A Frequency-Domain Program Phase Identification Approach for Execution Time Prediction

***Abstract*—In this paper, we present a systematic approach that transforms the program execution trace into frequency-domain and precisely identifies each program's unique execution phases. We annotate the program phase information into program code to represent the starting point and execution characteristics of each program phase. Each program phase information is used to predict runtime program execution time. With the execution time prediction, we can explore and develop more intelligent program task schedulers of deep learning frameworks to optimize inference performance.**

Keywords—basic block, frequency-domain analysis, program phase

I. INTRODUCTION

As users demand ever higher system performance, designers consider the runtime behaviors of target applications to optimize system performance. Particularly for embedded system designs for specific applications, such as deep learning neural networks, an accurate application behavior model is critical for task scheduler optimization.

Regarding behavior pattern, we observe that in practice, most applications have repetitive program phases, which alternatively occur in each execution run. In general, each program phase exhibits consistent performance value. In Figure 1 we show an illustrative example which has three different program phases, as indicated by labels A, B, and C, which alternatively occur through the execution run and each phase performs at a certain CPI (Cycles per Instruction) value. In practice, each different program phase implies specific data computing and access behaviors.

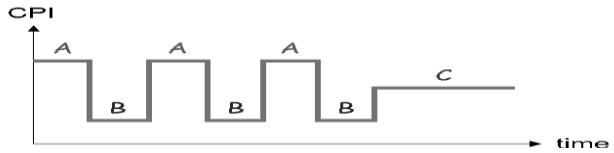

Figure 1. An example with three program phases. CPI varies according to phase.

Many researchers have developed various program phase prediction techniques in an attempt to consider and take advantage of the repetitive program behaviors for system performance and power optimization [1][2][3][4][5][6]. Although encouraging results were observed, there was no clear agreement on defining what a program phase is.

These past approaches, in general, attempted to divide the application execution trace into fixed-length execution segments and merge those consecutive segments of same performance metric into a phase. The chosen fixed length can be a time quantum in terms of a fixed number of instructions, or a fixed number of basic blocks executed. The advantage of having fixed-length segments is the convenience to perform the evaluation metric and final analysis. Nevertheless, a major issue is that the fixed-length segments may not match well with the actual phase boundaries, so phases of shorter length are ignored and phases with boundaries in between time quantum can be misjudged. Another dilemma is that there is no useful information for optimal fixed length decision before performing the analysis. Besides, there is no single optimal time quantum length for all cases.

To define what program phase is, we develop a systematic approach for program phase identification with no ambiguity based on the program execution behaviors. Essentially, we observed the fact that the repetitive behaviors are easy to identify in the frequency-domain analysis. Because the application behaviors follow closely with the code functions executed, a program phase strongly correlates to a program code structure, such as basic blocks, loops, and functions. We apply the spectrum analysis of the signal-processing field to program code structures for precise phase boundaries identification.

A program code structure of a basic block is a straight-line code sequence with no branches except to the entry and from the exit instruction. This feature makes a basic block amenable to analysis execution time. Program code structures of loops and functions exhibit stable behavior in repeated executions, so they are also suitable for execution time estimation.

Because basic blocks are the most basic execution units with consistent behavior, we compute the average CPI value of each basic block and use one basic block as one time-unit to index the performance trace. Therefore, we can avoid the choice of granularity problem of the time-quantum approaches.

The proposed approach has two contributions. First, our frequency-domain analysis method allows precise identification of program phases, including specific phase starting point, execution length, and behavior characteristics. Second, we clarify the association between program phases and code structures. We also greatly improves the execution time prediction accuracy for program task schedulers to optimize system performance.

II. RELATED WORK

There are two types of phase identification approaches. First, a time-quantum-based approach divides a program execution into fixed-length intervals. Each interval represents a phase segment, which is to be merged into a phase. Second, a program-structure-based approach uses the basic structures, such as loops or functions, to determine program phases. We will discuss the pros and cons of the two methods in the following.

A. Time-quantum-based Approaches

A time quantum is a fixed-length contiguous execution interval. A time-quantum-based approach divides a program execution trace into non-overlapping quanta for phase analysis. In general, an evaluation interval includes a fixed number of instructions or basic blocks. Then, one phase is a group of quanta of similar performance metric, such as CPI, cache miss rate, branch miss rate. Nevertheless, the term "similar" is not precisely defined, and it implies impreciseness and uncertainty. The Basic Block Vector (BBV) approach [8] [9][10] is one of the most representative time-quantum-based techniques.

