exhibit at runtime. Previous studies [1-5] have shown that program runtime characteristics exhibit 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 complex computer systems can be efficiently solved by exploiting program phases [1, 6-9]. For this reason, there is a growing interest in studying program phase behavior. Recently, several phase analysis techniques have been proposed [4, 7, 10-19]. Very few of these studies, however, focus on understanding and characterizing program phases from their dynamics and complexity perspectives. Consequently, these techniques generally lack the capability of informing phase dynamic behavior. Obtaining phase dynamics, in many cases, is of great interest to accurately capture program behavior and to precisely apply runtime application oriented optimizations. For example, complex, real-world workloads may run for hours, days or even months before completion. Their long execution time implies that program time varying behavior can manifest across a wide range of scales, making modeling phase behavior using a single time scale less informative. Prior studies have shown that while phase-based process scheduling targets a program execution interval on the level of a context switch (approximately 10 million instructions), a phase-aware dynamic cache reconfiguration for energy saving can be performed at a granularity of 100K instructions [6, 7]. Moreover, many symptoms that reflect system runtime operation conditions, such as inductive power supply noise (i.e. the dI/dt problem) and thermal flow are inherently linked to program dynamics at different granularities. For example, thermal issue is related to workload execution with relevant time constants of millions of cycles or more, while the dI/dt problem occurs due to the workload variability on a much finer granularity, e.g. tens to hundreds of cycles [20].

To complement current phase analysis techniques which pay little or no attention to phase dynamics, we develop new methods, metrics and frameworks that have the capability to analyze, quantify, and classify program phases based on their dynamics and complexity characteristics. Our techniques are built on wavelet-based multiresolution analysis, which provides a clear and orthogonal view of phase dynamics by presenting complex dynamic structures of program phases with respect to both time and frequency domains. Consequently, key tendencies can be efficiently identified. Specifically, the goal of our work is to

answer the following questions: How to define the complexity of program dynamics? How do program dynamics change over time? If classified using existing methods, how similar are the program dynamics in each phase? How to better identify phases with homogeneous dynamic behavior?

The contributions of our work are: We proposed using wavelet-based multiresolution analysis to characterize phase dynamic behavior. We developed metrics to quantitatively evaluate the complexity of phase structures. We proposed methodologies to classify program phases from their dynamics and complexity perspectives. We implemented our complexity-based phase analysis technique and evaluate its effectiveness over existing phase analysis methods based on program control flow and runtime information. We showed that in both cases the proposed technique produces phases that exhibit more homogeneous dynamic behavior than existing methods do.

The rest of this paper is organized as follows. Section 2 introduces wavelet transform and multiresolution analysis. Section 3 describes experimental setup including simulated machine configuration, studied workloads and evaluated metrics. Section 4 provides a detailed profile of program dynamics. Section 5 proposes multiresolution phase classification using wavelet transform techniques. Section 6 evaluates the efficiency of the proposed techniques in classifying phase dynamics. Section 7 discusses related work. Section 8 summarizes the paper and outlines our future work.

2. BACKGROUND: WAVELETS AND MULTIREOLUTION ANALYSIS

In this study, we use wavelets to effectively characterize and analyze the complexity and dynamics in program phases. To familiarize the reader with general methods used in this paper, we provide a brief overview on wavelet-based multiresolution analysis in this section. To learn more mathematical details of wavelets, a reader is encouraged to read [21].

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 anticipated to provide a compact and effective mathematical representation of data. In contrast to Fourier transforms which only offer frequency representations, wavelets are capable of providing time and frequency localizations simultaneously [22].

Wavelet analysis employs two functions, often referred to as the scaling function and the wavelet function, to generate a family of functions that break down the original data. The scaling function is similar in concept to an approximation function, while the wavelet function quantifies the differences between the original data and the approximation generated by the scaling function. Wavelet analysis allows one to choose the pair of scaling and wavelet functions from numerous functions. In this section, we provide a quick primer on wavelet analysis using the Harr wavelet, which is the simplest form of wavelets [21].

The Harr discrete wavelet decomposition (DWT) works by averaging two adjacent values on a series of data at a given scale

to form smoothed, lower-dimensional data (i.e. approximations), and the resulting coefficients (i.e. details), which are the differences between the values and their averages. Table 1 illustrates the procedure of using Harr-base DWT to transform a series of data {3, 4, 20, 25, 15, 5, 20, 3}. As can be seen, scale 1 is the finest representation of the data. At scale 2, the approximations {3.5, 22.5, 10, 11.5} are obtained by taking the average of {3, 4}, {20, 25}, {15, 5} and {20, 3} at scale 1 respectively. The details {-0.5, -2.5, 5, 8.5} are the differences of {3, 4}, {20, 25}, {15, 5} and {20, 3} divided by 2 respectively.

Table 1. An example of Harr wavelet transform on data {3, 4, 20, 25, 15, 5, 20, 3}

Scale	Scaling Coefficients (Approximation)	Wavelet Coefficients (Details)
1 (finest)	{3, 4, 20, 25, 15, 5, 20, 3}	
2	{3.5, 22.5, 10, 11.5}	{-0.5, -2.5, 5, 8.5}
4	{13, 10.75}	{-9.5, -0.75}
8 (coarsest)	{11.875}	{1.125}

The process continues by decomposing the scaling coefficient (approximation) vector using the same steps, and completes when only one coefficient remains. As a result, wavelet decomposition is the collection of average and details coefficients at all scales. In other words, the wavelet transform of the original data is the single coefficient representing the overall average of the original data, followed by the detail coefficients in order of increasing resolutions. Different resolutions can be obtained by adding difference values back or subtracting differences from the averages. For instance, {13, 10.75} = {11.875+1.125, 11.875-1.125} where 11.875 and 1.125 are the first and the second coefficient respectively. This process can be performed recursively until the finest scale is reached.

