Spring 4-20-2015

Runtime Power-Aware Energy-Saving Scheme for Parallel Applications

Vaibhav Sundriyal
ISU, vaibhavs@iastate.edu

Masha Sosonkina
masha@scl.ameslab.gov

Runtime Power-Aware Energy-Saving Scheme for Parallel Applications

Abstract
Energy consumption has become a major design constraint in modern computing systems. With the advent of peta ops architectures, power efficient software stacks have become imperative for scalability. Modern processors provide techniques, such as dynamic voltage and frequency scaling (DVFS), to improve energy efficiency on-the-fly. Without careful application, however, DVFS and throttling may cause significant performance loss due to the system overhead. Typically, these techniques are used by constraining a priori the application performance loss, under which the energy savings are sought. This paper discusses potential drawbacks of such usage and proposes an energy-saving scheme that takes into account the instantaneous processor power consumption as presented by the running average power limit" (RAPL) technology from Intel. Thus, the need for the user to define a performance loss tolerance apriori is avoided. Experiments, performed on NAS benchmarks, show that the proposed scheme saves more energy than the approaches based on the pre-defined performance loss.

Keywords
DVFS, Energy Modeling, Intel RAPL

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–15

Runtime Power-Aware Energy-Saving Scheme for Parallel Applications

Vaibhav Sundriyal[1,*], Masha Sosonkina[2]

[1] *Ames Laboratory, Iowa State University, Ames, IA 50011.*
[2] *Department of Modeling, Simulation and Visualization Engineering, Old Dominion University*

SUMMARY

Energy consumption has become a major design constraint in modern computing systems. With the advent of petaflops architectures, power-efficient software stacks have become imperative for scalability. Modern processors provide techniques, such as dynamic voltage and frequency scaling (DVFS), to improve energy efficiency on-the-fly. Without careful application, however, DVFS and throttling may cause significant performance loss due to the system overhead. Typically, these techniques are used by constraining *a priori* the application performance loss, under which the energy savings are sought. This paper discusses potential drawbacks of such usage and proposes an energy-saving scheme that takes into account the instantaneous processor power consumption as presented by the "running average power limit" (RAPL) technology from Intel. Thus, the need for the user to predefine a performance loss tolerance is avoided. Experiments, performed on NAS benchmarks, show that the proposed scheme saves more energy than the approaches based on the predefined performance loss. Copyright © 0000 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Dynamic Voltage and Frequency Scaling (DVFS), Energy Modeling, Workload Modeling, Intel RAPL, NAS Parallel Benchmarks

1. INTRODUCTION

The last few decades have witnessed a tremendous rise in the design of scalable applications for various scientific domains. Their computational requirements force system engineers to develop ever more performance-efficient architectures. As a result, power consumption is rapidly becoming a critical design constraint in modern high-end computing systems. For example, according to an U.S. Department of Energy guidelines [20], to sustain an exaflops machine, its power consumption cannot go beyond ten-fold that of the current petaflops machines, meaning that for a 1000-fold increase in performance, the increase in power consumption may not accede ten-fold. Moreover, if the focus of the high-performance computing (HPC) community is only to maximize application performance, the computing system operating costs and failure rates can reach prohibitive levels. To address this challenge, power and energy optimizations are needed in modern computing platforms at all levels: application, system software, and hardware.

*Correspondence to: Ames Laboratory, Iowa State University, Ames, IA 50011. Email: vaibhavs@iastate.edu.

Contract/grant sponsor: 1) Ames Laboratory and Iowa State University of Science and Technology, 2) Air Force Office of Scientific Research 3) National Science Foundation; contract/grant number: 1) DE-AC02-07CH11358 with the U.S. Department of Energy, 2) AFOSR award FA9550-12-1-0476, 3) NSF/OCI – 0941434, 0904782, 1047772.

