2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications

Fast and Accurate On-line Prediction of Performance and Power Consumption in Multicore-based Systems

Young-Ho Lee
Department of Computer Science and Engineering
Seoul National University
Seoul, Korea
buriburi205@davinci.snu.ac.kr

Jihong Kim
Department of Computer Science and Engineering
Seoul National University
Seoul, Korea
jihong@davinci.snu.ac.kr

***Abstract*—Although multi-core processors have emerged as a dominant low-power architectural solution in high performance processor design, it is still challenging to take a full advantage of the high power efficiency of multi-core processors. One such challenge occurs when an operating system tries to assign a multi-threaded application to a target multi-core processor in an energy efficient fashion. With an increasing number of cores combined with sophisticated power management schemes, it becomes more difficult to decide the most appropriate runtime configuration for a given application so that the overall energy efficiency is maximized. In this paper, we propose a novel performance and power estimation technique, called PET, for multi-core systems. The PET scheme is based on a compact but accurate performance and power transformation model, which aims to predict the performance and power consumption of a large number of runtime configurations using hardware performance counters collected in a small number of representative runtime configurations. Using a transformation model, PET enables to accurately determine the best runtime configuration of multi-threaded applications at runtime with a small overhead over an existing naive solution. Experimental results on an Intel Q6600 quad-core processor show that PET can accurately predict the performance and power consumption of multi-threaded applications running on 1-4 cores under two different frequency levels with an average prediction error of 2.1%-8.3% and 3.2%-6.5% over the measured data, respectively. We also show that PET is effective in estimating the performance and power consumption of two co-running applications with an average prediction error of less than 5%.**

***Keywords*-multi-core, performance estimation, power estimation, multi-threaded**

I. Introduction

As single-core processors rapidly reach the physical limits of possible design complexity and clock frequency, multi-core processors have emerged as a dominant low-power architectural solution in various computing systems ranging from handheld devices to high-end server systems. The trend towards multi-core design is motivated by the potential benefits of multiple power-efficient cores where each core may potentially sacrifice single-threaded performance to improve overall throughput and energy efficiency. As a result, applications can achieve higher performance at a reduced energy consumption by effectively exploiting thread-level parallelism (TLP). However, there are still many challenges that need to be addressed to take a full advantage of the high power efficiency of multi-core processors. One such challenge occurs when an operating system tries to assign a multi-threaded application to a target multi-core processor in an energy efficient fashion. For example, the optimal number of cores for a multi-threaded application can be significantly affected by the performance and power characteristics depending on a degree of TLP. In addition, when running a mixed workload that consists of multiple multi-threaded applications, the optimal number of cores for each application may be quite different from that under an isolated execution of each application because they may compete for the same number of cores. Without understanding each application's energy efficiency under a varying number of cores, however, it is very difficult for an operating system to determine the best runtime configurations of multiple applications so that the overall energy efficiency is maximized.

Unfortunately, an increasing design complexity of multi-core processors makes an operating system to apply an optimal decision even harder. First, as industry trends show that the number of cores will continuously increase with process generations, multi-core design continues to integrate more and more cores on a single chip. For example, processor manufacturers already have been developing high performance multi-cores such as Intel's 48-core SCC (Single-Chip Cloud Computer) [4] and Tilera's 64-core TilePro64 [12]. Second, modern multi-core processors employ sophisticated dynamic power management techniques such as clock gating, power gating, and DVFS, to reduce the processor power consumption at various design granularities. For example, Intel's Turbo Boost technology [9] implements an aggressive hardware-based power gating for idle cores and DVFS for active cores. As another example, Samsung's Exynos 4 Quad processor [10] provides 13 DVFS levels (i.e., 200 MHz to 1.4 GHz) to support fine-grained power management. With an increasing number of cores combined with sophisticated power management schemes, it becomes more difficult to decide the most appropriate runtime configuration for a given application in both the performance and power perspectives.

A naive approach to determine the best runtime configuration would be to use a special training period during which all possible runtime configurations are evaluated in advance. The results collected from the training period can be used to find the best runtime configuration for the rest of execution times. However, such an approach would not practical for most modern multi-core processors because the overall exploration space is potentially huge for an exhaustive search. Furthermore, the number of required run-

978-0-7695-5022-0/13 $26.00 © 2013 IEEE
DOI 10.1109/TrustCom.2013.249

time configurations can increase even higher when considering various power-saving techniques with multiple power modes. To address this problem, previous approaches have focused on energy-aware scheduling in homogeneous [6], [8] and heterogeneous [5], [11] multi-core systems and many-core systems [13]. However, the existing techniques mainly focus on determining the appropriate DVFS level for running applications (i.e., threads). Especially, they do not adequately consider performance and power trade-off depending on the number of threads for multi-threaded applications. For example, previous approaches assume that the number of multiple threads for an application is fixed by a programmer or determined by a system software based on the number of available physical cores. In such a case, the potential energy savings by applying DVFS can be limited because even an inefficiently parallelized multi-threaded application may unnecessarily waste on-chip power. Moreover, such ad-hoc decisions based on partial runtime information can lead to a sub-optimal energy efficiency of the overall system when running a mixed workload.

