Improving Dynamic Prediction Accuracy Through Multi-level Phase Analysis

Abstract
Phase analysis, which classifies the set of execution intervals with similar execution behavior and resource requirements, has been widely used in a variety of dynamic systems, including dynamic cache reconfiguration, prefetching, and cache detection. While phase granularity has been a major factor to the accuracy of phase prediction, it has not been well investigated yet and most dynamic systems usually adopt a fine-grained prediction scheme. However, such a scheme can only take account of recent local phase information and could be frequently interfered by temporary noises due to instant phase changes, which might notably limit the prediction accuracy.

In this paper, we make the first investigation on the potential of multi-level phase analysis (MLPA), where different granularity phase analysis are combined together to improve the overall accuracy. The key observation is that a coarse-grained interval, which usually consists of stably-distributed fine-grained intervals, can be accurately identified based on the fine-grained intervals at the beginning of its execution. Based on the observation, we design and implement a MLPA scheme. In such a scheme, a coarse-grained phase is first identified based on the fine-grained intervals at the beginning of its execution. The following fine-grained phases in it are then predicted based on the sequence of fine-grained phases in the coarse-grained phase. Experimental results show such a scheme can notably improve the prediction accuracy. Using Markov fine-grained phase predictor as the baseline, MLPA can improve prediction accuracy by 20%, 39% and 29% for next phase, phase change and phase length prediction for SPEC2000 accordingly, yet incur only about 2% time overhead and 40% space overhead (about 360 bytes in total).

To demonstrate the effectiveness of MLPA, we apply it to a dynamic cache reconfiguration system which dynamically adjusts the cache size to reduce the power consumption and access time of data cache. Experimental results show that MLPA can further reduce the average cache size by 15% compared to the fine-grained scheme.

Categories and Subject Descriptors C [Computer Systems Organization]: General - modeling of computer architecture

General Terms Design, Measurement, Performance

Keywords Dynamic prediction, Phase analysis, Multi-level, Cache reconfiguration

1. Introduction
Programs generally have abundant execution intervals with repetitive behavior. Such intervals, which are consecutive portions of program execution, are usually with similar performance characteristics and resources requirements. The repetitive behavior in execution intervals exists in multiple levels or granularities: fine-grained ones such as inner-most loop body, coarse-grained ones such as outer-most loop body and recursively or repetitively invoked functions. Accurately capturing such repetitive behavior could enable many online optimizations, such as dynamic cache reconfiguration, power reduction, software debug acceleration for multi-core architectures and data prefetching.

Phase analysis has been established as a standard technique that dynamically characterizes the set of execution intervals with similar performance behavior into the same phase. To perform phase analysis, the execution of a program is first divided into non-overlapping execution intervals. Then, intervals with similar behavior (e.g., similar IPC, and/or cache miss rates) are classified into the same phase (i.e., phase classification). Since intervals in the same phase have similar performance characteristics and resource requirements, many adaptive optimizations can be applied through predicting the results of future phase classification (i.e., phase prediction) without expensive detailed modeling or analysis.

The prediction accuracy of phases, which is the proportion of correctly predicted intervals (i.e., the predicted phase ID of the interval equals the actual phase ID) in the total execution intervals, directly influences the effectiveness of dynamic systems using phase analysis. In phase analysis, the prediction accuracy is mainly affected by two key factors: phase granularity, which is the size (e.g., instruction counts) of an execution interval composing a phase; phase metrics, which are metrics identifying which behavior is repetitive, including control flows such as basic-block vectors (BBVs) [22, 25], and/or data accesses such as memory reuse distance [24].

Though phase granularity has been one of the key parameters in phase analysis, there is currently little research on how it could affect the accuracy of phase prediction. Instead, most prior research usually partitions program execution into fine-grained intervals and applies a fine-grained prediction strategy. Such a strategy, called fine-grained phase analysis, exploits most recent local history information to predict future phase behavior. The main advantage of such a scheme is the flexibility in timely adjustment of the optimization strategies on the fly according to dynamic program execution behavior. To guarantee prediction accuracy, they usually use a confidence counter and set a threshold for the counter to filter irrelevant intervals. However, though flexible, fine-grained phase analysis can only take account of recent local phase information and lack global phase history information. Thus, the analysis could be frequently disturbed by temporary noises due to instant phase changes, which might notably limit the prediction accuracy.