The DWT provides a natural hierarchy structure for multiresolution data representation. The scaling coefficients contain an overall, coarse approximation of the data while the wavelet coefficients illustrate high detail. As described above, the original data can be precisely reconstructed if both scaling coefficients and wavelet coefficients are used. Alternatively, approximated versions of the original data can be generated by applying the wavelet reconstruct function only to scaling coefficients generated by DWT operations at different scales. This property is very useful in characterizing and quantifying phase dynamics and complexity. Namely, the approximations generated by wavelet analysis at different scales can be used to study the changing of phase dynamics and complexity over time. Figure 1 shows an example of using the Harr wavelet scaling coefficients to approximate the dynamic behavior of an execution interval of a SPEC CPU 2000 benchmark *gcc* at different scales. IPC (instructions per cycle) is the monitored program characteristic. Figure 1 (a) displays the program dynamics at the finest scale 1. Figures 1 (b) to 1 (g) show approximations based on the Harr scaling function, with increasing faithful wavelet approximations. As can be seen, the wavelet analysis acts as a mathematical “microscope” which allows one to zoom in on fine structures of a program execution interval or, alternatively, to reveal large scale structures by zooming out. As a result, the DWT can provide capability of representing program phase dynamics and complexity at different levels of resolution.

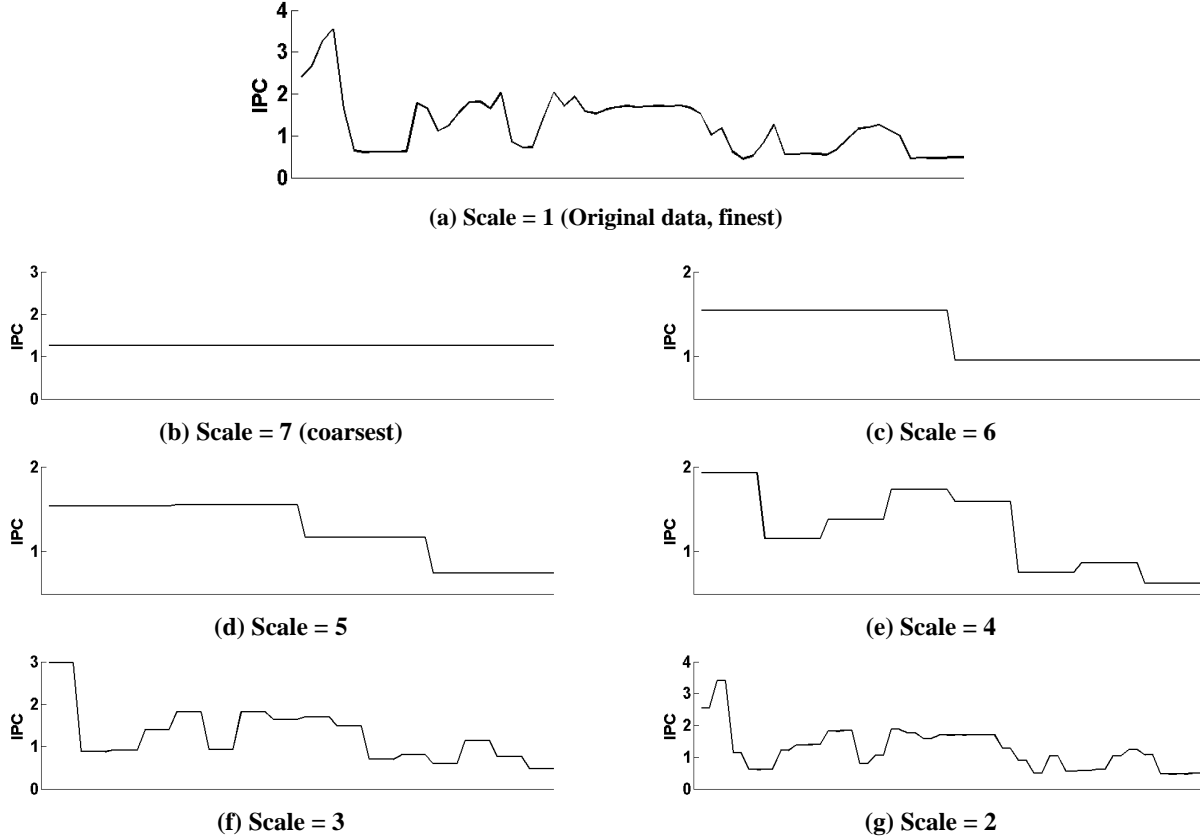


Figure 1. The wavelet can represent program behavior at different levels of resolution or scale. Above we see the raw IPC data which represents an execution interval of benchmark *gcc* (Fig. 1.a), with increasing faithful wavelet approximations (Fig. 1.b-Fig. 1.g) using the Harr scaling coefficients.

3. EXPERIMENTAL SETUP

Using the above described wavelet-based multiresolution analysis, we characterize, quantify and classify program dynamic behavior on a high-performance, out-of-order execution superscalar processor coupled with a multi-level memory hierarchy. This section describes our experimental methodologies, the simulated machine configuration, experimented benchmarks and evaluated metrics.