In this paper, we propose a novel performance and power estimation technique, called PET, for multi-core systems. The PET scheme is based on a compact but accurate performance and power transformation model that aims to accurately predict the performance and power consumption of a large number of different runtime configurations. In the PET scheme, a multi-core processor is modeled as a set of *virtual power states* (VPS) where each VPS refers to a different runtime configuration (e.g., a degree of TLP and the DVFS levels). In the offline model building stage, we construct both a performance and power model for each VPS and a transformation model to predict the performance and power among different VPS's using hardware performance counters. Then, at runtime, PET enables to accurately determine the best runtime configuration of multi-threaded applications with a significantly reduced runtime overhead over an existing naive solution.

In order to validate the effectiveness of the PET model, we evaluated the proposed approach using a large number of multi-threaded applications on an Intel Q6600 quad-core processor. The experimental results show that PET can accurately estimate the performance and power consumption of multi-threaded applications running on 1-4 cores under two DVFS levels with an average prediction error of 2.1%-8.3% and 3.2%-6.5%, respectively, depending on a target VPS. The results also show that PET is able to predict an overall performance and power consumption of two co-running applications with an average error of less than 5%.

The remainder of this paper is organized as follows. In Section II, we explain the main motivation of our work. Section III presents the proposed methodology in detail. Section IV demonstrates our experimental results. Section V concludes with a summary and directions for future work.

II. Motivation

Energy-efficient use of multi-core processors remarkably depends on how an operating system can properly assign the running applications to their most appropriate runtime configurations so that a target system can achieve the maximum energy efficiency. Because a primary design goal of multi-core processors is to provide effective parallel performance in a power efficient fashion, multi-core processors are likely to host both single-threaded and multi-threaded applications. Therefore, an effective workload assignment policy for multi-core processors is required to determine not only how many cores should be assigned, but also which power optimization technique (i.e., the DVFS level) should be applied to each application. However, most existing techniques have focused on how to decide the best DVFS level of each core (or a processor) based on the power characteristics of running applications. In particular, they do not consider multi-threaded applications to achieve potential benefits of energy efficiency by adequately exploiting TLP.

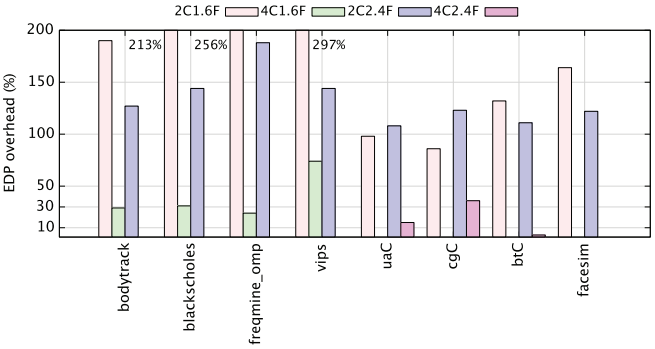


Figure 1: EDP overhead of multi-threaded applications among different runtime configurations

In order to understand how multi-threaded applications exhibit different energy efficiency depending on a degree of TLP as well as the DVFS levels, Figure 1 shows an energy-delay product (EDP) overhead of multi-threaded applications from PARSEC and NPB benchmark suite among different runtime configurations on an Intel Q6600 processor (a detailed experimental setup is described in Section IV-A). Given a workload, we define an EDP overhead as $(edp_i - edp_{best})/edp_{best}$, where edp_i is the EDP of the i-th runtime configuration and edp_{best} represents the minimum EDP overhead among all possible runtime configurations. In our evaluations, we use six different runtime configurations by changing the number of cores and the DVFS level. We denote a specific runtime configuration by *x*C*y*F where a processor is configured to use *x* cores running at *y* clock frequency. In Figure 1, we did not include the results for 1C1.6F and 1C2.4F because these two runtime configurations incurred a very large EDF overhead ranging between 370% and 2481%, which imply that assigning too small number of cores to multi-threaded applications leads to poor energy efficiency, regardless of the DVFS level.

The figure clearly shows that there are significant variations in the EDP overhead among different runtime configurations. For uaC, the EDP overhead varies between 15% and 108%. Even worse, vips exhibits the EDP overhead of between 74% and 297%. The larger variations in the EDP overhead indicates that making an incorrect decision in

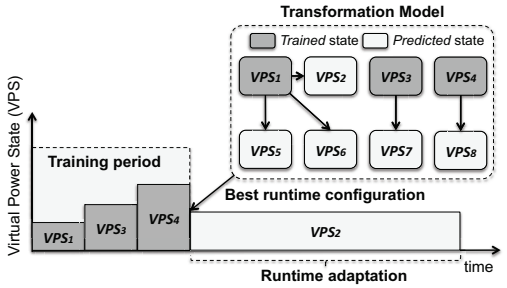


Figure 2: Overview of the proposed performance and power estimation technique using transformation models

workload assignments can lead to a significant degradation of energy efficiency. For example, executing vips whose best runtime configuration is 4C2.4F at 2C1.6F would occur the EDP overhead of 297%.

