Accurate Performance Estimation with BBFDA: Beyond Granularity Constraints

Hsuan-Yi Lin, Ren-Song Tsay
National Tsing Hua University, Taiwan
mark126688@hotmail.com, rstsay@cs.nthu.edu.tw

Abstract—**In this paper, we present BBFDA, a pioneering approach for precise performance estimation in computer systems. Conventional time-quantum-based methods often encounter granularity limitations, impeding their ability to capture program behavior accurately. BBFDA utilizes Basic Block Analysis and Recursive Frequency Domain Analysis to estimate performance waveforms. This method enables dynamic performance tracking without being constrained by granularity issues and remains robust in the face of input variations. We assess the performance of BBFDA using SPEC CPU2017 benchmarks, showcasing its exceptional accuracy and resilience, particularly in multi-phase scenarios.**

I. Introduction

As system optimization gains importance, understanding an application's runtime behaviors becomes crucial, especially in dedicated embedded systems. Accurate modeling of application behavior is essential for optimizing performance. Applications often exhibit repetitive execution phases with consistent performance values, as shown in Fig. 1. Numerous studies have developed phase prediction techniques [1-3], but a clear phase definition remains elusive.

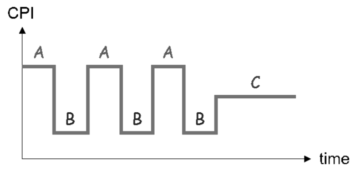

Fig. 1: An illustrative example of a program with three distinct execution phases, each exhibiting a unique CPI value.

Existing approaches segment execution traces into fixed-length segments, but this may lead to phase misalignment and disregarding shorter phases. We aim to identify fundamental program phase building blocks and develop a systematic, unambiguous program phase identification approach.

Our research leverages frequency domain analysis to detect repetitive behaviors in applications. We focus on basic blocks, loops, and functions as stable code structures for precise phase identification. Basic blocks serve as the most fundamental execution units, addressing granularity issues common in time-quantum methods.

Our approach offers precise phase identification, including phase boundaries and performance characteristics. It enhances prediction accuracy by linking program phases to loop/function code structures, offering insights for optimization, and enabling runtime embedding of analysis results into the application code.

The paper is structured as follows: Sec. II provides a review and discussion of related work. Section III outlines our precise program phase identification method using frequency-domain analysis, and Section IV presents experimental results and a brief conclusion.

II. RELATED WORK

In the realm of phase identification, two common approaches are typically employed. The first approach is known as time-quantum-based, which entails dividing a program's execution into fixed-length intervals, each representing a phase segment that is subsequently merged into a phase. The second approach is program-structure-based, which relies on fundamental program structures like loops or functions to delineate program phases. However, both approaches exhibit certain limitations, which we will discuss in greater detail.

A. Time-quantum Approaches

A time quantum is a fixed-length contiguous interval of program execution used to divide an execution trace into non-overlapping quanta for phase analysis. Typically, a fixed number of instructions or basic blocks form an interval for evaluation, and quanta with similar performance metrics, such as CPI, cache miss rate, and branch miss rate, are grouped into one phase. However, the term "similar" is not precisely defined, allowing for a certain margin of error.

One of the most representative time-quantum-based techniques is the Basic Block Vector (BBV) method. BBV is motivated by the observation that program phase behavior highly correlates with patterns of basic blocks [4-6]. Since program behavior primarily results from executing program code composed of basic blocks, BBV records the footprint of basic blocks in the execution trace. It checks if the compositions (or named signatures) of consecutive basic block vectors (a fixed number of basic blocks) display a difference greater than a given threshold value to identify phase boundaries.