The current generation of Intel processors provides various P-states for dynamic voltage and frequency scaling (DVFS) and T-states for introducing processor idle cycles (throttling). For example, the Intel "Sandy Bridge" microarchitecture provides a total of fifteen P-states and eight T-states. The delay of switching from one state to another depends on the relative ordering of the current and desired states, as discussed, e.g., in [17]. The user may write a value to model-specific registers (MSRs) to change the P- or T-states of the processor. Recent Intel microarchitectures, such as Sandy Bridge, estimate power and energy consumption of the CPU and memory through the built-in MSRs, which certainly facilitate power-measurement efforts while providing several levels of the DVFS and throttling similar to the processor experimented with in this work.

1.1. Intel Running Average Power Limit (RAPL)

With the advent of Sandy Bridge family of processors, Intel has introduced capabilities for both onboard power meters and power clamping. The Intel Running Average Power Limit (RAPL) provides a standard interface for measuring and limiting processor and memory power by HW, OS, applications etc. The power limit and the time window (Power, TimeWindow) form essential parameters of RAPL interface. Chapter 14.9 of the Intel Software Developer's Manual [1], provides detailed documentation of the current RAPL features.

Users measure and control processor power consumption using several model-specific registers, or MSR using two basic privileged instructions, readmsr and writemsr through the MSR kernel module. This module exports a file interface at `/dev/cpu/N/msr` (with `N` being the CPU number) that, given suitable file permission, can be used to read and write any MSR on the node.

Intel has separated the Sandy Bridge family into two classes namely *client* and *server*. The two architectures share a subset of RAPL features where the server class provides additional features such as DRAM power measurement. The Intel Xeon CPU E5-1650 6-core processor has been used in this work which provides server level RAPL features.

The Sandy Bridge architecture supports three power domains on both server and client architectures. Both architectures support package (PKG) and Power Plane 0 (PP0) domains, while the server adds a separate DRAM domain and the client adds a second power plane (PP1). The processor used in this work supports measurement and control of the DRAM domain. Therefore, the PKG and DRAM domain have been used to determine the instantaneous power consumption of the processor+DRAM with the change in P-states.

1.2. Employing DVFS

Various approaches exist to intelligently employ DVFS in the scientific parallel applications. The more sophisticated ones scale processor frequency on different intervals of application runtime while attempting to predict accurately the performance effects from the DVFS. Such approaches may be broadly classified into two types: One that first divides the application into execution intervals of predefined duration and then uses the performance counters to determine a suitable frequency for them [7, 10, 11]; and the other that first determines communication intervals in parallel applications that use either explicit message passing [6, 15, 22, 23] or global address-space primitives [24] and then scales the frequency for those intervals, usually based on the variation of the MIPS (million instructions per second) metric at different P-states. Typically these approaches first choose a (often user-defined) performance loss (PL) tolerance for the application and then try to maximize energy savings under this PL as constraint.

The value of the performance loss—if not chosen carefully—may negatively affect the energy change with frequency scaling during application execution. For example, a low PL value for a memory-intensive application or a high PL value for a compute-intensive one may actually result in an increase in an overall energy consumption of an application. One reason is that, typically, PL-based approaches do not take into account the instantaneous power consumption of the CPU at varying frequencies in their frequency-scaling decisions. To remedy this problem, the present work proposes an energy-saving scheme that is aware of the instantaneous CPU

power consumption, and thus, removes the necessity for user guesstimates in defining the performance loss by choosing a suitable frequency. The proposed scheme also presents a trace based workload predictor which considers all the previous workload traces instead of using the immediate history to predict the future workload characteristics. The experiments performed on NAS benchmarks depict that the proposed scheme works better than the one which employs a user-defined performance loss.

1.3. Related Work

There are two general approaches to obtaining energy savings during parallel application execution. The first approach is to focus on identifying stalls during the execution by measuring architectural parameters from performance counters as proposed in [7, 10, 11]. Rountree *et. al* [19], apart from using performance counters, do the critical path analysis to determine which tasks may be slowed down to minimize the performance loss in the parallel execution. This analysis appears beneficial when applications have computation or communication imbalances among participating processes, which is typically not the case for a highly efficient parallel application and which was not observed in the NAS benchmarks, for example. The workload prediction mechanism used in Adagio is similar to the last-value predictor but with feedback added, such that the future behavior of a communication call is predicted based on its last invocation and the resulting error is used as feedback for future predictions. Besides communications, Adagio also monitors computation parts of the application to determine suitable opportunities to apply DVFS.

