Workload prediction using run-length encoding for runtime processor power management

S.W. Kim, T.M. Kim and C. Yoo

In processor power management research, workload prediction is a requisite for adjusting frequency without performance loss, and previous studies have proposed various prediction algorithms. Among them, methods based on the workload history table are lightweight and have high prediction accuracy for a variable workload. However, such prediction algorithms lose their prediction accuracy in the case of repeated workload. An improved workload prediction method using run-length encoding is proposed, which handles workload repetition. Evaluation results show that the proposed algorithm improves the prediction of repeated workload by up to 14% and also improves 4% of energy saving.

Introduction: Dynamic voltage and frequency scaling (DVFS) is one of the most efficient power-saving mechanisms for microprocessors. It reduces the power consumption of the microprocessor by simultaneously adjusting both the frequency and the voltage of the processor. Since the frequency of the microprocessor influences the execution time of applications, prediction of the next frequency according to the application workload is important.

Global phase history table (GPHT) is a significant method, which predicts workload based on history [1]. Since GPHT has a simple prediction algorithm, GPHT not only can be easily integrated with operating system (OS) task scheduler but also can achieve almost 90% prediction accuracy. However, GPHT has a serious limitation when the repeated workloads fill the history table. As GPHT records the monitored workload in the history table for prediction, the repeated workload wastes the space of the history table. As a result, the repetition of the same workload reduces the 6% of prediction accuracy [2].

In this Letter, we propose GPHTex to prevent the wastage of the history table due to the repeated workload. GPHTex reduces the space for the repeated workload by using run-length encoding (RLE). As a result, GPHTex predicts the workload more accurately than GPHT, and provides an appropriate frequency to a workload, which leads to energy saving. Our evaluation result shows that GPHTex improves the prediction accuracy and the energy savings by up to 14.37 and 4.25%, respectively, as compared to GPHT.

Background and limitations of GPHT: In DVFS research, memory transaction ratio (MTR) is a metric for adjusting the processor’s frequency and voltage. MTR explains the CPU slack caused by the clock speed difference between the processor and the memory, and can be calculated by using (1). Each term of the equation is obtained from the performance monitoring unit (PMU) that is available in modern processors such as x86 and ARM. For example, the higher value of MTR leads to the higher slack of CPU, which means that the processor should wait until the memory operation completes. Therefore, the processor can decrease the frequency according to MTR’s increment without any performance decrease.

\[
\text{Memory transaction ratio} = \frac{\text{Memory bus cycles}}{\text{Executed instructions}} \quad (1)
\]

MTR is calculated as a floating-point value between 0 and 1. To use MTR as DVFS’ frequency decision matrix, the range of MTR value is divided by the number of processor’s scalable frequencies. Each divided region is called as phase and each phase is allocated with a processor’s frequency. Using PMU and phases, OS can measure MTR while application is running. As a result, the application workload can be translated into a sequence of phases.

The principle of GPHT is based on the locality of the program, and the locality is that the previously measured sequence of phases will appear in the near future [3]. Therefore, GPHT traces the last few measured phases and stores them as history. The prediction is performed by comparing the newly measured phases with the history. Fig. 1 shows the components and the operation of GPHT. For tracing phases, a right shift register, called the global phase history register (GPHR), collects every measured phase. In particular, we call the contents of the GPHR as a phase pattern. To store phase patterns, a history table, called pattern history table (PHT), appends GPHR to the PHT and stores the corresponding prediction value into PHT Pred-n for each tag. The GPHR depth specifies the size of the GPHR.

Fig. 1 describes five operations of GPHT: (i) measure the current phase, (ii) update Pred-n, (iii) shift GPHR, (iv) update PHT tag, and (v) predict next phase. These five operations are performed iteratively. When GPHT measures the current phase Ph\(_t\) (1), it updates Pred-n (2) of PHT tag that has the same pattern as GPHR. Then GPHR shifts right to remove the oldest phase Ph\(_{n-1}\) and appends Ph\(_t\) on the left of GPHR (3). Next phase prediction (shaded area in Fig. 1) is performed by comparing GPHR with PHT tags. If the matching result is ‘yes’, that means phase pattern is in GPHT, so Pred-n (Ph\(_{n-1}\) in Fig. 1) is used to change processor’s frequency to Ph\(_{n-1}\). Note that, we denote Ph\(_{n-1}\) (circled Pred-n entry in Fig. 1) to distinguish with the next phase (Ph\(_{n-2}\)) to be measured. In the case of a mismatch (dashed line labelled as ‘no’), the phase pattern appeared for the first time. GPHT predicts the current phase Ph\(_t\) as a prediction result, because there is no prediction information. So GPHR is added into GPHT, and Pht is stored in Pred-n. If all the entries are occupied, LRU is used to remove the oldest entry.