To illustrate the limited prediction accuracy in a fine-grained prediction scheme, we use the intervals of an execution segment from the facerec benchmark in SPEC2000 as an example. As shown in Fig. 1, interval $a$ and interval $b$ are two consecutive exe-

\footnote{According to our measurement, the average size of an iteration in out-most loops of all SPEC2000 benchmarks is about 2,000M (million) instructions on average. Hence, we consider an interval size in the range of 1M to 999M instructions as fine-grained, while an interval size over 1,000M instructions as coarse-grained.}
The experimental evaluation that demonstrates the notable improvement in prediction accuracy and effectiveness of MLPA to dynamic cache reconfiguration.

The remaining of this paper is organized as follows. Section 2 describes the related work. Section 3 motivates our approach by illustrating the limitation of fine-grained phase analysis and analyzing the characteristics of coarse-grained phases. Section 4 describes our multi-level phase classification architecture. Section 5 presents our phase prediction algorithm. Section 6 evaluates the effectiveness of MLPA and its applications. In section 7, we conclude this paper with a brief remark on future work.

2. Related Work

In this section, we discuss prior work in two areas related to this paper: phase analysis and applications of phase analysis. We summarize them in Table 1 according to their metrics, granularity and their usages.

2.1 Applications of Phase Analysis

Since many programs exhibit repetitive behavior over many different metrics, phase analysis has been widely used to identify repetitive program behavioral patterns for power reduction [10, 12, 13], cache optimization [19, 24], simulation acceleration [22, 25] and software debugging [20].

Power Reduction: There have been many proposals for adaptive hardware mechanisms targeted at energy optimization. They dynamically adapt different aspects of the processors, including cache organization, issue width, voltage and frequency. Isci et al. [12, 13] used the power breakdowns to identify the power phase behavior and discussed how to use the control flow and the event counter for power behavior analysis. Huang et al. [10] proposed an adaptive hardware method to partition program execution into fine-grained phases at the procedure level for power reduction.

Cache Optimization: A research hotspot in memory system is cache optimization based on the usage pattern of a running program. Balasubramonian et al. [1] designed a phase-based system that can dynamically change cache configurations to improve performance and save power. Shen et al. [24] proposed to collect information on data reuse distance for phase analysis and adaptive cache optimizations. Lu et al. [19] proposed a runtime data cache prefetching scheme based on online phase analysis. Dhodapkar et al. [5] proposed a phase detection mechanism according to changes in instruction working sets [4] and discussed their applications for cache reconfiguration.

Simulation Acceleration: Representative sampling techniques has been one of the most efficient techniques to accelerate the architecture simulation speed. As one of the most representative phase techniques, SimPoint [22, 25] uses Basic-block Vectors (BBVs) as the metric for identifying phases and simulation point selection.
The software phase marker (SPM) [18] method is also a BBV-based technique. Instead of using fixed-length intervals, it selects fine-grained phases according to loop or procedure boundaries for simulation point selection and cache reconfiguration.

Software Debugging: Dynamic software debugging tools for multi-threaded programs have been widely used due to their accuracy and immense help to programmers. However, a significant impediment to their adoption is their runtime overhead. Marino et al. [12] showed that BBV-based phase analysis is efficient in dynamic systems and presents the observation on the distribution of the execution phases. Huffmire et al. [11] used hashed memory accesses as the metric to partition program execution into fine-grained phases. Georges et al. [8] used an offline implementation of this algorithm to characterize phase behavior of Java programs. Duesterwald et al. [7] used hardware counters for phase prediction to find phase behavior. Cho et al. [3] used wavelet method to characterize phase complexity and the changing of program behavior.

### 3. Motivation of Multi-level Phase Analysis

This section analyzes the limitation of fine-grained phase analysis and presents the observation on the distribution of the execution intervals of a program, which motivates the design of our MLPA scheme.