The second approach determines the communication phases to apply DVFS as, for example, in [15] and [6]. Etinski *et al.* [5] propose a technique that applies both DVFS and over-clocking to the CPUs to save energy and to improve execution time. The main shortcomings of [5] are using simulation infrastructure instead of real computing platform and predetermining memory intensity of applications by fixing the value of a particular parameter, which can make a substantial difference in energy savings. Both these issues have been addressed sufficiently in the proposed scheme. DVFS is combined in [4] with concurrency throttling on multicore platforms to obtain energy savings. In [14], algorithms to save energy in the collectives, such as MPI_Alltoall and MPI_Bcast, are proposed. The work in [12] describes a runtime system for the Intel Single-chip Cloud Computer (SCC) processor. This system detects repeatable communication phases followed by an application of frequency scaling. In [16], a detailed comparison of the benefits offered by RDMA versus TCP/IP is given in terms of power efficiency. Authors in [21] have presented models for power-performance efficiency using performance counter data with machine learning for modern GPUs. DVFS is combined in [4] with concurrency throttling on multicore platforms to obtain energy savings. Ge *et. al* [8] study the impacts of frequency scaling on application performance and energy consumption for GPU computing. In [2], an analytical framework is proposed that studies the trade-offs between parallelism, performance, and overall energy consumption of an application based on Amdahl's law.

Both these approaches tend to choose a performance loss for the application and then attempt to minimize energy consumption under that performance loss. The work presented here discusses the possible pitfalls of the approaches that choose the performance loss *a priori* and proposes a novel power-aware scheme that takes into account the instantaneous power consumption of the platform at different frequencies to make frequency scaling decisions. Thus, the scheme leads to the DVFS approach independent of the chosen *a priori* performance loss. The proposed underlying model correlates the micro-operations retired to both the operating frequency and the number of memory accesses per micro-operation instead of just using the frequency [13, 15] so that a relatively low value of micro-operations retired, which are sometimes seen at a relatively high frequency, may be accounted for in frequency scaling decisions.

Paper organization. The rest of paper is organized as follows. Section 2 discusses potential problems with the PL-based techniques. Section 3 presents the design of the proposed energy-saving scheme. Section 4 discusses the experimental results, while Section 5 concludes.

2. GENERAL ENERGY CONSUMPTION MODEL

The execution time t of a program can be divided into two separate parts, on-chip time t_{on} and off-chip time t_{off}, such that t_{on} and t_{off} are non-overlapping [3], even though in an out-of-order (OOO) processor, a portion of t_{off} may overlap with t_{on}. This work determines only the "non-overlapping" portion of the off-chip time because only does this portion of t_{off} serves to analyze the energy-saving potential from applying frequency scaling.

The time t_{off} consists of stall cycles, such as memory, I/O, branch misprediction, and reservation station stalls, during which the PE is not doing any useful work. In an out-of-order processor, the stall cycles can also overlap with the on-chip execution. DVFS affects only t_{on} of the program execution as it scales linearly with the change in frequency whereas DVFS does not have any effect on the t_{off}.

Typically, during a PL-based energy-saving technique, a performance loss tolerance is prescribed by the user for a given application, and the energy savings are maximized under this tolerance. Let a PL-based technique increase t_{on} by a factor of k, so that $t(\overline{f})= kt_{on} + t_{off}$ by executing the application at an average frequency $\overline{f}$ by using DVFS. The total energy savings may appear if

$$P(f_1)t(f_1) > P(\overline{f})t(\overline{f}) \tag{1}$$

$$P(f_1)(t_{on} + t_{off}) > P(\overline{f})(kt_{on} + t_{off}) \ , \tag{2}$$

where $P(f_1)$ is the average power consumption of a compute node (CN) at the highest frequency f_1 and $P(\overline{f})$ is the average power consumption at frequency $\overline{f}$. The inequality Eq. (2) may be used to determine the feasibility of total energy savings, being re-written for convenience, as

$$\frac{t_{off}}{t_{on}} > \frac{kP(\overline{f}) - P(f_1)}{P(f_1) - P(\overline{f})} \ . \tag{3}$$