However, a common problem with this approach is that quanta often span program phase boundaries, resulting in the so-called transition phase issue [7]. When the quantum spans an actual phase boundary, it's considered a unique phase with different performance measures compared to the phases before and after the boundary point. Fang et al. [8] observed that execution phases should have a hierarchical structure and proposed a Multi-Level Phase Analysis method that classifies phases into fine-grain (inner-loop) and coarse-grain (outer-loop or function) phases. Coarse-grain phases consist of stably distributed fine-grain phases. While this multi-level approach aligns better with practical cases, it still may not fully capture general scenarios. In contrast, our approach provides a more systematic method to identify the full phase hierarchy.

A significant challenge of fixed time-quantum-based approaches is that the fixed quantum length may not align with the irregular rhythm of program behavior. These approaches may not accurately capture actual program phases and fail to account for the dynamic nature of real program phases. Therefore, in this paper, we propose an effective approach that identifies the natural rhythm of program behavior.

B. Program-structure Approaches

Another approach to identifying program execution phases focuses on loops and functions, common program structures with consistent performance behavior, making them suitable for defining program phases. This method assumes that loops and functions are the basic components of a phase, and the goal is to identify the transitions that connect phases. For instance, Huang et al. [1] applied this idea to configure combinations of function calls (or subroutines) for performance or energy consumption improvements.

Lau et al. [9] employed a Hierarchical Call-Loop Graph analysis technique to identify program phases based on loops or functions (or named procedures). In this technique, each node represents a loop or a function, and each edge represents an execution path from one node to another, associated with a call count and average and standard deviation of instruction count across various invocations. Edges with lower deviation numbers represent code segments of relatively consistent performance behaviors, marked as phases. In contrast, Jiang et al. [10] did not explicitly identify program phases but aimed to find correlations among repeatedly executed loops and functions under various inputs. For instance, the analysis may reveal that a certain function's call count consistently maintains a fixed ratio to a loop's trip count under different tested inputs, which is then applied to predict runtime program behavior.

A common challenge with these program-structure-based approaches is the difficulty in calculating precise performance values (e.g., CPI) due to variations in different execution runs. Existing approaches rely on pre-conceived phase patterns, such as fixed phase length, fixed basic block vector size, or connections to loops or functions, to determine program phases, which may not necessarily match the natural program phase patterns. In contrast, our approach is based on identifying program phases using the inherent characteristics of basic blocks with hints from program code structure.

Next, we will present our proposed frequency-domain-based approach, offering a more systematic method of identifying program phases.

II. Frequency-Domain Phase Analysis Method

Recognizing program phases in waveforms is intuitive, but automating this process challenges us. To address this, we introduce a novel frequency-domain analysis method for phase detection. Our practical approach precisely identifies phase start, length, and average performance. It accommodates program loop and function structures for accuracy. While we use CPI as a performance metric, our method is versatile and not limited to this metric.

In practice, our approach transforms a program execution trace into a time-domain waveform with instruction counts as the time index and performance values for the waveform amplitudes. This waveform reveals the dominant program phase. By converting it into the frequency domain using the Fourier Transform, the major phase emerges as the main spectrum in the low-frequency region, representing repeated program patterns.

The proposed phase identification process involves analyzing the frequency domain spectrum (Fig. 2). We locate the main spectrum, i.e., the spectrum with the highest dynamic performance value, corresponding to the primary program phase, and calculate its length using the spectrum's occurrence value. For instance, if the main spectrum occurs four times with a total execution length of 4,500 instructions, the main phase's length is precisely 1,125 instructions (L=D/X, where L is the phase length, D is the total execution length, and X is the occurrence frequency). We then combine this length with code structure components like basic blocks, loops, and functions to pinpoint the phase's starting point. Our approach adopts a recursive method to explore hierarchical phase structures. Special cases include identifying high and low-phase boundaries and handling flat waveforms.