3.1 Machine Configuration

We performed our analysis using ten SPEC CPU 2000 benchmarks *crafty*, *gap*, *gcc*, *gzip*, *mcf*, *parser*, *perlbmk*, *swim*, *twolf* and *vortex*. All programs were run with reference input to completion. We chose to focus on only 10 programs because of the lengthy simulation time incurred by executing all of the programs to completion. The statistics of workload dynamics were measured on the SimpleScalar 3.0 [23] *sim-outorder* simulator for the Alpha ISA. The baseline microarchitecture model is detailed in Table 2.

Table 2. Baseline machine configuration

Parameter	Configuration
Processor Width	8
ITLB	128 entries, 4-way, 200 cycle miss
Branch Prediction	combined 8K tables, 10 cycle misprediction, 2 predictions/cycle
BTB	2K entries, 4-way
Return Address Stack	32 entries
L1 Instruction Cache	32K, 2-way, 32 Byte/line, 2 ports, 4 MSHR, 1 cycle access
RUU Size	128 entries
Load/ Store Queue	64 entries
Store Buffer	16 entries
Integer ALU	4 I-ALU, 2 I-MUL/DIV
FP ALU	2 FP-ALU, 1FP-MUL/DIV
DTLB	256 entries, 4-way, 200 cycle miss
L1 Data Cache	64KB, 4-way, 64 Byte/line, 2 ports, 8 MSHR, 1 cycle access
L2 Cache	unified 1MB, 4-way, 128 Byte/line, 12 cycle access
Memory Access	100 cycles

3.2 Metrics to Quantify Phase Complexity

To quantify phase complexity, we measure the similarity between phase dynamics observed at different time scales. To be more specific, we use cross-correlation coefficients to measure the similarity between the original data sampled at the finest granularity and the approximated version reconstructed from wavelet scaling coefficients obtained at a coarser scale. The cross-correlation coefficients (XCOR) of the two data series are defined as:

$$XCOR(X, Y) = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2 \cdot \sum_{i=1}^n (Y_i - \bar{Y})^2}} \quad (\text{Eq. 1}),$$

where X is the original data series and Y is the approximated data series. Note that XCOR = 1 if program dynamics observed at the finest scale and its approximation at coarser granularity exhibit perfect correlation, and XCOR = 0 if the program dynamics and its approximation varies independently across time scales.

X and Y can be any runtime program characteristics of interest. In this study, we use instruction per cycle (IPC) as a metric due to its wide usage in computer architecture design and performance evaluation. To sample IPC dynamics, we break down the entire program execution into 1024 intervals and then sample 1024 IPC data within each interval. Therefore, at the finest resolution level, the program dynamics of each execution interval are represented by an IPC data series with a length of 1024. We then apply wavelet multiresolution analysis to each interval. As shown in Table 1, each DWT operation produces an approximation coefficients vector with a length equal to half of the input data. We remove the detail coefficients after each wavelet transform and only use the approximation part to reconstruct IPC dynamics and then calculate the XCOR between the original data and the reconstructed data. We apply discrete wavelet transform to the approximation part iteratively until the length of the approximation coefficient vector is reduced to 1. Each approximation coefficient vector is used to reconstruct a full IPC trace with a length of 1024 and the XCOR between the original and reconstructed traces are calculated using Eq. 1. As a result, for each program execution interval, we obtain an XCOR vector, in which each element represents the cross-correlation coefficients between the original workload dynamics and the approximated workload dynamics at different scales. Since we use 1024 samples within each interval, we create an XCOR vector with a length of 10 for each interval, as shown in Figure 2.

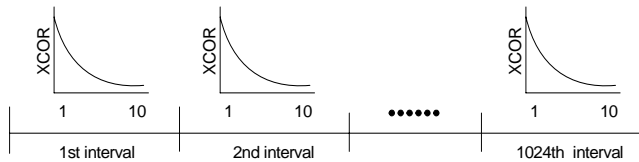


Figure 2. A XCOR vector for each program execution interval is generated by calculating cross-correlation coefficient between the original program dynamics and the approximated program dynamics using wavelet multiresolution analysis.

Because coiflet wavelets perform better representation of data approximation at a given resolution than Harr and Daubechies

wavelets [21], we use coiflet wavelets with an order of 5 for the all of the experiments we perform in the rest of this paper.

4. PROFILING PROGRAM DYNAMICS AND COMPLEXITY

We use the above described metrics to quantify program dynamics and complexity of the studied SPEC CPU 2000 benchmarks. Figure 3 shows the results of the total 1024 execution intervals across ten levels of abstraction for the benchmark *gcc*. As can be seen, the benchmark *gcc* shows a wide variety of changing dynamics during its execution. As the time scale increases, XCOR values are monotonically decreased. This is due to the fact that wavelet approximation at a coarse scale removes details in program dynamics observed at a fine grained level. Rapidly decreased XCOR implies highly complex structures that can not be captured by coarse level approximation. In contrast, slowly decreased XCOR suggests that program dynamics can be largely preserved using few samples. Figure 3 also shows a dotted line along which XCOR decreases linearly with the increased time scales. The XCOR plots below that dotted line indicate rapidly decreased XCOR values or complex program dynamics. As can be seen, a significant fraction of the benchmark *gcc* execution intervals manifest quickly decreased XCOR values, indicating that the program exhibits highly complex structure at the fine grained level. Figure 3 also reveals that there are a few *gcc* execution intervals that have good scalability in their dynamics. On these execution intervals, the XCOR values only drop 0.1 when the time scale is increased from 1 to 8. The results shown in Figure 3 clearly indicate that some program execution intervals can be accurately approximated by their high level abstractions while others can not.