The figure also shows that the EDP overhead is more sensitive to a degree of TLP rather than the DVFS levels. At 1.6 GHz, the applications show the maximum EDP overhead by up to 297%, and 74% on two, and four cores, respectively. We find the similar results at 2.4 GHz: the maximum EDP overhead by up to 188% on two cores, and 36% on four cores. Moreover, even though all workloads yield the minimum EDP overhead when running on four cores, they can still incur non-negligible EDP overhead depending on the DVFS levels. Some benchmarks (e.g., uaC, and cgC) exhibit the minimum EDP overhead at 4C1.6F, while other workloads (e.g., blackscholes and bodytrack) are preferred to run at 4C2.4F. In addition, some workloads (e.g., btC and facesim) show a similar EDP overhead at both DVFS levels. The results strongly indicate that an energy-efficient workload assignment policy must take into account of both the DVFS levels and a degree of TLP in a unified manner. However, most existing techniques on workload assignment for multi-core processors have focused on how to determine the best DVFS levels for given applications. Despite significant variations in energy efficiency depending on a degree of TLP, they do not carefully determine a degree of TLP for multi-threaded applications, assuming that the number of threads for multi-threaded applications is typically fixed by a programmer or determined by a system software based on the number of available physical cores. As shown in Figure 1, we argue that such policies might lead to a sub-optimal energy efficiency of the overall system.

In order to decide the best runtime configuration, one possible approach is to apply a training period to exercise all possible runtime configurations. While such a naive approach would be appropriate for single-threaded applications where a target processor provides a limited number of DVFS levels, it might occur too much energy overhead for multi-threaded applications because the overall exploration space is potentially huge for exhaustive search. Moreover, these overheads increase with the number of cores and DVFS levels. For example, assuming that applications in Figure 1 are trained in a naive approach to decide the best runtime configuration, the total EDP overhead for each application can be defined as $(\sum_{i=1}^{n} edp_i - edp_{best})/edp_{best}$, where n is the number of possible runtime configurations. In such a case, we observe that the total EDP overhead spikes by up to 815% for vips and 639% on average, making a naive approach to be infeasible to be implemented. To overcome these drawbacks, the proposed PET scheme aims to estimate the power and performance among different runtime configurations so that applications can be assigned to their most appropriate runtime configurations at online with a significantly reduced runtime overhead over an existing naive solution.

III. Methodology

A. Overview of PET

In the PET scheme, a multi-core processor is modeled as having a set S_{VPS} of N virtual power states (VPS), denoted by $S_{VPS} = \{vps_1, vps_2, ..., vps_N\}$, where vps_i corresponds to the i-th runtime configuration (e.g., different DVFS level and a degree of TLP). Each VPS vps_i in S_{VPS} is classified as a trained state or a predicted state. A trained state vps_t refers to a VPS where the performance and power consumption are obtained using the measured hardware performance counters (HPCs) during the training period. If the performance and power consumption of a VPS vps_p can be predicted using the measured HPC data at trained states, we define vps_p as a predicted state. We define S_T and S_P as a set of trained states and predicted states, respectively. We also define $perf_i$ and $power_i$ as performance and power metrics at vps_i, respectively. For a given application, PET estimates the performance and power consumption of the application in predicted states based on the performance and power data collected in trained states.

In order to accomplish this goal, PET constructs two models during an offline design stage. First, we construct a performance and a power model for each VPS in S_{VPS} using HPCs which are generally available for modern microprocessors. For each VPS vps_i, we use MIPS (million instructions per second) to model $perf_i$. To derive $power_i$, we apply a simple linear regression model using HPCs (as described in Section III-B). Second, we derive a transformation model which is responsible for predicting the performance and power consumption of each predicted VPS vps_p in S_P (as described in Section III-C). Using a transformation model, PET can accurately estimate the performance and power consumption in predicted states during runtime. Since a small number of trained states can cover most of predicted states, the best runtime configuration can be decided with a significantly reduced cost over a naive approach.

Figure 2 illustrates an overall procedure of PET using an example to determine the best runtime configuration for a multi-threaded application. In this example, we consider a target processor with eight VPS's (e.g., four cores with two DVFS levels at chip-level granularity). In a naive approach, an application should exercise at eight VPS's during a training period to find the best runtime configuration (i.e., $S_T = S_{VPS}$). On the other hand, PET allows to train the workloads only at three trained VPS's because the

performance and power consumption in predicted states can be estimated using a transformation model. For instance, the measured power and performance at vps_1 can be used to predict the estimated ones at vps_2, vps_5, and vps_6.

B. Power Modeling of Multi-Core Processors

Most previous work on power modeling have shown that processor power dissipation can be accurately estimated by applying a simple linear regression model [1], [2]. The rationale behind this is twofold. First, specific HPCs, such as the number of instructions, L2 misses, and resource stalls, have a strong correlation with the actual processor power consumption. Second, processors exhibit synchronous power behavior, that is, it is proportional to the computational load (i.e., utilization) on the processor. Therefore, the instantaneous processor power consumption can be directly estimated by using current utilization of those HPCs.

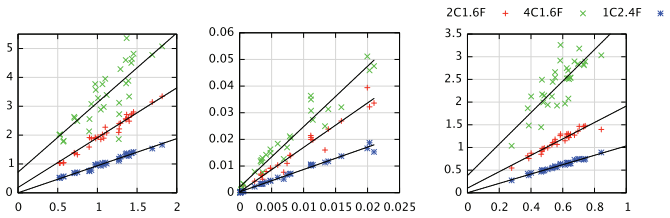
While linear regression models have been widely used to predict the power consumption of single-core processors because of the simplicity and reasonable modeling accuracy, the advent of multi-core processors poses new challenges for deriving an accurate power model. The key difficulty is that the power consumption of modern multi-core processors is not always proportional to the computational load on the processor because recent multi-core processors have an increasing number of hidden power states that are not directly exposed to an operating system due to sophisticated built-in DPM policies, processor variations, and the operating environment (i.e., temperature) [1]. For these reasons, multi-core processors are likely to exhibit a different proportionality between the measured HPCs and actual power consumption, implying that a single power model is no longer appropriate for multi-core processors. As a result, the existing techniques based on a single power model would yield reasonable modeling accuracy under specific runtime configuration of a target multi-core processor. In order to rectify this problem, we use multiple power models to accurately estimate the power consumption of multi-core processors.