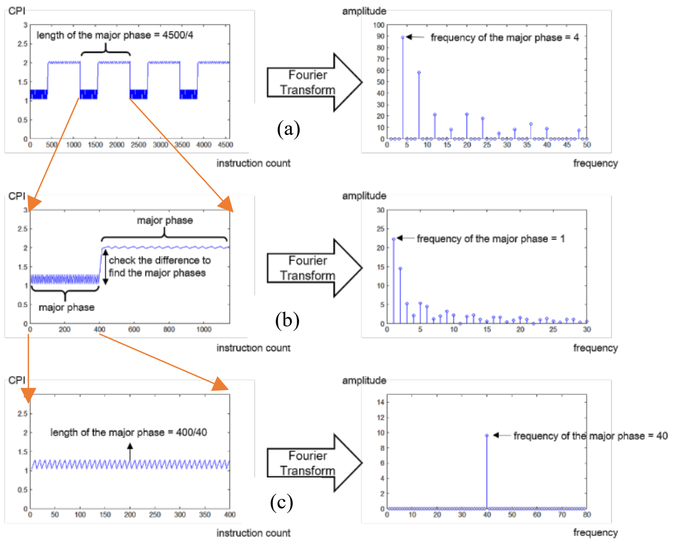

Fig. 2: An example demonstrating the proposed Frequency Domain Analysis method for systematic phase identification.

To address the limitations of time-quantum approaches, it is observed that program phases can be fully decomposed into basic blocks, as the transition point between any two program phases is always a branch instruction. Since there are no branch instructions within a basic block, the number and types of instructions are fixed, resulting in relatively consistent performance measures. To calculate the practical performance value, p_b, of a basic block b, which considers factors like cache hit-miss and other optimization effects, we compute a weighted average of the performance values associated with the time quanta in which the basic block b is executed. This is expressed by the formula: $p_b = (\sum_q n_q v_q)/(\sum_q n_q)$, where v_q is the performance value of time quantum q, and n_q is the number of times basic block b occurs in time quantum q. Our method's accuracy is verified to align well with actual values.

To efficiently identify basic blocks as phase starting points at runtime, we simply trace branch instructions, eliminating the need for complex control flow graph analysis. When the memory address of the first instruction within a basic block is used as the unique identifier for that basic block, phase detection involves identifying the starting point by comparing consecutive basic blocks using their performance values. We identify potential phase starting points by marking pairs of basic blocks with performance value differences exceeding a predefined threshold, derived from the main phase's dynamic performance value. We then use this information along with the length of the main phase obtained from frequency domain analysis to precisely identify program phases in the execution trace.

Our basic block information serves a dual purpose: it aids in identifying not only phases but also loops and functions. Loop starting points are detected by checking if branch instructions point backward, while function starting points are identified by recognizing function call instructions, like jump-and-link or JAL in MIPS. Since loops and functions share branch instruction identification, their starting points align with certain basic blocks. We annotate each basic block to indicate if it's the starting point of a loop, function, or neither. This information helps relate identified program phases to loops or functions, which can lead to more effective system optimization.

In practice, we create a hierarchical program phase table during recursive frequency domain analysis, recording head basic blocks, lengths, and performance values for each phase. Basic blocks are marked with their associated phases. This information simplifies runtime phase-detection for scheduling and optimizations. When executing a new basic block, we check if it's a phase head. If so, we access the phase table for its performance value (CPI) and code structure (loop or function).

Alternatively, we can embed phase information into the application code using custom instructions. These instructions are placed at the start of the head basic blocks and include performance and code structure details, functioning like NOP instructions, signaling the start of a new phase during runtime. We summarize the algorithm below.

BBFDA Algorithm

1. Profile the target program (by execution or simulation).
2. Apply frequency-domain analysis on the time-domain waveform to generate the frequency spectrum.
3. Calculate the phase length by the spectrum.
4. Scan the profiling trace to find basic blocks that match the phase length.
5. Check if there are minor phases in this major phase.
6. If yes, go to step 2; otherwise, the process is completed.

Basically, we profile the target application through execution or simulation, obtaining basic block traces, CPI values, and the time-domain performance waveform. Next, we use frequency domain analysis on the waveform to identify the main spectrum and calculate the phase length. We then search for the main phase's head basic block in the waveform. If there's only one occurrence of the main spectrum, we identify high and low-performance phases. We stop if the waveform is flat; otherwise, we recursively analyze the phase segment. Our method, implemented and verified, yields promising results discussed in the next section.