#### 3.1 Evaluation Methodology

We performed our analysis using several SPEC2000 programs with reference inputs. The programs include gzip, mgrid, gcc, equake, facerec, lucas, bzip2, ammp and mcf. The major reasons that we chose these programs are as follows. First, they were widely used in prior similar research, such as [17, 18, 24, 26]. Moreover, Aashish et al. [23] demonstrated that the programs in SPEC2000 can be clustered into 8 subsets based on similarity analysis. Our selected programs basically cover those subsets, which can effectively represent the entire suite. At last, as analyzed in [21], the programs in SPEC2006 have similar phase behavior and phase distribution with those in SPEC2000. Hence, to make a comparable study with prior literatures, we choose SPEC2000 instead of SPEC2006 in our study. The statistics of phase behavior in these workloads were measured using SimpleScalar. The base simulation configuration is detailed in Table 2, which is the same as that in [17, 22, 25, 26] and [18]. The geometric mean is used to compute the average results (AVG).

#### Fine-grained Phase Prediction Schemes to Study

In a fine-grained phase prediction, the execution history is usually composed of most recent phase information, including past phase IDs and phase lengths. Based on different history information used for prediction, there are two widely-used prediction methods including last value and Markov. As analyzed by Lau et al. [17], the last value (LV) prediction, which simply predicts the phase of the next interval as the same phase of the last executed interval, incurs little hardware or software overhead. The basic idea of Markov Model is
that the next state of a system is only related to the last set of states. Hence, a Markov scheme of order k predicts the next phase based on the phase behavior of k previous phases. The Markov model is a classical predictor, which is easily implemented in hardware or software and has been widely used to predict various events, such as branch prediction [2] and prefetching [14].

Since the last value (LV) strategy cannot be used to predict phase change and phase length, we only use the Markov model as the baseline for the evaluation of those two parts. Markov predictor with order 1 (Markov-1) and order 2 (Markov-2) [17, 26] are two most widely used models in prior research and Markov-2 is a more accurate scheme. We thus focus on Markov-2 in this paper and use Markov-2 (Mkov) with run length information [26].

### 3.2 Limitations of Fine-grained Phase Analysis

![Figure 2: The prediction accuracy of fine-grained prediction scheme using the Markov Model for three phase prediction types: next phase prediction (NP), phase change prediction (PC) and phase length prediction (PL).](image)

To illustrate the possible inaccuracy in a fine-grained prediction scheme, we evaluated the Markov model using the evaluation methodology described above. Fig. 2 shows the prediction results of three prediction types: next phase prediction (NP), which predicts the phase ID of the next interval; phase change prediction (PC), which predicts the phase ID that will occur after the next phase change; and phase length prediction (PL), which predicts the length of next phase. The prediction accuracy is defined as the number of correctly predicted intervals divided by the number of totally executed intervals.

As shown in the figure, the average prediction accuracy is 66%, 26% and 65%, respectively. The major reason behind the low prediction accuracy is that a program usually contains multi-level repetitive execution behavior, such as nested loops and recursive functions, which leads to not only fine-grained phase behavior, but also coarse-grained phase behavior exhibiting in program execution. Thus, not considering the multi-level execution behavior would lose many prediction opportunities. Consequently, there are 21%, 66% and 33% phases that are not predicted for NP, PC and PL prediction accordingly.

After a detailed analysis of the phase distribution of typical program execution, we found that different coarse-grained intervals (e.g., outer-most loop) usually consist of different sequences of fine-grained intervals. Hence, the prediction tends to be inaccurate if using the fine-grained phase information in one coarse-grained phase to predict the fine-grained phase behavior in another coarse-grained phase, which has been shown in Fig. 1.

Even if two consecutive coarse-grained intervals belong to the same coarse-grained phase, it might not be accurate to use the fine-grained phase information at the end of the previous interval to predict the fine-grained phase behavior at the beginning of the next interval. Fig. 3 illustrates such a situation by showing the phase behavior and execution sequence of the gzip benchmark from SPEC2000. As shown in the figure, 1, 3, 5, 6, 7, 8 and 9 are fine-grained phase IDs and both coarse-grained interval a and coarse-grained interval b belong to the same coarse-grained phase. Since the fine-grained phase behavior at the end part of coarse-grained interval b is not similar with that at the beginning part of coarse-grained interval a, both last value prediction and the Markov prediction achieve poor prediction accuracy (lower than 45% based on our evaluation results).