Let $S_E = \{e_1, e_2, ..., e_n\}$ be a set of n HPC events used in a power model. For j-th HPC event e_j in S_E, we define ${r_i}^{e_j}$ as a measured value of e_j per cycle at vps_i. Then, $power_i$ can be generally modeled by the following equation:

$$power_i = c_{i,0} + \sum_{j=1}^{n} c_{i,j} \cdot f_j({r_i}^{e_j}) + \delta_i \qquad (1)$$

where $C_i = \{c_{i,0}, c_{i,1}, ..., c_{i,n}\}$ is a set of the model coefficients, δ_i is an error term, and $f_j()$ denotes a function form of e_j depending on the relationship between e_j and $power_i$ (i.e., linear, logarithmic, or exponential).

In the PET scheme, a power model is constructed for each VPS during an offline design stage. A large amount of training data are collected while running a wide range of training benchmarks on a target processor. The model coefficients are then determined by correlating HPCs with measured power consumption to minimize the modeling error (i.e., δ_i). Once a power model is generated, it can be used online to predict power consumption of running applications without real power measurements.



(a) Instruction (b) L2 miss (c) Stall

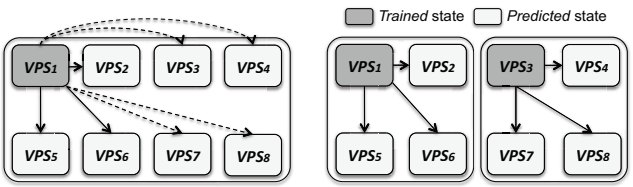
Figure 3: The correlation of HPCs among different VPS

C. Building a Transformation Model

The key component of the PET scheme is a transformation model that enables to accurately estimate the performance and power consumption at predicted states using the measured HPC data at trained states. As shown in Figure 2, a transformation model is represented as a graph where each node corresponds to an individual VPS and an edge, $vps_t \rightarrow vps_p$, from a trained state vps_t to a predicted state vps_p represents that the performance and power consumption of a given workload at vps_p can be estimated using the measured HPC data at vps_t. In this paper, we also call vps_p is *transformable* from vps_t for each edge, $vps_t \rightarrow vps_p$.

The key observation in building a transformation model is that the HPC data at vps_p can be accurately estimated using the measured data at vps_t because of a strong linear relationship of HPCs between vps_t and vps_p. To correlate whether HPCs show a linear relationship among different VPS's, Figure 3 plots the measured HPCs of multi-threaded applcations on an Intel Q6600 processor. In our evaluation, we use the following three HPCs in our power model: the number of instructions (*inst*), the number of L2 misses (*miss*), the number of resource stalls (*stall*). We assume that 1C1.6F is a trained state and the other VPS's are predicted states; the measured HPCs at 1C1.6F are plotted on the x-axis and those at 2C1.6F, 4C1.6F, and 1C2.4F are plotted on the y-axis. The figure clearly shows that HPCs have a strong linear relationship at 2C1.6F and 1C2.4F with the correlation factor of close to 1. High correlation factors indicate that the HPC data at predicted states (i.e., 2C1.6F and 1C2.4F) can be accurately estimated using the measured HPC data at a trained state (i.e., 1C1.6F). On the other hand, there is a relatively weak linear relationship between 1C1.6F and 4C1.6F. For this combination, *inst*, *miss*, and *stall* show the correlation factor of 0.82, 0.98, and 0.79, respectively. The results imply that 1C1.6F is not a good trained state for 4C1.6F, that is, 4C1.6F should be transformed from another trained state. (If such a trained state does not exist, 4C1.6F should be set to a trained state.)

Based on the above observation, the performance and power consumption at vps_p can be accurately estimated by exploiting a linear relationship of HPCs between vps_t and vps_p. Specifically, we first derive a transformation model for individual HPCs. Given a HPC event e, the HPC data at vps_p can be predicted using the measured data at vps_t, by



(a) Baseline model (b) Splitted model

Figure 4: An example of building a transformation model

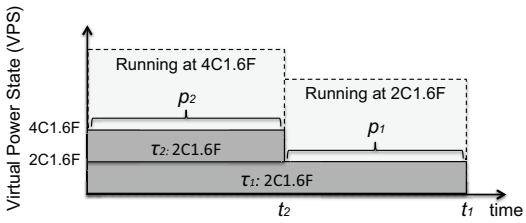


Figure 5: An example of VPS changes when running two multi-threaded applications

the following equation:

$$r_p{}^e = c_0 + c_1 \cdot r_t{}^e \ \ (+ c_2 \cdot r_t{}^{e'}) \tag{2}$$

where c_0, c_1, and c_2 are the model coefficients, and $r_p{}^e$ and $r_t{}^e$ are the HPC data per cycle at a predicted state and a trained state, respectively. Note that a term $c_2 \cdot r_t{}^{e'}$ is optionally included in Equation (2) because some HPCs such as e exhibit higher modeling accuracy when other HPCs such as e' are used together. For example, we observe that L2 miss is also a good indicator in deriving a transformation model for instruction (as described in Section IV-A). By applying Equation (2) to all HPCs used in a performance and power model, the performance and power consumption at vps_p can be predicted using the HPC data at vps_t.