The BBV method observed that the program phase behavior highly correlated with the patterns of basic blocks. The program behavior is a result of executing program code, which is composed of basic blocks. The BBV approach identifies the boundary between phases by recording the footprint of basic blocks and checking if the compositions (or named signatures) of any consecutive basic block vectors (a fixed number of basic blocks) have a difference more than a given threshold value.

A common problem of the BBV approach is that quanta are often across program phase boundaries and cause the transition phase problem [11], so the performance estimation of phases have larger errors.

Fang et al. [12][15] observed that phases were composed of a hierarchical structure and proposed a Multi-Level Phase Analysis method that classified phases into fine-grain (an inner-loop) and coarse-grain (an outer-loop or function) phases. A coarse-grain phase includes stably distributed fine-grain phases. The multi-level phases approach has better performance estimation accuracy under practical cases. However, with the differentiation of coarse-grain and fine-grain phases still cannot cover the necessary time quantum granularities of the general cases.

B. *Program-structure-based Approaches*

Another type of approaches is program-structure-based approaches. These approaches focus on identifying phases based on loops and functions of program structures. Loops and functions are repeatedly executed with consistent behavior, so the program-structure-based approaches assume that loops and functions are basic components of a phase and focus on finding the transition that connects basic components to form phases.

For instance, Huang et al. [1] followed the basic idea of phases and attempted to configure combinations of function calls for performance or energy consumption improvement.

Lau et al. [7][13] used a Hierarchical Call-Loop Graph analysis technique to identify program phases from loops or functions. On the graph, each node represents a loop or a function. Each edge represents an execution path from a node to another node and is associated with a call count information, which includes the average and standard deviation of instruction number across various invocations. The edges with lower deviation values represent consistent performance behaviors of program codes, so Lau grouped the nodes connected by the edges as a phase.

In contrast, Jiang et al. [14] did not explicitly identify program phases. Jiang attempted to identify the correlations among repeatedly executed loops and functions under various inputs. For instance, the analysis may find that a certain function's call count is almost a fixed ratio of a loop's trip-count under different inputs tested, and then this fixed ratio is adopted for runtime program behavior prediction.

A general problem of the program-structure-based approaches is the difficulty in calculating a precise performance value (e.g., CPI) due to the variations of different execution runs.

In summary, the above-mentioned existing approaches all try to use a pre-conceived phase pattern, such as a fixed phase length, a fixed basic block vector size, or connections to loops or functions, to determine a program phase, but the pre-conceived phase pattern generally may not match to the real program phase patterns. Hence, we propose a frequency-domain program phase identification approach to solve the problem as elaborated in the following.

III. FREQUENCY-DOMAIN PROGRAM PHASE ANALYSIS METHOD

Normally, a human being can recognize program phases, simply by viewing the waveform. For example, the waveform in Figure 2(a) has a high-performance phase and a low-performance phase. However, the challenge is to identify such patterns automatically through a systematic algorithm. To address this challenge, we leverage the fact that repeated patterns in time-domain often show up as a particular frequency response.

The proposed approach can precisely identify phase starting point, phase length, and the average program performance of each phase. Note that since CPI is a commonly used performance measure, we simply use CPI for program behavior characterization in the following discussion, but the approach is not limited from other performance measures.

A. *Frequency-domain Analysis*

With a given program execution trace for analysis, we first create a time-domain waveform as shown in Figure 2(a), in which the vertical axis represents the performance value and the horizontal axis represents the time quantum sequence number. Each sample point on the waveform represents the performance value obtained at the corresponding instruction count of the program execution in the time quantum.

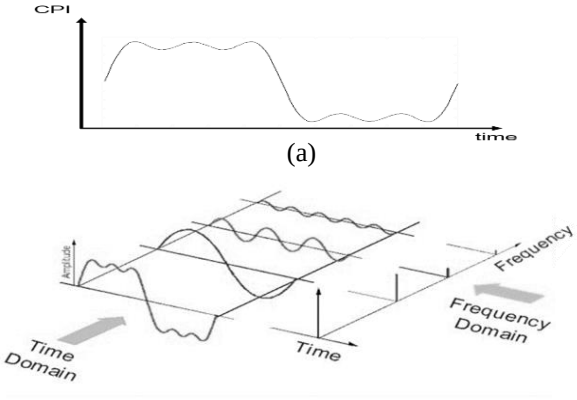

Figure 2. (a) A time-domain performance waveform example. (b) The low frequency spectrum in frequency domain corresponds to the major phase in time-domain.