The average power consumption $P(\overline{f})$ can be defined as

$$P(\overline{f}) = \sum_{i=1}^{m} P(f_i)t(f_i) / \sum_{i=1}^{m} t(f_i) \ , \tag{4}$$

where m is the number of frequency transitions at the runtime, $P(f_i)$ and (ft_i) are the power consumption and time spent at the frequency f_i, $i = 1, \ldots, m$, respectively. For a PL-based scheme, the actual performance loss $\delta(\overline{f})$ may be calculated as

$$\delta(\overline{f}) = \frac{t(f_1) - t(\overline{f})}{t(f_1)} \ , \quad \text{where} \quad t(\overline{f}) = \sum_{i=1}^{m} t(f_i) \ . \tag{5}$$

The percentage power decrement at a frequency $\overline{f}$, (PD($\overline{f}$)) can be defined as,

$$PD(\overline{f}) = \frac{P(f_1) - P(\overline{f})}{P(f_1)}. \tag{6}$$

Using Eqs. (5) and (6) and rearranging Eq. (1), the necessary condition for obtaining energy savings is

$$\delta(\overline{f}) < \frac{PD(\overline{f})}{1 - PD(\overline{f})}. \tag{7}$$

Table I. Power decrements (PD) ratio for NAS EP benchmark at various frequency transitions (FC) on two platforms along with the resultant performance loss (PL).

FC	$\frac{PD}{1-PD}$ (FScal)	$\frac{PD}{1-PD}$ (Dynamo)	PL
3 GHz $\rightarrow$ 2.67 GHz	8.7%	5.2%	12.3%
3 GHz $\rightarrow$ 2.33 GHz	17.3%	12.3%	28.7%
3 GHz $\rightarrow$ 2.00 GHz	29%	21.9%	50%

2.1. Possible Pitfalls of PL-based Techniques

For an application operated under a PL-based technique, it can be observed from Eqs. (1) to (3) that the average power consumption and, consequently, the resulting energy consumption depends on the 1) ratio of t_{off} and t_{on}, 2) performance loss δ and 3) power consumption P_i of CN at frequencies f_i, $i = 1, \ldots, m$.

Hardware platforms differ as to their power decrements, i.e., the changes in power consumption provided by the application of DVFS, even for the same frequency switching values. For example, Table I shows the power decrement ratios (Right Hand Side of Eq. (7)) for various frequency transitions (column `FC`) in two platforms, namely "FScal" and "Dynamo", executing a compute-intensive NAS EP benchmark. The FScal platform consists of an Intel core 2 Duo processor with 2 GB of RAM, whereas a node of Dynamo cluster has two Intel Xeon quad-core processors with 16 GB of RAM.

It can be observed from Table I that, for the same frequency transitions, the power decrement ratios differs significantly in FScal and Dynamo, even though the performance loss percentage stays the same since the EP benchmark has zero off-chip time t_{off}. Consider a scenario in which the performance loss chosen for the EP benchmark is greater than or equal to 12.3%. Then, the PL-based technique will *increase* the energy consumption since the decrease in the power consumption will not be enough to mitigate the degradation in performance. If the performance loss is always kept low to warrant against possible energy increases, then a PL-based technique may not achieve any energy savings for moderately memory-intensive applications. Hence, choosing a performance loss without actually knowing the change in power consumption of the node at various frequencies may result in no energy savings or—even worse—in an increase in energy consumption.

3. DESIGN AND IMPLEMENTATION OF THE NOVEL ENERGY-SAVING SCHEME

There are three components of the proposed scheme: 1) Phase, 2) Workload modeling, and 3) RAPL and frequency scaling infrastructure. The phase component characterizes and predicts the future phases of an application according to certain parameters obtained during runtime, which provides information regarding the degree to which the application is compute- or memory-intensive. After determining the phase information, the workload model uses it to predict the performance of an application at various frequencies. Finally, the RAPL and frequency scaling component makes use of the instantaneous power consumption to choose a frequency for the application which will minimize the energy consumption.