Another key problem in building a transformation model is to determine trained states and predicted states. Given a multi-core processor where a set S_{VPS}=$\{vps_1, vps_2, ..., vps_N\}$ of N VPS's, we begin to build a baseline model where vps_1 is the only trained state. Next, we perform a correlation analysis to verify the baseline model if other VPS's are transformable from vps_1. Let $cor(vps_t, vps_p)$ be the minimum correlation factor of all HPCs between vps_t and vps_p. Because the larger value of $cor(vps_t, vps_p)$ means that all HPCs show the higher linear relationship between vps_t and vps_p, we define that vps_p is transformable from vps_t if $cor(vps_t, vps_p)$ is larger than a predefined threshold θ (i.e., $0 \leq \theta \leq 1$). For example in Figure 3, when θ=0.9, both 2C1.6F and 1C2.4F are transformable from 1C1.6F, while 4C1.6F is not transformable from 1C1.6F. If there exists some predicted states which is not transformable from any trained state, we split a baseline model into two submodels where those predicted states form a new submodel. In the submodel, we perform the above steps repeatedly until all predicted states are transformable from their corresponding trained states.

Figure 4 illustrates an overall procedure of building a transformation model using an example where a target multi-core processor consists of eight VPS's. In the figure, we assume that a dashed arrow between vps_t and vps_p denote that vps_p is not transformable from vps_t, that is, a correlation factor between vps_t and vps_t is below θ. As shown in Figure 4(a), we first build a baseline model and verify the model using a correlation analysis. The results indicate that four VPS's (i.e., vps_3, vps_4, vps_7, and vps_8) are not transformable from vps_1. In such a case, we split a baseline model where those four VPS's form the second submodel, as shown in Figure 4(b). In a splitted model, vps_3 is set to a trained state, assuming that the other three VPS's are transformable from vps_3. We again perform a correlation analysis in the splitted model. The results indicate that all VPS's in the submodel are transformable from vps_3. Consequently, the resulting transformation model consists of two submodels where $S_T = \{vps_1, vps_3\}$.

D. Extending PET for Mixed Workloads

In the previous section, we have demonstrated that a transformation model can be effective in deciding the best runtime configuration for *individual* applications. In addition to such individual applications (i.e., running in isolation), multi-core processors are likely to concurrently host a mixed workload that consists of multiple single-threaded and multi-threaded applications to fully utilize the available cores. In order to handle such realistic workloads, we extend the PET scheme to estimate the overall performance and power consumption of a mixed workload.

When multiple applications are running concurrently, the overall performance and power consumption depends on the runtime configurations of individual applications in a mixed workload. To determine the optimal runtime configuration for each application in a mixed workload, one possible approach is to examine all possible combinations of runtime configurations in advance. However, such an approach is impractical due to a huge exploration space that substantially increases with not only the possible runtime configurations of a target processor, but also the number of applications. While another approach is to build a transformation model for a mixed workload, such an approach is not still feasible because it is almost impossible to consider all possible combinations of different runtime configurations for a mixed workload. For example, consider N co-running applications where the number of possible runtime configurations for each application is M. Then, the resultant transformation model would include M^N nodes at maximum, which is infeasible to be used at online and even generated at the offline design stage.

In order to address this issue, we derive an aggregation function, denoted by $\oplus$, to estimate the performance and power consumption for a mixed workload, instead of building a huge transformation model. Because the proposed transformation model can accurately estimate the performance and power consumption of individual applications, the overall performance and power consumption of a mixed workload can also be accurately predicted using

an aggregation function. The key observation in deriving an aggregation function is that a mixed workload exhibits the VPS changes depending on the execution phase of individual applications in the workload. Figure 5 shows an example of how the overall performance and power consumption can be accurately estimated when two multi-threaded applications, τ_1 and τ_2, are running concurrently. In this example, we assume that both τ_1 and τ_2 run at 2C1.6F with an isolated execution time of t_1 and t_2, respectively. The figure shows that an overall VPS changes depending on the execution phase in which τ_1 and τ_2 are concurrently running. Although each application is executed at 2C1.6F, an overall VPS is 4C1.6F until t_2 because four cores are occupied by both applications. After τ_2 finishes its execution, however, it again changes to 2C1.6F until t_1 because two cores are occupied only by τ_1. For this reason, to accurately predict the overall performance and power consumption, it should be taken into an account of how an overall VPS changes depending on the execution phase of τ_1 and τ_2.