Since a phase by definition is a frequently occurred pattern, the major phase normally dominates the performance waveform. Once the time-domain performance waveform is transformed into frequency-domain, the major phase normally shows up in the low-frequency spectrum, as illustrated in Figure 2(b). In fact, each frequency response corresponds to certain repeated program behavior patterns shown in the time-domain performance waveform.

In practice, a Fourier Transform tool [18] is used to perform frequency-domain analysis (FDA). We use the example in Figure 3 to illustrate the proposed phase identification algorithm. For the frequency-domain spectrum shown on the right-hand side of Figure 3, the horizontal axis indicates the pattern occurrence number in the time-domain

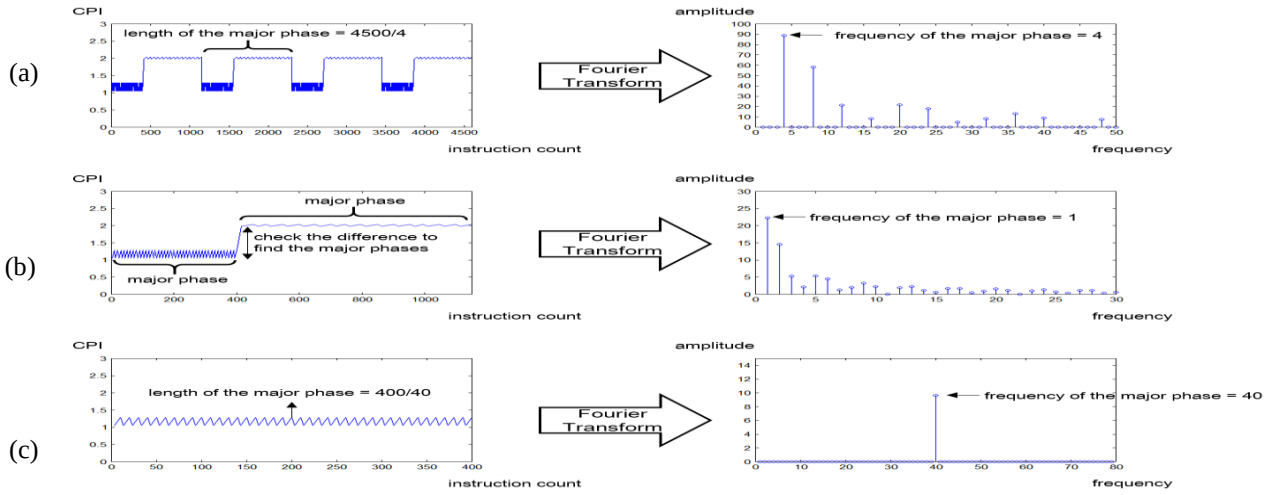

Figure 3. An example illustrates the proposed Frequency Domain Analysis method for systematic phase identification.

execution period. The vertical axis represents the dynamic performance value. For clarity, we always ignore the first zero-occurrence value, which is the average performance value of the whole execution. In circuit analysis, this value is the DC (direct current) value.

After removing the DC value, we locate the spectrum with the largest dynamic performance value, named the main spectrum. The main spectrum corresponds to the main program phase. We use the value of the main spectrum to calculate the length of the main phase. For example in Figure 3(a), the value of the main spectrum is 4, which means the main phase occurs four times. Since the total execution length is 4,500 instructions, we have the main phase length equals to 1,125 instructions or 4,500 instructions divided by 4. Therefore, the length of program phase can be identified more easily in the frequency domain. Since program phases are a hierarchical structure [12], the primary phase may contain many secondary phases. Therefore, we simply apply a recursive approach to analyze the primary phase identified using the same frequency-domain transform approach and identify smaller phases within the primary phases.

Note that for the special case as shown in Figure 3(b), when the occurrence of the main phase is one, this case indicates no repeated pattern. Then we check if the difference between the performance measurement of two neighboring phase is larger than the given variance. Therefore, we can identify the boundary of two separated phases, a high-phase and a low-phase.