Fig. 2 describes the two types of GPHT limitations when the same phase is repeated: PHT pollution and prediction ambiguity. First, GPHT pollution (Fig. 2a) occurs when PHT is occupied with the same phases. In Fig. 2a, since phase 3 is repeated 4 times, 4 tags are used to cover the repetition while the prediction value is the same. We call this PHT pollution because GPHT cannot hold other phase patterns of different prediction values, which reduces the accuracy of prediction. In this case, the numbers of if the GPHT knows that the same phase will repeat, prediction can be done without PHT pollution. Secondly, prediction ambiguity (Fig. 2b) occurs when a PHT Tag can have more than two prediction values. For example, in Fig. 2, the last PHT tag (3,3,3,3) has two prediction values, 3 (in Fig. 2a) and 5 (in Fig. 2b). Owing to the fixed size of GPHT depth, GPHT cannot capture the whole repetition of phase. So later when phase 3 is repeated, GPHT predicts 5 as the next phase, which is in accurate. In summary, PHT pollution and prediction ambiguity reduce the prediction accuracy when the same phase is repeated.

GPHTex with run-length encoding: To handle phase repetition, we present GPHTex that captures phase repetition by using RLE. RLE compresses a sequence of the same characters into two tuples-symbol (PHASE) and repetition count (RUN_LENGTH)-and thus, RLE helps the GPHTex to not fill up the history table. However, RLE consumes the memory twice more than the original GPHT due to the two-tuple encoding. To reduce the memory usage,
GPHTex introduces a new encoding scheme. The new encoding scheme is based on the fact that the phase has the maximum phase (MAX_PHASE) because a microprocessor has a finite number of scalable frequencies. This fact implies that the symbol and repetition count are encoded into a single tuple. Equation (2) describes how the new encoding scheme of GPHTex combines PHASE and RUN_LENGTH into a single tuple.

$$\text{PhaseEnc} = \text{phase} + (\text{MAX}_\text{PHASE} \times (\text{RUN}_\text{LENGTH} - 1))$$

Fig. 3 describes how GPHTex encodes phase repetition. The numbers in the left side present phase pattern, and the circle represents the currently measured phase. In Fig. 3a, GPHTex records phase 5 into GPHR and counts phase repetition in variable RL. $\{RL = 1\}$ means that phase 5 appears for the first time, and thus, the run length of the current phase is 1. Then, phase 3 is measured, and GPHTex records phase 3 in GPHR (Fig. 3b). At this time, the RL value is reset to 1 because 3 is the newly measured phase. Fig. 3c depicts how GPHTex predicts the value 3. When phase 5 is measured after the repetition of 3 six times, as shown in Fig. 3d, GPHTex updates GPHR with the encoded phase value 53 by using (2), and records the newly measured phase 5. The subsequent phase 4 is measured and encoded in GPHR, as shown in Fig. 3e. With the new encoding algorithm, GPHTex can capture not only phase repetition, but also phase variations as the number of GPHR depth.

$$\text{GPHTex configuration} = 4 \_ 64 \_ 128 \_ 1024$$

From our evaluation, we chose $4, 6, 8,$ and $12$ for the size of GPHR and $1, 64, 128, 1024,$ CPU 2006 benchmarks that consist of 29 applications. The measured phase. In Fig. 3 the left side present phase pattern, and the circle represents the currently measured phase. In Fig. 3a, GPHTex records phase 5 into GPHR and counts phase repetition in variable RL. $\{RL = 1\}$ means that phase 5 appears for the first time, and thus, the run length of the current phase is 1. Then, phase 3 is measured, and GPHTex records phase 3 in GPHR (Fig. 3b). At this time, the RL value is reset to 1 because 3 is the newly measured phase. Fig. 3c depicts how GPHTex counts the phase repetition in variable RL, and GPHR holds the repeating phase value 3. When phase 5 is measured after the repetition of 3 six times, as shown in Fig. 3d, GPHTex updates GPHR with the encoded phase value 53 by using (2), and records the newly measured phase 5. The subsequent phase 4 is measured and recorded in GPHR, as shown in Fig. 3e. With the new encoding algorithm, GPHTex can capture not only phase repetition, but also phase variations as the number of GPHR depth.