Based on the above observation, we derive the overall performance and power consumption of two co-running applications, τ_1 and τ_2. We assume τ_1 and τ_2 is executed at vps_i and vps_j, respectively. We define vps_k as the overall VPS at which τ_1 and τ_2 are executed simultaneously. Then, the overall performance of τ_1 and τ_2, denoted by $perf(\tau_1 \oplus \tau_2)$, can be expressed by the following equation:

$$perf(\tau_1 \oplus \tau_2) = f(perf_k) \cdot p_2 + f(perf_i) \cdot p_1 \quad (3)$$

where $p_1 = (t_1 - t_2)/t_1$, $p_2 = t_2/t_1$, $perf_k = perf_i + perf_j$. We also introduce a performance calibration function $f()$ to adjust the overall performance at vps_k due to resource contention. For example, we empirically identify $f(perf_k)$ for an Intel Q6600 processor by the following equation:

$$f(perf_k) = 0.88508 * perf_k - log({r_k}^{miss}) * 0.03941 \quad (4)$$

where ${r_k}^{miss}$ is the aggregated number of L2 misses at vps_k (i.e., ${r_i}^{miss} + {r_j}^{miss}$). Likewise, we model the overall power consumption, $power(\tau_1 \oplus \tau_2)$, as follows:

$$power(\tau_1 \oplus \tau_2) = power_k \cdot p_2 + power_i \cdot p_1 \quad (5)$$

where $power_k$ is an aggregated power consumption at vps_k. Based on Equation (1), the overall power consumption, $power_k$, can be modeled by the following equation:

$$power_k = c_{k,0} + \sum_{j=1}^{n} c_{k,j} \cdot f_j({r_k}^{e_j}) \quad (6)$$

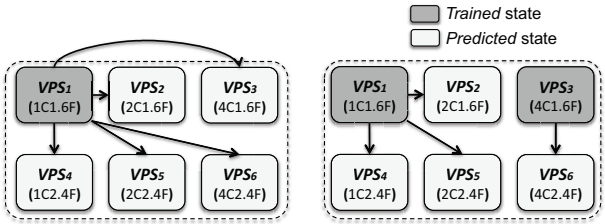
where ${r_k}^{e_j}$ is the aggregated HPC data e_j at vps_k.

Based on equation (3) and (5), the overall performance and power consumption of N co-running applications can be generalized as $perf(\tau_1 \oplus \tau_2 \oplus \tau_3 \oplus ...)$ and $power(\tau_1 \oplus \tau_2 \oplus \tau_3 \oplus ...)$, respectively.

IV. Experiments

A. Experimental Setup

Our testbed is an Intel Q6600 quad-core processor with 6 MB on-chip L2 cache running Linux 2.6.31. The processor supports per-chip DVFS with two frequency levels: 1.6 GHz and 2.4 GHz. Based on the target configuration, we consider six VPS's from 1C1.6F to 4C2.4F. In our evaluation, we ignore some VPS's such as 3C1.6F and 3C2.4F because multi-threaded applications typically run with 2^n threads. In order to measure processor power consumption, we use the approach proposed in [2]. An agilent 34410A digital multimeter (DMM) is used together with a Fluke i410 current probe to collect the current running through the 12V power lines from the processor. The actual power consumed would be the measured current multiplied by 12V, where the current is read through a USB cable using a USBTMC device driver in Linux kernel. We also collect HPCs using the Linux perf infrastructure and the perfmon2 library [3]. The measurement period of power consumption and HPCs are set to 20 ms and 100 ms, respectively. For a regression analysis, we use the R statistical software. In our evaluation, we use 14 multi-threaded benchmarks from the PARSEC and NAS Parallel Benchmark (NPB) combined with various input data sets, totaling 33 different benchmark combinations.



(a) Baseline model (b) Splitted model

Figure 6: Transformation models for Intel Q6600 processor

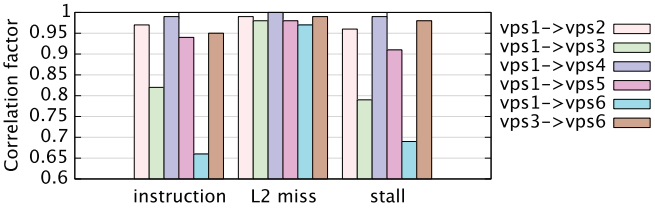
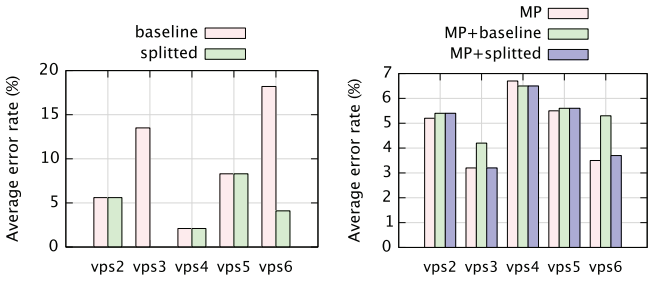


Figure 7: A correlation analysis for Intel Q6600 processor

In order to build a power model, we first classify the available HPCs of our target processor into three different categories covering the major microarchitectural components that heavily influence processor power consumption: instruction, memory, and stall. We then choose the top-ranked HPC from each category which has the largest correlation with actual power consumption among all HPCs in the same category. In particular, we use the following three HPCs in our power model: the number of instructions (*inst*), the number of L2 misses (*miss*), the number of resource stalls (*stall*). Based on Equation (1), we derive a power model for an Intel Q6600 processor by the following equation:

$$power_i = c_{i,0} + c_{i,1} \cdot {r_i}^{inst} + c_{i,2} \cdot log({r_i}^{miss}) + c_{i,3} \cdot {r_i}^{stall} \quad (7)$$