Another special case occurs when the whole waveform is almost flat, and the whole spectrum is also almost flat (after excluding the DC portion). The recursive exploration stops when the whole waveform is one single phase. For example, as shown in Figure 3(c), after the third recursive step analysis, we find a pattern which repeats every 10 (= 400/40) instruction count units, while the CPI variation is within 0.3. In this case, we may decide to treat this as a last level phase instead of exploring further.

B. *Basic Block Performance Value*

Since the separation of any two program phases is always a branch instruction, a program phase can be decomposed into basic blocks. We develop a method that converts the measured performance numbers to each basic block.

There are no branch instructions inside a basic block, so the number of instructions and instruction types are fixed, and the performance measure of each basic block is consistent. To compute the performance value p_b of a basic block b, we simply take the weighted average of the time quanta that the basic block b occurs in. Assume that the performance value of time quantum q is v_q and the basic block b occurs n_q times in the time quantum q. Then we have

$$p_b = (\sum_q n_q v_q)/(\sum_q n_q) .$$

With the basic block performance values, we redraw the performance waveform using the basic block instruction count as horizontal axis index instead of a quantum sequence. With this basic-block instruction indexed waveform, we avoid the ambiguous phase boundary problem and precisely set the phase boundary to align with a basic block.

C. *Basic Block as Phase Starting Point*

Because the starting instruction address of a basic block is based on a branch instruction, therefore, we follow branch instructions during program execution to identify basic blocks at runtime instead of doing complicated control flow graph

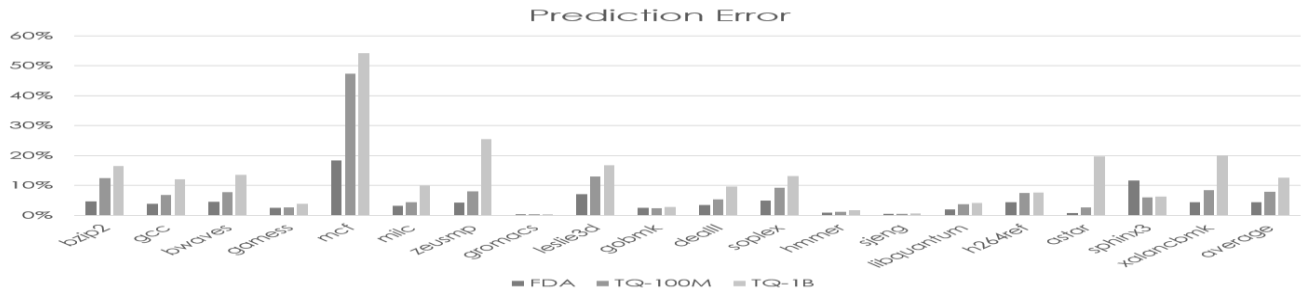

Figure 4. The error rate of program behavior prediction.

analysis. For convenience, we simply use the first instruction address of a basic block as its label for easy identification.

With the basic block performance values, we compare all consecutively occurring basic blocks and mark the pair whose difference exceeds a given threshold, derived from the performance value of the main phase. The basic block of the pair with a large difference is taken as a candidate for the starting point of a phase.

Additionally, with the length of the main phase identified from the frequency-domain analysis, we then check the execution trace (time-domain waveform) with the candidate basic blocks to identify program phase precisely.

D. Code Structure and Phase

The basic block information is also used to identify loops and functions. The starting point of a loop is identified by checking whether the branch instruction is pointing backward, i.e., to a target instruction address before the branch instruction address. Similarly, the starting point of a function is identified by checking if the branch instruction is a function call instruction, such as the jump-and-link or JAL instruction in MIPS.

Since loops and functions are identified through branch instructions, the starting points of loops and functions must be starting points of certain basic blocks. Therefore, we may annotate each basic block whether it is the starting point of a loop or function or none of them.

Because each identified program phase has a head basic block, which now also contains a code structure information, we can find whether a phase encountered is related to loop or function.

E. Hierarchical Phase Information

Based on recursively executing the frequency-domain analysis to identify phases, we construct a hierarchical program phase table recording the phase's head basic block, length and performance value. We use the phase table to detect program phases at runtime. When executing a new basic block, we first check if the basic block is the head of any program phase. If so, we look up the phase table to obtain the performance value (CPI) and code structure (loop or function) information.