3.1. Phase Characterization

Phase characterization of an application is needed to characterize an application execution interval (timeslice) by degree of its memory intensity and guide the runtime energy saving scheme. The phase classification can be done by making use of either the architectural parameters or the application communication characteristics.

In this work, the phase characterization has been done through hardware performance monitoring counters (PMCs), which periodically provide the information regarding the

Table II. Range of MAPM values and phases associated with them.

MAPM Value	Phase
< 0.18	1
$[0.18, 0.20)$	2
$[0.20, 0.22)$	3
$[0.22, 0.24)$	4
$[0.24, 0.26)$	5
≥ 0.26	6

architectural parameters during application execution. The period for which the information is obtained is chosen so as to minimize the overhead on the application performance. The memory boundedness of an application is defined as the ratio of the *memory accesses* to *micro operations retired*, commonly known as Memory Access Per Micro-operation (MAPM). MAPM will provide the information regarding the slack based on which the processor frequency can be reduced to save energy without significantly harming the performance. The reason for using MAPM for the purpose of phase characterization is because it is invariant to the changes in frequency for a given application whereas other parameters such as *micro-operations per cycle* fluctuate drastically. Due to this property of the MAPM parameter, the phase behavior of an application can be effectively gauged during runtime even when it is subjected to varying frequencies. Based on the value of MAPM, a specific application execution interval is assigned a particular phase type as shown in Table II which defines 6 phase types. The MAPM values corresponding to Phase type 1 and 6 have been selected according to the runtime behavior of EP and CG benchmarks, were found to be the most compute-intensive and memory-intensive, respectively, from the experiments conducted on the hardware platform used in this work. Therefore, all the phase types have been assigned based on the MAPM values of these two benchmarks. Consequently, applications exhibiting the lowest phase are largely compute-intensive and are to be operated at the highest frequency. Similarly, applications depicting a larger value of phase are primarily memory-intensive and can be operated at less than maximum frequency to save energy.

3.2. Phase Prediction

To be able to predict the future phase behavior is of utmost importance for an energy-saving scheme so that it can select an appropriate frequency for a particular execution interval at its beginning. Various predictors have been proposed in the past the most prominent of which have been the *last value* and *history window* based predictor [25, 13]. The *last value* predictor predicts the next phase same as the last seen phase, i.e., $Phase[n+1] = Phase[n]$. The *history window* based predictor considers a window of past k phase values and predicts the future phase as a function of these values i.e.,

$$Phase[n+1] = f(Phase[n], Phase[n-1], \ldots, Phase[n-k+2], Phase[n-k+1]).$$

Figure 1 depicts the execution trace of NAS EP, CG, and MG benchmarks in terms of phases exhibited by rank 0, when these benchmarks were executed on a single node for 5 seconds with 250ms timeslice.The phase values for EP and CG benchmarks do not seem to change during the execution. Therefore, the *last value* and *history window* predictors will be quite accurate in predicting the future phases of these two benchmarks. But in the case of MG benchmark, the phase values keep varying in an iterative manner. For the benchmarks depicting phase behavior similar to MG, the *last value* and *history window* predictors will not be able to predict phases accurately, thus misleading the frequency scaling operation.

This work proposes a novel History Trace Based Predictor (HTBP) that is based on the *two-level branch prediction* mechanism [18, 13]. The fundamental idea behind the operation of HTBP is to store traces of phases in a *phase table* (PT) which is a hash table of size N