In order to generate a transformation model, we begin



(a) Performance (b) Power

Figure 8: Overall transformation accuracy

to build a baseline model where S_T=$\{vps_1\}$, as shown in 6(a). Figure 7 shows a correlation analysis for an Intel Q6600 processor with θ=0.9. The first five bars show the results for a baseline model. We observe that vps_2, vps_4, and vps_5 are transformable from vps_1 with the minimum correlation factor of 0.96, 0.99, and 0.91, respectively. On the other hand, vps_3 and vps_6 are not transformable from vps_1 because the minimum correlation factor at those VPS's are 0.79 and 0.69, respectively. As depicted in the sixth bar $vps_3 \rightarrow vps_6$, such a weak linear relationship of $vps_1 \rightarrow vps_3$ and $vps_1 \rightarrow vps_6$ can be improved by exploiting a strong linear relationship between vps_3 and vps_6. Based on the correlation analysis, we divide a baseline model into two submodels where $S_T = \{vps_1, vps_3\}$. Figure 6(b) describes the detailed transitions between trained states and predicted states in a splitted model. For each transition between vps_t and vps_p, we derive a transformation model for individual HPCs based on Equation (2), by the following equations:

$$r_p{}^{inst} = c_0 + c_1 \cdot r_t{}^{inst} + c_2 \cdot r_t{}^{miss} \quad (8)$$

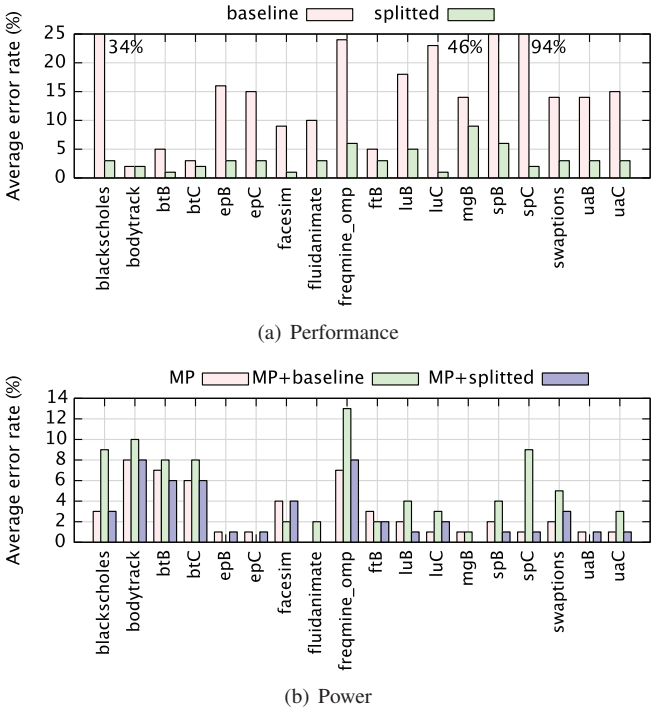
$$log(r_p{}^{miss}) = c_0 + c_1 \cdot log(r_t{}^{miss}) \quad (9)$$

$$r_p{}^{stall} = c_0 + c_1 \cdot r_t{}^{stall} + c_2 \cdot r_t{}^{miss} \quad (10)$$

B. Validation of Transformation Models

Figure 8(a) compares the average estimation errors of performance between a baseline and a splitted model. In the figure, the prediction error for a splitted model at vps_3 is 0% because vps_3 is a trained state in a splitted model. Both transformation models show a reasonable modeling accuracy with the prediction error of 5.6% at vps_2, 2.1% at vps_4, and 8.3% at vps_5. However, a baseline model exhibits a high degree of prediction errors of 13.5% at vps_3 and 18.2% at vps_6, because those VPS's are not transformable from vps_1. Such a high prediction error at vps_6 is dramatically improved under a splitted model with the estimation error of 4.1%.

Figure 8(b) shows the average prediction errors of power consumption between a baseline and a splitted model. In the figure, MP uses the measured HPC data at all VPS's, while MP+baseline and MP+splitted use the estimated HPC data at predicted states by applying a baseline and a splitted model, respectively. Hence, the estimation errors for MP indicate the modeling accuracy of the power model itself for each VPS (i.e., the lower bound of the estimation error



(a) Performance

(b) Power

Figure 9: Transformation accuracy between a baseline model and a splitted model at vps_6

under MP+baseline and MP+splitted). The estimation error under MP is between 3.2% and 6.7%. Under MP+baseline, the estimation error varies between 4.2% and 6.5%. The error is further reduced between 3.2% and 6.5% under MP+splitted. The results indicate that the prediction errors under MP+baseline and MP+splitted are within 2% and 1% of MP, respectively. In other words, MP+splitted is close to MP in terms of the modeling accuracy.

C. Effects of Splitting a Transformation Model

In this section, we further study how a splitted model can achieve a higher modeling accuracy than a baseline model. Figure 9(a) compares the prediction errors of performance across benchmarks under those two models at vps_6. A baseline model performs quietly poor with the prediction error of up to 94%. While some benchmarks are predicted well (e.g., bodytrack, isB, and uaA), a baseline model still exhibits significantly higher estimation errors than a splitted model for most of the benchmarks. We observe that a splitted model is almost 10 times more accurate on average than a baseline model. Under a baseline model, the prediction errors of blackscholes and spB are 34% and 46%, respectively. On the other hand, the errors of those workloads under a splitted model remarkably decrease by 3% and 6%, respectively. Moreover, for spC, a baseline model shows a remarkably large estimation error of 94%, while the error is just 2% under a splitted model.