IV. EXPERIMENTS

To verify our proposed prediction method, we adopt SPEC CPU2006 [17] benchmark suite as the target applications and adopt the simulation model of SimpleScalar [16] to collect target program information. We modify SimpleScalar to collect basic blocks trace, compute CPI values of basic blocks for each test case. With the trace and CPI values, we apply the recursive frequency-domain analysis to extract hierarchical program phases.

A. Evaluation Results

To validate the accuracy of the proposed approach, we adopt the commonly used time-quantum length, 100 Million, and 1 Billion instructions to calculate the runtime CPI waveform for each test case for comparison. We then apply the proposed method discussed in Section 3 using the CPI values annotated on the basic blocks to estimate the CPI waveform. Note that the basic block CPI values are computed once based on a given reference input, and the results are used for runtime performance estimation. For a fair comparison, we use the golden simulation results as the baseline for error calculation.

Figure 4 displays the CPI estimation errors of our approach and the time-quantum based approaches (TQ-100M and TQ-1B). The results show that the average error rate of our proposed BBFDA is 4.45%, lower than that of TQ-100M (7.88%) and TQ-1B (12.54%). Therefore, the accuracy of the proposed approach is about two to three times less than the time-quantum approaches.

Specifically for our approach, the *mcf* case gives the maximum error of 18.4%, and the *gromacs* case gives the least error of 0.42%. The difference is due to the variations in performance patterns. For *mcf* case, the CPI waveform exhibits large variations, so the prediction is less accurate.

B. Discussions

Shown in Figure 5 is the CPI waveform of the *mcf* benchmark case. Note that the lower CPI (0.2-0.5) phase exhibits large variations from the average value. Specifically, the high CPI value is approximately twice than the least value. The high variation cases have serious impact on the time-quantum based approach. For a larger-sized time quantum, the CPI variation, in general, is higher since more components of different performance values are included. Therefore, the one-billion quantum-size shows higher error than the 100-million quantum-size. The average performance value of time quanta cannot properly represent dynamic performance behaviors, whereas our approach can track the dynamical performance behaviors because we avoid the choice of granularity problem.

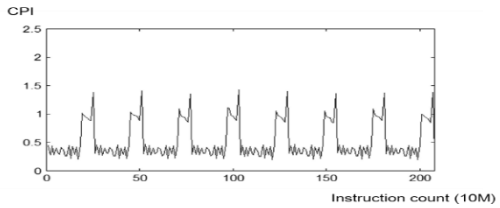

Figure 5 The CPI waveform of the *mcf* benchmark case.

We also observe that the performance estimation error rate is higher for those cases with many phases. The reason is that more phases imply higher variations of performance behaviors. In general, our approach performs more accurately than the traditional time-quantum based approach.

V. CONCLUSION

In this paper, we present a frequency-domain program phase identification approach for execution time prediction. The low-frequency spectrum is used to find the major program phase precisely. Our approach eliminates the issue of granularity choices and is more robust than the time-quantum based approach.

Experimental results show that our approach provides accurate performance estimations with an average of only 4.45% error rate as compared with the golden simulation references.

Additionally, our approach provides detailed hierarchical phase information annotated on basic blocks. For future work, we plan to extend the proposed method for facilitating program task schedulers of deep learning frameworks to optimize inference performance.

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] N. Peleg, B. Mendelson, "Detecting change in program behavior for adaptive optimization," Proceedings of 16th International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2007.

[3] R. Sarikaya, C. Isci, A. Buyuktosunoglu, "Runtime workload behavior prediction using statistical metric modeling with application to dynamic power management," Proceedings of 2010 IEEE International Symposium on Workload Characterization (IISWC).

[4] L. Sawalha, S. Wolff, Monte P. Tull, Ronald D. Barnes, "Phase-guided scheduling on single-ISA heterogeneous multicore processors," Proceedings of 14th Euromicro Conference on Digital System Design (DSD), IEEE, 2011.

[5] 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.

[6] 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.

[7] J. Lau, J. Sampson, E. Perelman, G. Hamerly, B. Calder, "The Strong correlation Between Code Signatures and Performance," Proceedings of 2005 International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2005.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] FreeMat - http://freemat.sourceforge.net/