### 3.3 Multi-level Phase Analysis: Observation and Motivation

To gain insight into possible solutions to increase the accuracy in phase prediction, we studied the distribution of phases in SPEC2000 and found that the sequences of fine-grained phases are similar and stable for different coarse-grained intervals that correspond to the same phase. According to our measurement of SPEC2000, the identical sequences of fine-grained phases are more than 80% for different coarse-grained intervals in the same phase.

**Observation in Phase Classification:** Based on the characteristics of phase distribution in SPEC2000, we found that the phase of a coarse-grained interval can be accurately identified based on the executed fine-grained intervals at the beginning of its execution. Hence, instead of identifying a coarse-grained phase by executing all fine-grained intervals within the coarse-grained interval, we can use a few fine-grained intervals at the beginning of a coarse-grained interval to identify which coarse-grained phase it belongs to. Fig. 4 shows the accuracy of coarse-grained phase identification based on different number of fine-grained intervals in the front of them. As the results shown, when using five fine-grained (10M) intervals to predict the coarse-grained phase, we can correctly identify more than 95% coarse-grained phases, which is close to the results of ten 10M intervals. Hence, the result would be quite accurate if using five fine-grained intervals for phase identification. By contrast, the average number of fine-grained intervals is about 200 in a coarse-grained phase (as described in footnote 1).

![Figure 4: The prediction accuracy of coarse-grained phase based on number of fine-grained intervals at the beginning of their execution.](image)
Observation in Phase Prediction: The pervasive existence and stable distribution of coarse-grained phases in many programs open opportunities of exploiting the coarse-grained phase behavior in phase prediction. Hence, not only the fine-grained phase behavior, but also the coarse-grained phase information could be leveraged to predict both the fine-grained and coarse-grained phases. Specifically, we can first identify which (coarse-grained) phase current coarse-grained interval belongs to, based on the beginning phase sequence of already executed fine-grained intervals. Afterwards, the rest of fine-grained phase sequence can be predicted based on the recorded phase sequence in the coarse-grained phase that corresponds to the current interval.

Fig. 3 also shows an example of using coarse-grained phase to predict fine-grained phases. As shown in the figure, $a$ and $b$ are two coarse-grained intervals that will be classified into the same coarse-grained phase $A$. Interval $a$ is executed before interval $b$. Once the execution of interval $a$ is completed, the sequence of fine-grained phases in it will be recorded. When interval $b$ is being executed, it will be identified as phase $A$ after five fine-grained intervals at the beginning of its execution are finished, because the beginning 5 fine-grained phases matches those of interval $a$. As interval $b$ is identified as phase $A$, the rest sequence of fine-grained phases can be accurately predicted according to the recorded phase sequence in phase $A$.

4. Multi-level Phase Classification

In this paper, we use BBV [25] as the signature to classify phases because it provides a higher sensitivity and can produce more stable phases as analyzed in [6].

Phase Classification. To perform phase classification, the signature will be compared with past signatures to determine whether the current interval belongs to an existing phase or is a new one. If a match occurs, the phase ID for the matched signature table entry is returned. Otherwise, the signature is marked as a new phase and will be inserted into the signature table. The detailed algorithm and steps will be presented in next section.

4.2 Multi-level Phase Classification

Instead of classifying phases by only considering fine-grained intervals, we combine fine-grained and coarse-grained phase classification together, which forms multi-level phase classification. Fine-grained phases and coarse-grained phases are identified respectively and they are combined to identify transition phase and later phase prediction described in the next section. The phase classification architecture is shown in Fig. 5, which consists of two parts: coarse-grained phase classification and fine-grained phase classification.

![Figure 3: An example phase sequence of gzip from SPEC2000, which illustrates the limitation of fine-grained phase prediction.](image)

![Figure 5: The architecture of multi-level phase classification.](image)
stored for later phase identification. As shown in Fig. 4, five 10M intervals (50M) in the front of a coarse-grained interval can accurately represent the corresponding coarse-grained phase. Therefore, we use the signature of 50M in the front of a coarse-grained interval as its signature and the signature is compared after 50M instructions are executed.