Figure 9(b) shows the prediction errors of power consumption across benchmarks under those two models at

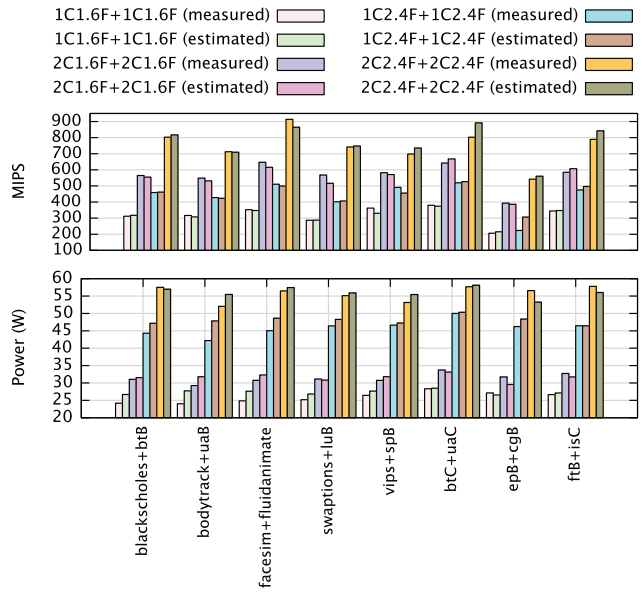


Figure 10: Performance and power estimation of two co-running applications

vps_6. We observe that the results of power consumption are similar to those of performance. MP+splitted is apparently more accurate than MP+baseline. While the prediction errors of the benchmarks are up to 13% under MP+baseline, the errors are 8% at maximum under MP+splitted. Specifically, an estimation error of freqmine_omp is 13% and 8% under MP+baseline and MP+splitted, respectively. In addition, a prediction error of spC is 9% under MP+baseline, while the error is 1% under MP+splitted. More importantly, we observe that the modeling errors of MP+splitted across benchmarks is within 2% of MP on average. The results imply that MP+splitted is close to MP in terms of the modeling accuracy.

D. Transformation Accuracy for Mixed Workloads

We also evaluate how PET can accurately estimate the performance and power consumption of a mixed workload. We consider two multi-threaded applications running concurrently on the target processor where each application is assigned one or two cores at 1.6 GHz or 2.4 GHz (i.e., 1C1.6F to 2C2.4F). For evaluation, we train each application at 1C1.6F to collect the HPC data and predict the performance and power consumption at three other VPS's using a transformation model. We then compute the overall performance and power consumption of a mixed workload at four possible combination of VPS's from 1C1.6F+1C1.6F to 2C2.4F+2C2.4F. Figure 10 compares the estimated performance and power consumption of two co-running applications, compared to the measured data. The figure shows that PET also enables to predict the overall performance and power consumption of mixed workloads with an average modeling error within 5% for both performance and power consumption.

V. Conclusions

In this paper, we have proposed a fast and accurate performance and power estimation technique, called PET, for multi-core processors. The novelty of the PET scheme is that a transformation model enables to accurately predict the performance and power consumption of multi-threaded applications with a significantly reduced runtime overhead by selectively training them on a small set of VPS's. In addition, we also present how the PET scheme can be extended to estimate the overall performance and power consumption of a mixed workload. While this paper has focused on applying PET for multi-core systems, PET can also be applied to more complicated systems such as homogeneous many-core systems and heterogeneous multi-core systems. Exploring these extensions is a part of our future work.

VI. Acknowledgments

This work was supported by NRF funded by MEST (No. 2012-0006417). The ICT at Seoul National University and IDEC provided research facilities for this study.

References

[1] J. C. McCullough., et al, "Evaluating the Effectiveness of Model-Based Power Characterization," In *Proc. of USENIX ATC*, 2011.

[2] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," In *Proc. of MICRO*, 2003.

[3] S. Eranian, "Perfmon2: A Flexible Performance Monitoring Interface for Linux," In *Proc. of Linux Symposium*, 2006.

[4] T. G. Mattson., et al, "The 48-core SCC Processor: The Programmer's View," In *Proc. of SC*, 2010.

[5] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez and J. Emer, "Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (PIE)," In *Proc. of ISCA*, 2012.

[6] K. Meng, R. Joseph, R. P. Dick and L. Shang, "Multi-Optimization Power Management for Chip Multiprocessors," In *Proc. of PACT*, 2008.

[7] M. A. Suleman, M. K. Qureshi and Y. N. Patt, "Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-threaded Workloads on CMPs," In *Proc. of ASPLOS*, 2008.

[8] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," In *Proc. of MICRO*, 2003.

[9] J. Charles, P. Jassi, A. N. S, A. Sadat and A. Fedorova, "Evaluation of the Intel Core i7 Turbo Boost feature," In *Proc. of IISWC*, 2009.

[10] Se-Hyun Yang., et al, "A 32nm High-K Metal Gate Application Processor with GHz Multi-Core CPU," In *Proc. of ISSCC*, 2012.

[11] D. Koufaty, D. Reddy and S. Hahn, "Bias Scheduling in Heterogeneous Multi-Core Architectures," In *Proc. of EuroSys*, 2010.

[12] S. Bell., et al, "TILE64 Processor: A 64-Core SoC with Mesh Interconnect," In *Proc. of ISSCC*, 2008.

[13] K. Ma, X. Li, M. Chen and X. Wang, "Scalable Power Control for Many-Core Architectures Running Multi-Threaded Applications," In *Proc. of ISCA*, 2011.