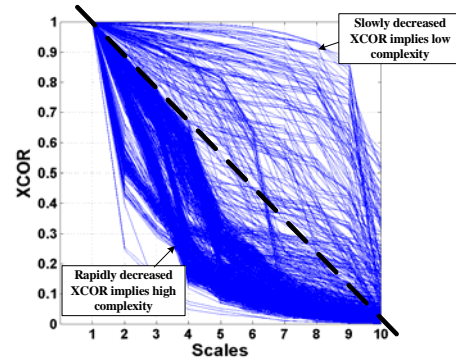


Figure 3. Dynamic complexity profile of benchmark *gcc* (A total 1024 execution intervals which represent the entire program run are shown)

We further break down the XCOR values into 10 categories ranging from 0 to 1 and analyze their distribution across time scales. Due to space limitations, we only show the results of three programs (*swim*, *crafty* and *gcc*, see Figure 4) which represent the characteristics of all analyzed benchmarks. Note that at scale 1, the XCOR values of all execution intervals are always 1. Programs show heterogeneous XCOR value distributions starting from scale level 2. As can be seen, the benchmark *swim* exhibits good scalability in its dynamic complexity. The XCOR values of all execution intervals remain above 0.9 when the time scale is

increased from 1 to 7. This implies that the captured program behavior is not sensitive to any time scale in that range. Therefore, we classify *swim* as a low complexity program. On the benchmark *crafty*, XCOR values decrease uniformly with the increase of time scales, indicating the observed program dynamics are sensitive to the time scales used to obtain it. We refer to this behavior as medium complexity. On the benchmark *gcc*, program dynamics decay rapidly. This suggests that abundant program dynamics could be lost if coarser time scales are used to characterize it. We refer to this characteristic as high complexity behavior.

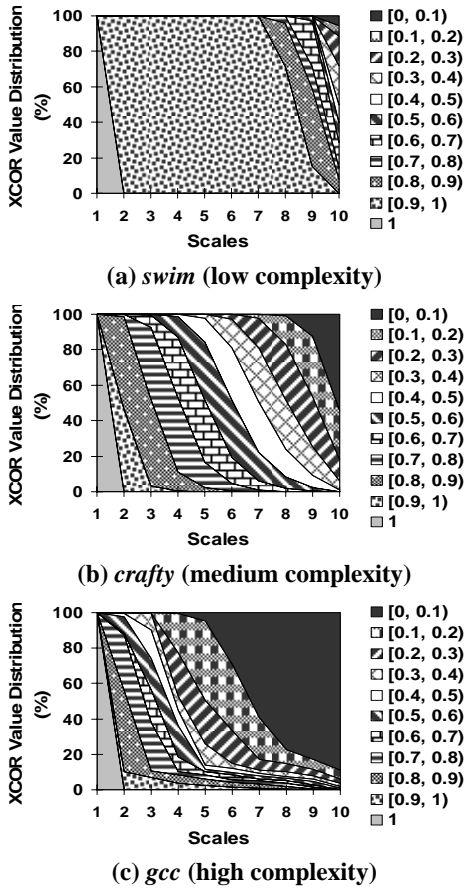


Figure 4. Cross-correlation coefficients (XCOR) value distribution plots

The dynamics complexity and the XCOR value distribution plots shown in Figure 3 and Figure 4 provide a quantitative and informative representation of runtime program complexity. Using the above information, we classify the studied programs in terms of their complexity and the results are shown in Table 3.

Table 3. A classification of benchmarks based on their complexity

Category	Benchmarks
Low complexity	<i>swim</i>
Medium complexity	<i>crafty, gzip, parser, perlbnk, twolf</i>
High complexity	<i>gap, gcc, mcf, vortex</i>

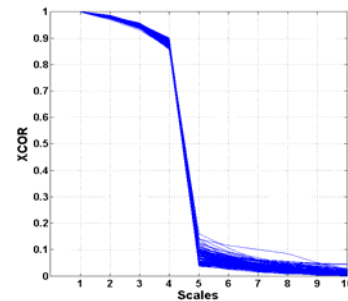
5. CLASSIFYING PROGRAM PHASES BASED ON THEIR DYNAMICS BEHAVIOR

In Section 4, we show that program execution manifests heterogeneous complexity behavior. In this section, we further examine the efficiency of using current methods in classifying program dynamics into phases and propose a new method that can better identify program complexity. Classifying complexity based phase behavior enables us to understand program dynamics progressively in a fine-to-coarse fashion, to operate on different resolutions, to manipulate features at different scales, and to localize characteristics in both spatial and frequency domains.

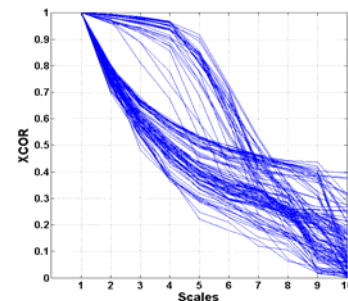
5.1 Simpoint

In [1], Sherwood and Calder proposed a phase analysis tool called Simpoint to automatically classify the execution of a program into phases. They found that intervals of program execution grouped into the same phase had similar statistics. The Simpoint tool clusters program execution based on code signature and execution frequency. We identified program execution intervals grouped into the same phase by the Simpoint tool and analyzed their dynamic complexity.