III. Results and Conclusions

We used SPEC CPU2017 benchmarks [11], conducted simulations on an Intel Core i5-3320m 3.30GHz machine, and modified SimpleScalar [12] to gather basic block traces and CPI values. We applied recursive frequency domain analysis to extract hierarchical program phases for evaluation.

We compared our BBFDA method with time-quantum-based (TQ) approaches (TQ-100M and TQ-1B) using CPI estimation errors. BBFDA showed an average error rate of 4.45%, significantly better than TQ-100M (7.88%) and TQ-1B (12.54%). BBFDA can be practically used for system optimization. The maximum error was 18.4% for *mcf*, while the least error was 0.42% for *gromacs* due to varying performance patterns.

We also verified that our approach maintained minimal variations when different inputs were used across benchmark cases, confirming its effectiveness in identifying program phases with varying input data through basic block information.

We compared CPI waveforms for *mcf* and *gromacs* cases. The time-quantum-based approach suffered in cases with higher CPI variations, while BBFDA effectively tracked dynamic performance behaviors using basic block analysis. BBFDA precisely captured phase lengths, while determining an appropriate quantum size remained challenging for time-quantum methods.

In summary, the proposed BBFDA has been verified to be highly effective in the identification of program phases and the estimation of performance waveforms. Notably, the BBFDA method maintains its robustness when confronted with diverse input data, accurately capturing phase durations, making it an excellent choice for program performance prediction and system optimization.

References

[1] Michael C. Huang, J. Renau and J. Torrellas, “Positional adaptation of processors application to energy reduction,” Proceedings of the 30th annual international symposium on Computer architecture (ISCA), ACM, 2003.

[2] A. Sembrant, D. Black-Schaffer ,E. Hagersten, “Phase guided profiling for fast cache modeling,” Proceedings of the Tenth annual IEEE/ACM international symposium on Code generation and optimization (CGO), ACM, 2012.

[3] S. Padmanabha, A. Lukefahr, R. Das, S. Mahlke, “Trace based phase prediction for tightly-coupled heterogeneous cores,” Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), ACM, 2013.

[4] T. Sherwood, E. Perelman and B. Calder, “Basic block distribution analysis to find periodic behavior and simulation points in application,” Proceedings of 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2001.

[5] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, “Automatically characterizing large scale program behavior,” Proceedings of the 10th international conference on Architectural support for programming languages and operating systems (ASPLOS), ACM, 2002.

[6] T. Sherwood, S. Sair and B. Calder, “Phase tracking and prediction,” Proceedings of the 30th annual international symposium on Computer architecture (ISCA), ACM, 2003.

[7] J. Lau, S. Schoenmackers and B. Calder, “Transition phase classification and prediction,” Proceedings of 11th International Symposium on High-Performance Computer Architecture (HPCA), IEEE, 2005.

[8] Z. Fang, J. Li, W. Zhang, Y. Li, H. Chen, B. Zang, “Improving dynamic prediction accuracy through multi-level phase analysis” Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES), ACM, 2012.

[9] J. Lau, E. Perelman and B. Calder, “Selecting software phase markers with code structure analysis,” Proceedings of the International Symposium on Code Generation and Optimization (CGO), ACM, 2006.

[10] Y. Jiang, Eddy Z Zhang, K. Tian, F. Mao, M. Gethers, X. Shen, Y. Gao, “Exploiting statistical correlations for proactive prediction of program behaviors,” Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization (CGO), ACM, 2010.

[11] John L. Henning, “SPEC CPU2006 benchmark descriptions,” ACM SIGARCH Computer Architecture News 34.4 (2006): 1-17.

[12] D. Burger, Todd M. Austin, “The SimpleScalar tool set, version 2.0,” ACM SIGARCH Computer Architecture News 25.3 (1997): 13-25.