To determine whether current signature is similar with past phases, we search the signature table. If there is an entry in the table within a similarity distance (i.e., Manhattan distance [25]) to the current signature, the interval of the current signature will be classified into as an existing phase. In this process, the threshold of similarity distance is 25%, which is also the same as that in [17]. If a match occurs, the value in the signature table will be replaced with the current signature. Otherwise, the current signature will be inserted into the table as the representative of a new phase. Multiple signatures in the signature table may satisfy the similarity threshold. In this case, we choose the phase whose signature is most similar to current signature.

To decide the number of entries for coarse-grained phase classification, we measured the coarse-grained phase data. The data are shown in Table 3. Cphase Num is the number of coarse-grained phases in each benchmark and Cintval Num is the average interval number belonging to each coarse-grained phase. According to our test, the average phase number of coarse-grained phases in SPEC2000 is three and the largest phase number is eight. Since the phase number of most benchmarks is not larger than three, we set three entries to store past signatures for coarse-grained classification. When the table is full, a First-In-First-Out strategy is applied for replacement.

Table 3: Statistics of Coarse-grained Phase.

<table>
<thead>
<tr>
<th></th>
<th>Cphase Num</th>
<th>Cintval Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>3</td>
<td>47</td>
</tr>
<tr>
<td>gcc</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>facerec</td>
<td>2</td>
<td>63</td>
</tr>
<tr>
<td>bzip2</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>mcf</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>mgrid</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>equake</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td>lucas</td>
<td>2</td>
<td>61</td>
</tr>
<tr>
<td>ammp</td>
<td>2</td>
<td>86</td>
</tr>
<tr>
<td>Avg</td>
<td>3</td>
<td>40</td>
</tr>
</tbody>
</table>

4.3 Transition Phase

Generally, consecutive phases, where the same phase appears consecutively for a relatively long period, are more suitable for optimizations. However, there might be some transition phases [17] between two consecutive phases. Since such transition phases are usually short and may happen rare, the optimizing strategies should not be adjusted when a transition phase occurs. Therefore, it is necessary to identify transition phases, which can improve prediction accuracy and reduce pressure on signature table.

In prior fine-grained phase classification scheme [17], the transition phases are identified through checking whether the occurrence of a phase has been larger than a threshold (e.g., eight). However, such an approach has difficulties in some conditions. For example, consider phase sequence $a, a, a, b, a, a, a, a$ and $b$, where $a$ and $b$ are two phase IDs. Phase $a$ will be identified as a transition phase if the threshold is larger than four. This leads to the waste of some optimizing opportunities on phase $a$. Even worse, it is difficult to choose a uniform threshold for different benchmarks.

In our multi-level phase classification scheme, we identify transition phases in each coarse-grained interval by checking if the fine-grained phase appears with low proportion in it. If so, the fine-grained phase is thus not important and can be considered as a transition phase. Therefore, transition phases can be identified through checking the proportion of a newly-appeared fine-grained phase in the prior coarse-grained interval. Such a design can lead to more accurate results and more optimizing opportunities. As analyzed in [17], the transition phases averagely accounts for about 6% of program execution. Therefore, we set the threshold for the proportion of fine-grained phases as 6%. If the proportion of a newly-appeared fine-grained phase is lower than 6% in the prior coarse-grained interval, it will be identified as a transition phase.

5. Multi-level Phase Prediction

This section introduces how to use multi-level phase analysis to predict future phase behavior, including next phase, phase change and phase length.

As shown in section 3, coarse-grained phases are stable in the composition and distribution of fine-grained phases. Further, a coarse-grained interval can be accurately identified according to the fine-grained intervals at its beginning execution. Therefore, based on such features of coarse-grained phases, we design a multi-level scheme to improve the accuracy of phase prediction. The basic idea is to first identify a coarse-grained phase based on the sequences of its beginning fine-grained intervals. Then, the remaining fine-grained intervals will be predicted based on the history information in the corresponding coarse-grained phase.