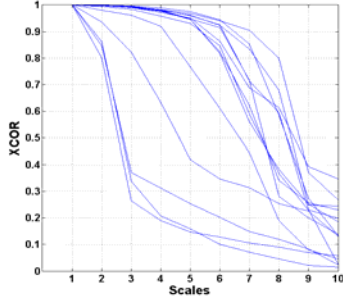
Figure 5 shows the results for the benchmark *mcf*. Simpoint generates 55 clusters on the benchmark *mcf*. Figure 5 shows program dynamics within three clusters generated by Simpoint. Each cluster represents a unique phase. In cluster 7, the classified phase shows homogeneous dynamics. In cluster 5, program execution intervals show two distinct dynamics but they are classified as the same phase. In cluster 48, program execution complexity varies widely; however, Simpoint classifies them as a single phase. The results shown in Figure 5 suggest that program execution intervals classified as the same phase by Simpoint can still exhibit widely varied behavior in their dynamics.



(a) Simpoint Cluster #7



(b) Simpoint Cluster #5



(c) Simpoint Cluster #48

Figure 5. Program (*mc*) execution intervals identified as the same phase by the Simpoint can still exhibit widely varied dynamic behavior.

5.2 Complexity-aware Phase Classification

To enhance the capability of current methods in characterizing program dynamics, we propose complexity-aware phase classification. Our method uses the multiresolution property of wavelet transforms to identify and classify the changing of program code execution across different scales.

We assume a baseline phase analysis technique that uses basic block vectors (BBV) [10]. A basic block is a section of code that is executed from start to finish with one entry and one exit. A BBV represents the code blocks executed during a given interval of execution. To represent program dynamics at different time scales, we create a set of basic block vectors for each interval at different resolutions. For example, at the coarsest level (scale = 10), a program execution interval is represented by one BBV. At the most detailed level, the same program execution interval is represented by 1024 BBVs from 1024 consecutively subdivided intervals, as shown in Figure 6. To reduce the amount of data that needs to be processed, we use random projection to reduce the dimensionality of all BBVs to 15, as suggested in [1].



Figure 6. Basic Block Vectors (BBVs) that represent the same program execution interval at different resolutions (The coarser scale BBVs are the approximations of the finest scale BBVs generated by the wavelet-based multiresolution analysis)

The coarser scale BBVs are the approximations of the finest scale BBVs generated by the wavelet-based multiresolution analysis. As shown in Figure 7, the discrete wavelet transform is applied to each dimension of a set of BBVs at the finest scale. The XCOR calculation is used to estimate the correlations between a BBV element and its approximations at coarser scales. The results are the 15 XCOR vectors representing the complexity

of each dimension in BBVs across 10 level abstractions. The 15 XCOR vectors are then averaged together to obtain an aggregated XCOR vector that represents the entire BBV complexity characteristics for that execution interval.

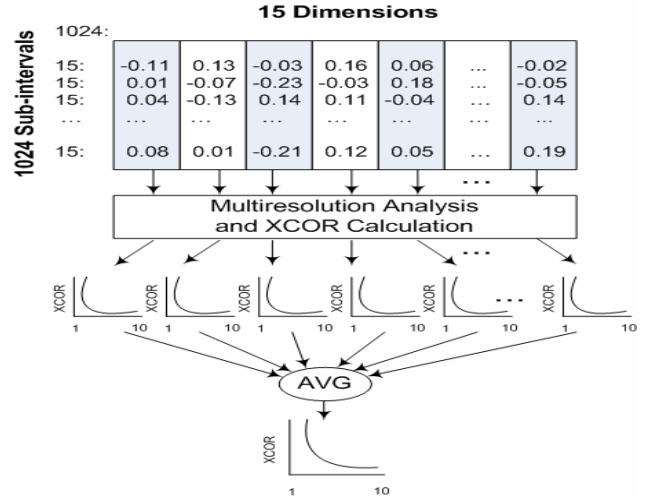


Figure 7. Multiresolution analysis of the projected basic block vectors (BBVs) that represent one program execution interval. The XCOR calculation is used to estimate the correlations between a BBV element and its approximations at coarser scales.

Using the above steps, we obtained an aggregated XCOR vector for each program execution interval. We then run the k-means clustering algorithm [24] on the collected XCOR vectors which represent the dynamic complexity of program execution intervals and classified them into phases. This is similar to what Simpoint does. The difference is that the Simpoint tool uses raw BBVs and our method uses aggregated BBV XCOR vectors as the input for k-means clustering.

6. EXPERIMENTAL RESULTS

We compare Simpoint and the proposed approach in their capability of classifying phase complexity. Since we use wavelet transform on program basic block vectors, we refer to our method as multiresolution analysis of BBV (MRA-BBV). We examine the similarity of program complexity within each classified phase by the two approaches. Instead of using IPC, we use IPC dynamics as the metric for evaluation. After classifying all program execution intervals into phases, we examine each phase and compute the IPC XCOR vectors for all the intervals in that phase. We then calculate the standard deviation in IPC XCOR vectors within each phase, and we divide the standard deviation by the average to get the Coefficient of Variation (COV). As shown in Figure 8, we calculate an overall COV metric for a phase classification method by taking the COV of each phase, 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 classification for a given program. Since COV measures standard deviations as the percentage of the average, a lower COV value means better phase classification technique.