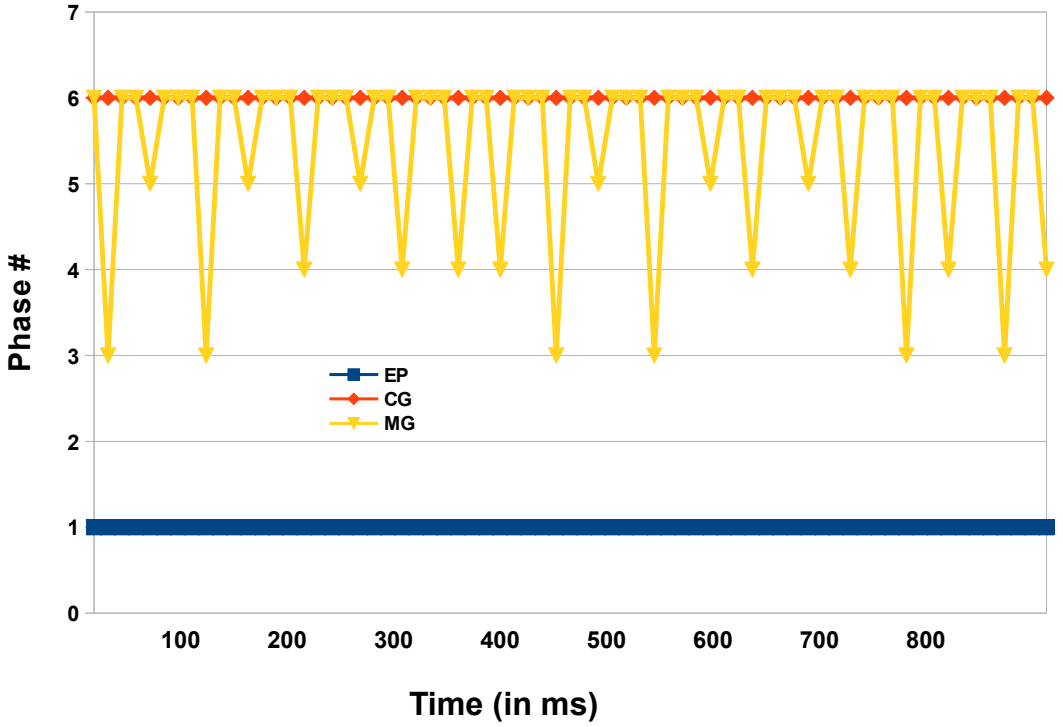

Figure 1. Execution trace for EP, CG, and MG benchmarks depicting their phase behavior for rank 0.

Table III. Mapping of phase type to corresponding MAPM values.

Phase	MAPM Value
1	0.18
2	0.19
3	0.21
4	0.23
5	0.25
6	0.26

and index the phase table by a *phase shift register* (PSR). Implementing PT as hash table features an $O(1)$ access time compared to an array-based implementation, which may take $O(NM)$ time, where N is the size of the array and M is the length of the PSR. Using the hash table also reduces the amount of memory required to store the phases since new phases can be appended by chaining, when they appear, unlike in an array, where memory allocation has to be done at one time.

The PSR contains the value of previous M phases observed where M is the chosen depth of PSR. The PSR is shifted left by a single step for each execution interval and the PT is checked whether or not that particular instance of PSR is present in the PT. If present, the predicted value of the phase is obtained from the PT. Otherwise, the PSR value is stored in the PT with predicted phase value as the phase of next execution interval. Figure 2 shows the design of the HTBP implemented in this work. The PT is a hash table indexed by a key, which is obtained by hashing the current instance of the PSR through a hash function.

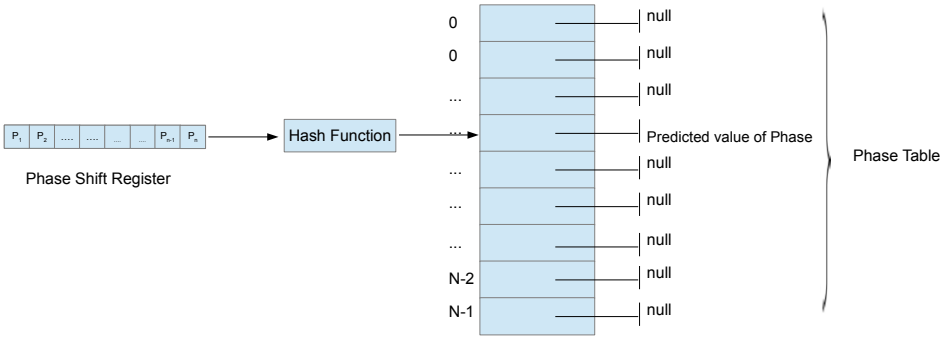

Figure 2. History trace based predictor structure.