5.1 Saving History Information

In multi-level phase prediction, the sequence of fine-grained phases in the just-finished coarse-grained interval will be saved for future uses. There are two methods to save this information. The first one is to sequentially save the corresponding phase ID of each fine-grained interval in a coarse-grained one into an array. The second one compresses the space to store consecutive phases by using a triple $(id, beginpos, length)$. In the triple, $id$ is the phase ID of a fine-grained interval, $beginpos$ is the beginning position of this phase and $length$ is the phase length that fine-grained intervals repeat. Since a stable fine-grained phase generally lasts for a long execution time, the second one is more space-saving. Table 4 is a comparison of the space overhead of those two methods for the longest coarse-grained intervals in different benchmarks. As the data shown, the second one can significantly reduce the space overhead and such a scheme saves around 87% space compared to that saving phase ID sequentially in an array. Since the longest space requirement is 140 bytes, we use it as the length of history table. In the phase classification architecture, each coarse-grained phase item corresponds to such a history table item. While the phase signature is updated, this table is also updated.

5.2 Multi-level Phase Prediction

Before a coarse-grained phase is identified, the fine-grained intervals at the beginning of its execution are predicted based on the fine-grained phase prediction. After there are enough executed fine-grained intervals (i.e., five fine-grained intervals in this paper), the coarse-grained phase is identified. If this coarse-grained phase is a new one, the following fine-grained intervals in it will also be predicted based on the fine-grained phase prediction. Otherwise, the following fine-grained intervals are predicted based on the history information in this coarse-grained phase. Since the fine-grained phase sequence of two coarse-grained instances classified into one phase will not be always totally the same, the position of a fine-grained phase in history information has to be located. After the execution of current fine-grained interval is finished, its phase ID is identified. Based on its sequence position (interval number) in
the coarse-grained interval, its phase ID and the phase ID adjacent to it, we search the history table for the history information of corresponding coarse-grained phase. Specifically, the corresponding position in current interval and history table are first searched. If the phase ID matches, then the history information is used for the following prediction. Otherwise, several intervals adjacent to the current position are compared to find a match. In this paper, we use the first matched one as the final position. Fig. 6 is the comparison of different search distance where $d-n$ is the search distance with $n$. As the data shown, the distance length of five can achieve good time and accuracy tradeoff. Therefore, we use it as the parameter of searching.

![Figure 6: The probability of finding a match with various search distances (1, 5, 10, 15 and 20).](image)

### 6. Experimental Results

To demonstrate the effectiveness of our multi-level phase analysis (MLPA), we evaluated the improvement of prediction accuracy in next phase prediction, phase change prediction, phase length prediction and the overall overhead in space and time. We also described an example application of MLPA, dynamic cache reconfiguration, to demonstrate the effectiveness of our approaches.

#### 6.1 Effectiveness of Multi-level Phase Analysis

**Next Phase Prediction:** Next phase prediction predicts the phase ID that the next interval belongs to, which is done for every execution interval. The results are shown in Fig. 7. As the results shown, MLPA gets more accurate prediction results for most benchmarks and achieves 20% accuracy improvement on average compared to the fine-grained counterpart. The major reason is that the combination and distribution of fine-grained phases in different intervals of a coarse-grained phase is stable in most cases. Moreover, a coarse-grained phase can be accurately identified according to the execution of a few fine-grained intervals at its beginning execution. Therefore, compared to the fine-grained methods, more accurate history information is available, which makes the prediction of next phase more accurate. For mcf, the prediction accuracy degrades a little for MLPA. The reasons are two folds: first, the regular fine-grained phases lead to very accurate prediction result for fine-grained Markov-2 prediction; second, the number of coarse-grained intervals is small (about 45) and the coarse-grained phase behavior is not so stable as that in other benchmarks. In such a case, most coarse-grained intervals are identified as different phases. Therefore, the related fine-grained phases are essentially predicted using the fine-grained strategies.

**Phase Change Prediction:** Phase change prediction predicts the outcome of the next phase change, i.e., predicting which phase ID will occur after the next phase change. When such a change occurs, the optimizing strategies can be adjusted accordingly. Fig. 8 is the comparison of the results of multi-level phase prediction and those of Markov-2. As the results shown, multi-level phase prediction achieves about 39% accuracy improvement over fine-grained Markov-2, resulting in 65% accuracy on average.

**Phase Length Prediction:** Besides predicting what the next phase ID is, it is also useful to know how long the next phase will repeat. Because the execution of programs generally consists of long-term stable periods and short transition periods, knowing the length of the next phase will avoid some unnecessarily expensive reconfigurations for phases that will not execute long enough. The length of the next phase will be predicted upon the finish of current phase.