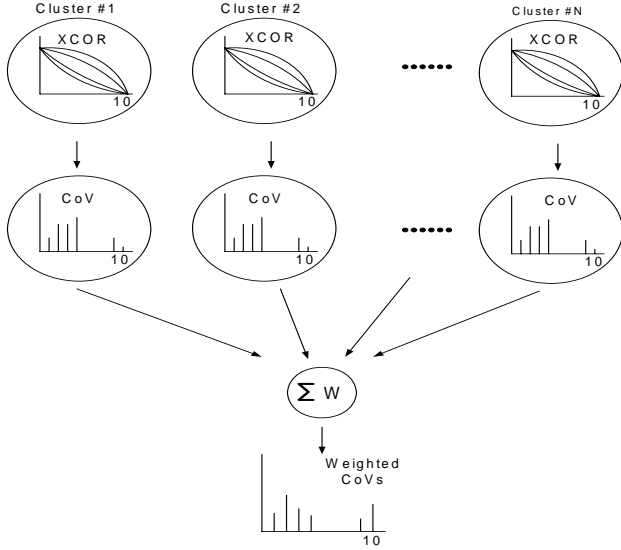


Figure 8. Methods to compute weighted COVs

Figure 9 shows experimental results for all the studied benchmarks. As can be seen, the MRA-BBV method can produce phases which exhibit more homogeneous dynamics and complexity than the standard, BBV-based method. This can be seen from the lower COV values generated by the MRA-BBV method. In general, the COV values yielded on both methods increase when coarse time scales are used for complexity approximation. The MRA-BBV is capable of achieving significantly better classification on benchmarks with high complexity, such as *gap*, *gcc* and *mcf*. On programs which exhibit medium complexity, such as *crafty*, *gzip*, *parser* and *twolf*, the two schemes show a comparable effectiveness. On benchmark (e.g. *swim*) which has trivial complexity, both schemes work well.

We further examine the capability of using runtime performance metrics to capture complexity-aware phase behavior. Instead of using BBV, the sampled IPC is used directly as the input to the k-means phase clustering algorithm. Similarly, we apply multiresolution analysis to the IPC data and then use the gathered information for phase classification. We call this method multiresolution analysis of IPC (MRA-IPC). Figure 10 shows the phase classification results. As can be seen, the observations we made on the BBV-based cases hold valid on the IPC-based cases. This implies that the proposed multiresolution analysis can be applied to both methods to improve the capability of capturing phase dynamics.

7. RELATED WORK

Prior research has considered a range of possible phase analysis techniques. Sherwood and Calder proposed the use of Basic Block Vectors as a metric to capture a program’s phase behavior [1, 10]. 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 at a single time scale. Hind et al [25] studied the impact of granularity and similarity on phase analysis. The work of Lau and Calder [26] showed there exists a hierarchy of phase behavior and proposed the use of variable length intervals for phase classification. The focuses of our work are 1) to quantitatively analyze the changing of phase dynamics across different scales and 2) to develop a method and a framework that can classify phases which exhibit homogeneity in their scaling behavior.

Wavelets have been widely used in numerous fields of study including computer science and engineering [27-29]. Hardware and algorithms that perform fast and efficient wavelet transforms have been proposed in the past [30, 31]. In [32], Joseph and Martonosi used wavelets to analyze and predict the change of processor voltage over time. In an earlier work [20], they used Fourier analysis to characterize the power behavior of programs. In this work, we demonstrate that wavelets could play an important role in the modeling and analysis of program phase behavior. In [4], Shen and Ding used wavelets as a filter to remove the gradual changes in reuse-distance trace to identify locality phase in programs. In their work, only single level wavelet transform was performed. In this work, we apply wavelet-based multiresolution analysis to a wide range of scales.

8. CONCLUSIONS AND FUTURE WORK

Studying program phase behavior is of growing interest in computer architecture research. Recently, several phase analysis techniques have been proposed. The performance, power and dependability optimizations of future computer workloads and systems could involve analyzing program dynamics across many time scales. Modeling program behavior at single scale can yield many limitations. For example, samples taken from a single, fine-grained interval may not be useful in forecasting how a program behaves at a medium or large time scales. In contrast, observing program behavior using a coarse-grained time scale may lose opportunities that can be exploited by hardware and software in tuning resources to optimize workload execution at a fine-grained level.

In this paper, we proposed new methods, metrics and framework that can help researchers and designers to better understand phase complexity and the changing of program dynamics across multiple time scales. We proposed using wavelet transformations of code execution and runtime characteristics to produce a concise yet informative view of program dynamic complexity. We demonstrated the use of this information in phase classification which aims to produce phases that exhibit similar degree of complexity. Characterizing phase dynamics across different scales provides insightful knowledge and abundant features that can be exploited by hardware and software in tuning resources to meet the requirement of workload execution at different granularities. In our future work, we plan to study architecture with multiple granularity configurations capable of taking advantage of our phase dynamics classification work.