Fig. 3 New encoding of GPHTex ($\text{MAX}_\text{PHASE}=10$)

a) Initial phase 5 is measured and updated to GPHR
b) New phase 3 is updated and variable RL remains to 1
c) If phase starts to repeat, variable RL just increase
d) At the end of repetition, variable RL resets to 1 and encoded phase value (53) is applied to GPHR
e) New phase 4 is updated and variable RL remains to 1

Experimental result: To evaluate GPHTex performance, first, we explore the optimal configuration of GPHTex that consist of GPHR depth and the number of PHT entries. Even though both GPHT and GPHTex predict next phase based on the history table, finding the new optimal configuration of GPHTex is necessary, because GPHTex compresses repeating workload using RLE, and GPHT does not. Fig. 4 shows the average prediction accuracy of GPHTex along with the possible configurations. To come up with possible configurations, we use $4, 6, 8,$ and $12$ for the size of GPHR and $1, 64, 128, 1024,$ and $4096,$ and $8192$ for the number of PHT entries, which leads to 24 configurations in accordance with [1]. For 24 configurations, we run SPEC CPU 2006 benchmarks that consist of 29 applications. The $X$-axis in Fig. 4 represents the possible configurations of the depth and the number of entries. For example, if the $X$-axis value is 4.64, the configuration of GPHTex is 4 GPHR depth and 64 PHT entries. The $Y$-axis represents the prediction accuracy.

From our evaluation, we chose $4, 128$ as the optimal configuration for GPHTex, although $6, 1024$ has the highest prediction accuracy. The reason is that in the case of $6, 1024$, the cost is higher than the prediction improvement. In other words, $6, 1024$ requires more memory and matching time than the $4, 128$ configuration, but improves the accuracy by only about 1%. For a similar reason, GPHT employs the $8, 128$ configuration as the optimal configuration [1]. On average, we find that GPHTex predicts with 69% accuracy, whereas GPHT predicts 61%.

Fig. 5 shows the phase prediction results of SPEC CPU 2006 benchmarks. In case of Soplex benchmark, GPHTex improves 14% prediction accuracy, however, GPHTex shows 0.03% reduced prediction accuracy on Gromacs benchmark, which is already predicted as 90% in GPHT.

Fig. 5 Prediction accuracy of both predictors

To measure the energy saving of GPHTex, we integrated it with the Linux 3.14.1 kernel. In the experiment, we use Intel i7 sandybridge processor that accumulates the energy consumed by the running core. For comparison, we measure the energy consumption of the CPUFreq on demand policy [4], which is the default power management policy of Linux. Fig. 6 shows the results of the normalised energy delay product (EDP) of GPHT, GPHTex, and Linux. Linux EDP is set to 100% to compare the EDP gains of GPHT and GPHTex. The results show that GPHT shows 7.31% less EDP than Linux, GPHTex shows 11.56% less EDP than Linux. These results show that GPHTex’s enhanced prediction also affects the frequency change, which improves 4% of energy saving over GPHT.

Fig. 6 Normalised improved EDP compared with GPHT and Linux

Conclusion: This Letter identifies the limitation of GPHT on workload repetition and proposes GPHTex that enhances GPHT with RLE. When both predictors are integrated with the Linux kernel, GPHTex shows more accurate workload prediction than GPHT. In addition to prediction accuracy, GPHTex achieves more energy saving than GPHT and the Linux CPUFreq on demand policy.

Acknowledgement: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (no. 2010-0029180).

© Crown Copyright 2015
Submitted: 20 January 2015
doi: 10.1049/el.2014.4529
S.W. Kim, T.M. Kim and C. Yoo (College of Informatics, Korea University, Seoul 136-713, Republic of Korea)
✉ E-mail: chuckyoo@os.korea.ac.kr

References