When predicting the next phase length, it is difficult to predict the exact length (i.e., the number of intervals in it). Fortunately, in many conditions, it is enough to know the approximate length of the next phase, i.e., whether it is short or long. Therefore, in this paper, we also use the classification method with Lau et al. [17]. The phase lengths are grouped into four sets: 1-15, 16-127, 128-1023, and intervals longer than 1024, which roughly corresponds to the phase lengths of 10-100M instructions, 100M-1B instructions, 1B-10B instructions and more than 10B instructions. Fig. 9 shows the comparison results of multi-level phase prediction with those of fine-grained phase prediction. As the results shown, MLPA achieves 94% prediction accuracy, which has 29% improvement over fine-grained phase prediction (65%).

**Overhead Evaluation:** Multi-level phase analysis has to analyze and predict phase behavior based on the information of both fine-grained phases and coarse-grained ones. Therefore, there will be additional space and time overhead. We thus evaluate the associated time overhead and space overhead with our multi-level phase analysis.

Our MLPA can be implemented in hardware, software and a hybrid manner. For hardware implementation, the coarse-grained phase analysis can be implemented in parallel with that of fine-grained phase. Therefore, it will not involve any additional time overhead. For software implementation, we measure the time overhead in our software implementation. Based on the measurement, the average software time overhead compared to the Markov fine-grained method is about 2% and the largest one is less than 3%.

The space overhead is fixed after the phase classification and prediction algorithm is designed for no matter software implementation or a hardware design. In our current design, the space overhead is about 40%. Since the total space requirement for fine-grained phase prediction is about 900 bytes, such a space increment is not a problem for current hardware and software platform.

#### 6.2 Data Cache Reconfiguration

Adaptive cache reconfiguration [17, 18] is a technique that dynamically reduces physical cache size without increasing the miss rate. Hence, it can reduce the energy consumption and access time of the data cache without losing performance. To illustrate the effective-
ness of our multi-level phase analysis (MLPA), we apply it to dynamic cache reconfiguration. In our current design, the basic cache mode is similar to that in prior work [18, 24]. The cache is configured to be a 256KB cache with 8-way associativity (8 banks) and each 32KB bank can be dynamically powered off or powered on. The cache reconfiguration is achieved through powering off some banks when some intervals are executed. To get the resizing information, the best cache configuration of a phase is first collected when its first two intervals are executed. The best cache configuration of a phase is decided, its best cache configuration is applied when the phase is predicted.

We compared our MLPA against the method in [17]. Fig. 10 shows the average cache size for each approach. The data shows the average data cache size used over the execution of the program. As the data shown, our MLPA approach can reduce cache size by about 15% (i.e., about 26KB) compared to the fine-grained method [17]. The major reason is MLPA can achieve better phase prediction accuracy. Though MLPA achieves more accurate prediction results for mgrid, the reconfigured cache size of the phases correctly predicted by MLPA is very close to that of the intervals executed before it. Therefore, the average cache size of MLPA is similar to that of Markov-2 for mgrid.

7. Conclusion and Future Work

In this paper, we presented a comprehensive study on the phase granularity and observed that a coarse-grained interval consists of stably distributed fine-grained intervals, which led to the design and implementation of our multi-level phase analysis system. Experimental results showed that our system can improve the prediction accuracy by 20%, 39% and 29% for next phase, phase change and phase length prediction on average, yet with little time and space overhead. To demonstrate the usefulness of our approach, we also applied it to dynamic cache reconfiguration. Experimental results showed that our system can reduce the average cache size by 15% compared to the fine-grained strategies. In our future work, we intend to apply our multi-level phase analysis framework to other systems, such as power reduction and race detection, to reduce the performance overhead associated with current tools.

8. Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was funded by China National Natural Science Foundation under grant numbered 60903015, Key Project of National 863 Program of China under Grant No. 2009AA012201, National 863 Program of China under Grant No. 2012AA010905, Key Project of Major Program of Shanghai Committee of Science and Technology under Grant No. 08dz0501600, Fundamental Research Funds for the Central Universities in China and Shanghai Leading Academic Discipline Project (Project Number: B114).

References