3.3. Workload Modeling

A model is needed which can effectively map the variation in frequency to the application performance using the predicted phase type information.The effect of frequency scaling on application execution time can be determined through MIPS (millions of instructions per second) rate. For the proposed scheme, a model based on the MIPS metric is used, which correlates the execution time impact with corresponding frequency and MAPM changes, given the micro-operations retired $\mu\tau(f_i)$ at a particular frequency $f_i,\ i = 1, \ldots, n$, an MAPM value α and a timeslice of given duration.

$$\mu\tau(f_i) = a \times f_i + b \times \alpha + c\ . \tag{8}$$

Equation (8) depicts that the value of micro operations retired at a particular frequency is dependent upon both the operating frequency and the MAPM value. The parameters a, b and c are determined through linear regression analysis during runtime. The MAPM values can be obtained by using the mappings provided in Table IV which match a phase types to a particular MAPM values. If an application is compute-intensive , the coefficient of MAPM b will be about zero indicating that the change in micro-operations retired is almost linear with the change in frequency. On the other hand, for a memory-intensive application, b will be a significantly large nonzero indicating that decreasing frequency does not result in a linear decrease of the performance.

The micro-operations retired at a particular frequency and, thus, the MAPM value are related to the performance loss δ as:

$$\frac{\mu\tau(f_1) - \mu\tau(f_i)}{\mu\tau(f_1)} = \delta_i\ , \tag{9}$$

where δ_i is the resultant performance loss from a change of the maximum frequency f_1 to a lower one f_i. The performance loss values at various frequencies are used to determine the normalized instantaneous energy consumption for future timeslices, thus guiding the scheme for choosing a suitable frequency.

3.4. RAPL and Frequency Scaling

To introduce the instantaneous power consumption into the proposed scheme and to make it user-defined performance loss free, Intel RAPL (running average power limit) technology is used which provides instantaneous core and DRAM power consumptions. Using the model specific registers MSR_PP0_ENERGY_STATUS and MSR_DRAM_ENERGY_STATUS, the core and DRAM power consumption denoted as P_{core} and P_{DRAM}, respectively, can be continuously monitored. The instantaneous power consumption which is obtained by using RAPL is DC in nature. It may be converted to the corresponding AC power by using an

Input Parameters:
$f_1, \ldots, f_n$: Available frequencies in CN.
$s_0, \ldots, s_m$: Timeslice sequence of application execution.
Phase Shift Register (PSR) of length M. $(M = n)$
Phase Table (PT).
Mapping table MAPM which returns an MAPM value for a given phase type.

Algorithm:
Step 1. Initialize $\mu\tau(f_i)$, $P(f_i)$, and PSR$[0 \ldots n-1]$ for $i = 1, 2, \ldots, n$ for the first n timeslices.

For $(j = n,\ j < m,\ j++)$ **do**
Step 2. If entry in PT corresponding to PSR$[0 \ldots n-1]$ is NULL
Predicted_Phase=$(\text{PSR}[n-1] + \text{PSR}[n-2] \ldots \text{PSR}[n-k])/k$.
else
Predicted_Phase=PT[Hash(PSR$[0 \ldots n-1]$)].

Step 3. Calculate parameters a, b, and c by linear regression in Eq. (8).
Step 4. Get the next timeslice s_{j+1}.
Step 5. Predict the values of $\mu\tau(f_i)$ in s_{j+1} using a, b, and c.

$$\mu\tau(f_i) = a \times \mu\tau(f_i) + b \times \text{MAPM[Predicted\_Phase]} + c, \quad i = 1, \ldots, n\ .$$

Step 6. Calculate the performance loss δ_i for each frequency f_i:

$$\delta_i = \frac{\mu\tau(f_{1)} - \mu\tau(f_i)}{\mu\tau(f_1)}\ .$$

Step 7. Choose the operating frequency f_o for s_{j+1}, such that

$$P(f_o)(1+\delta_o) = \min_{i=1,\ldots,n} [P(f_i)(1+\delta_i)]\ .$$

Step 8. PT[Hash(PSR$[0 \ldots n-1]$)]=Current_Phase.