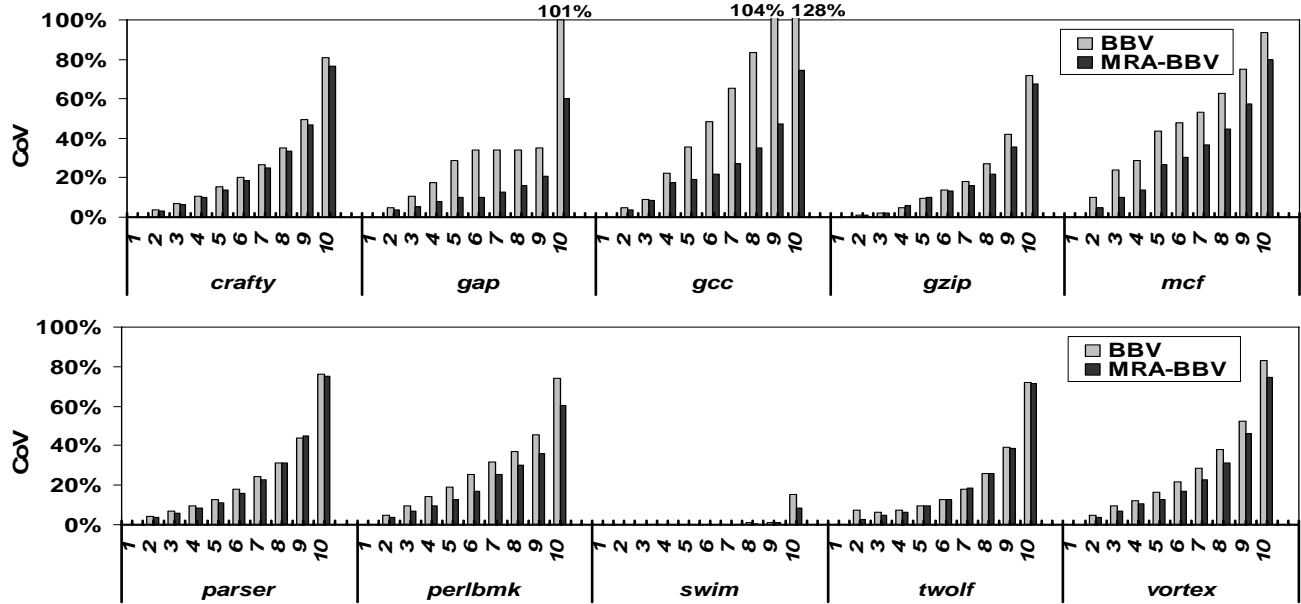


Figure 9. A comparison of BBV and MRA-BBV schemes in classifying phase dynamics (The x-axis represents different time scales)

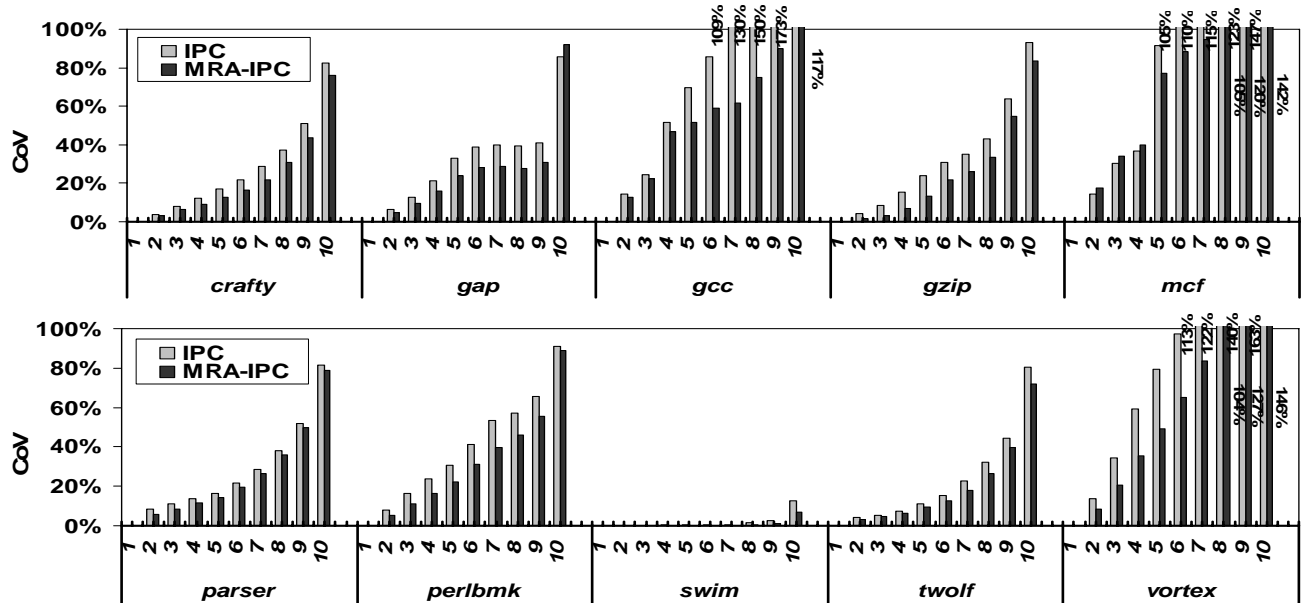


Figure 10. A comparison of IPC and MRA-IPC schemes in classifying phase dynamics (The x-axis represents different time scales)

9. ACKNOWLEDGMENT

This research is partially supported by a Microsoft Research Trustworthy Computing Award and an IBM Faculty Award.

10. REFERENCES

- [1] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, Automatically Characterizing Large Scale Program Behavior, In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.
- [2] E. Duesterwald, C. Cascaval and S. Dwarkadas, Characterizing and Predicting Program Behavior and Its Variability, In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2003.
- [3] J. Cook, R. L. Oliver, and E. E. Johnson, Examining Performance Differences in Workload Execution Phases, In Proceedings of the IEEE International Workshop on Workload Characterization, 2001.
- [4] X. Shen, Y. Zhong and C. Ding, Locality Phase Prediction, In Proceedings of the International Conference on

- Architectural Support for Programming Languages and Operating Systems, 2004.
- [5] C. Isci and M. Martonosi, Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data, In Proceedings of the International Symposium on Microarchitecture, 2003.
- [6] T. Sherwood, S. Sair and B. Calder, Phase Tracking and Prediction, In Proceedings of the International Symposium on Computer Architecture, 2003.
- [7] A. Dhodapkar and J. Smith, Managing Multi-Configurable Hardware via Dynamic Working Set Analysis, In Proceedings of the International Symposium on Computer Architecture, 2002.
- [8] M. Huang, J. Renau and J. Torrellas, Positional Adaptation of Processors: Application to Energy Reduction, In Proceedings of the International Symposium on Computer Architecture, 2003.
- [9] W. Liu and M. Huang, EXPERT: Expedited Simulation Exploiting Program Behavior Repetition, In Proceedings of International Conference on Supercomputing, 2004.
- [10] T. Sherwood, E. Perelman and B. Calder, Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications, In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2001.
- [11] A. Dhodapkar and J. Smith, Comparing Program Phase Detection Techniques, In Proceedings of the International Symposium on Microarchitecture, 2003.
- [12] C. Isci and M. Martonosi, Identifying Program Power Phase Behavior using Power Vectors, In Proceedings of the International Workshop on Workload Characterization, 2003.
- [13] C. Isci and M. Martonosi, Phase Characterization for Power: Evaluating Control-Flow-Based Event-Counter-Based Techniques, In Proceedings of the International Symposium on High-Performance Computer Architecture, 2006.
- [14] M. Annavaram, R. Rakvic, M. Polito, J.-Y. Bouguet, R. Hankins and B. Davies, The Fuzzy Correlation between Code and Performance Predictability, In Proceedings of the International Symposium on Microarchitecture, 2004.
- [15] J. Lau, S. Schoenmackers and B. Calder, Structures for Phase Classification, In Proceedings of International Symposium on Performance Analysis of Systems and Software, 2004.
- [16] J. Lau, J. Sampson, E. Perelman, G. Hamerly and B. Calder, The Strong Correlation between Code Signatures and Performance, In Proceedings of the International Symposium on Performance Analysis of Systems and Software, 2005.
- [17] J. Lau, S. Schoenmackers and B. Calder, Transition Phase Classification and Prediction, In Proceedings of the International Symposium on High Performance Computer Architecture, 2005.
- [18] C. Isci and M. Martonosi, Detecting Recurrent Phase Behavior under Real-System Variability, In Proceedings of the International Symposium on Workload Characterization, 2005.
- [19] E. Perelman, M. Polito, J. Y. Bouguet, J. Sampson, B. Calder, C. Dulong, Detecting Phases in Parallel Applications on Shared Memory Architectures, In Proceedings of the International Parallel and Distributed Processing Symposium, Apr. 2006.
- [20] R. Joseph, M. Martonosi and Z. G. Hu, Spectral Analysis for Characterizing Program Power and Performance, In Proceedings of the International Symposium on Performance Analysis of Software and Systems, 2004.
- [21] I. Daubechies. Ten Lectures on Wavelets. Capital City Press, Montpelier, Vermont, 1992.
- [22] I. Daubechies, Orthonormal bases of Compactly Supported Wavelets, Communications on Pure and Applied Mathematics, vol. 41, pp. 906-966, 1988.
- [23] SimpleScalar, <http://www.simplescalar.com/>
- [24] J. MacQueen, Some Methods for Classification and Analysis of Multivariate Observations, In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967.
- [25] M. J. Hind, V. T. Rajan, and P. F. Sweeney, Phase Shift Detection: A Problem Classification, IBM Research Report RC-22887, IBM T. J. Watson, Aug. 2003.
- [26] J. Lau, E. Perelman, G. Hamerly, T. Sherwood and B. Calder, Motivation for Variable Length Intervals and Hierarchical Phase Behavior, In Proceedings of the International Symposium on Performance Analysis of Systems and Software, 2005.
- [27] S. Mallat, Multifrequency Channel Decompositions of Images and Wavelet Models, IEEE Transactions on Acoustic, Speech, and Signal Processing, vol. 37, pp. 2091-2110, 1989.
- [28] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz, The Changing Nature of Network Traffic: Scaling Phenomena, ACM Computer Communication Review, vol. 28, pp. 5-29, Apr. 1998.
- [29] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, On the Self-Similar Nature of Ethernet Traffic, IEEE/ACM Transactions on Networking, vol. 2, no. 1, pp. 1-15, Feb. 1994.
- [30] C. Chakrabarti, M. Viswanath, R. M. Owens, Architectures for Wavelets Transforms: A Survey, Journal of VLSI Signal Processing, vol. 41, pp. 171-192, Dec. 1996.
- [31] P. Y. Chen, VLSI Implementation for One-Dimensional Multilevel Lifting-Based Wavelet Transform, IEEE Transactions on Computers, vol. 53, no. 4, Apr. 2004.
- [32] R. Joseph, Z. G. Hu, and M. Martonosi, Wavelet Analysis for Microprocessor Design: Experiences with Wavelet-Based dI/dt Characterization, In Proceedings of the International Symposium on High Performance Computer Architecture, 2004.