Step 9. Shift PSR left by single step and PSR$[n-1]$=Current_Phase.

Step 10. Update $P(f_o)$ and $\mu\tau(f_o)$.
EndFor

Figure 3. Power-aware energy-saving scheme.

appropriate scaling factor s, which was determined experimentally for the platform used in this work. Hence, the total power consumption $P(f_i)$ of a compute node at a frequency f_i, can be calculated as,

$$P(f_i) = (P_{core} + P_{DRAM}) \times s + P_{static}\ , \tag{10}$$

where P_{static} is the static power consumption of the compute node. To modify the frequency of the cores, a specific value is written to the MSR IA32_PERF_CTL which has address $0x199$. The delay of modifying frequency runs in order of ∼20 microseconds which can be easily compensated for by choosing a suitable value for the timeslice.

3.5. Step-by-Step Description of the Scheme

Figure 3 displays the steps of the proposed scheme. Step 1 initializes the $\mu\tau(f_i)$,$P(f_i)$ $i = 1, \ldots, n$ and PSR register (with the `Current_Phase` values) for the first n timeslices of the application execution.

The power consumption $P(f_i)$ is obtained from Eq. (10) and the values of $\mu\tau(f_i)$ and PSR are determined using the performance counters. Then (in Step 2) the Phase table (PT) is indexed by using the PSR to determine whether an entry at that location in the PT is present or not. If present, then that value is considered as the predicted phase for the next timeslice, else a *history window* based predictor is used to predict the phase for the next timeslice. In Step 3, linear regression is performed to predict the values $\mu\tau(f_i)$, $i = 1, \ldots, n$, to calculate the value of the regression parameters (8) and also to predict the values of $\mu\tau(f_i)$ using the predicted phase value obtained from Step 5. The performance loss is calculated in Step 6 at each frequency by using the predicted micro-operations retired at the frequencies $f_1, \ldots, f_n$. Next, the frequency f_o that results in the minimum energy consumption under the estimated performance losses is calculated in Step 7 and is chosen as the operating frequency for the next timeslice. At the end of timeslice, the PT is updated by storing the value of the current phase (Step 8) and subsequently shifting the PSR to left by a single step and storing the value of the current phase at its end (Step 9). In Step 10, the $P(f_o)$ and $\mu\tau(f_o)$ are obtained for this operating frequency f_o; and the for-loop repeats if the timeslices are available.

4. EXPERIMENTAL RESULTS

Experimental platform. The experiments were performed on 6 nodes of the computing platform Bolt, which comprises 18 Infiniband QDR-connected compute nodes, each of which has 32 GB of main memory and an Intel Xeon CPU E5-1650 6-core processor. The Intel Xeon CPU E5-1650 provides fifteen P-states ranging from 1.2 to 3.2 GHz, out of which four P-states are used in this work which are 3.2, 2.5, 1.8, and 1.2 GHz. To measure the *whole* system power and energy consumption, a Wattsup[†] power meter was employed with a sampling rate of 1 Hz. This low sampling rate of Wattsup does not affect the measurements considerably since all the benchmarks used have a rather large execution time. The static power consumption of the platform P_{static} was determined as 126 watts and the value of timeslice s was chosen as 250 ms. The NAS parallel benchmarks are used to demonstrate efficacy of the proposed power consumption aware scheme. Their classes are chosen such that the benchmarks are executed in a reasonable time, i.e., to accommodate the Wattsup granularity without overburdening the computing platform.

Phase prediction accuracy. First the phase prediction accuracy of the predictors discussed in Section 3.1 is evaluated for NAS benchmarks. The EP, CG, LU, SP and BT benchmarks depict a linear phase type behavior i.e their phase types rarely change during the execution. Therefore, the HTBP and *history window* predictors perform identically as they achieve a near 100% phase prediction accuracy for these benchmarks. The MG and FT benchmark on the other hand exhibit a variable phase behavior as shown in Fig. 4(a) and Fig. 4(b). The phase prediction accuracy for MG benchmark operated under HTBP and *history window* predictors is 71% and 89%, respectively. The phase type variability for the FT benchmark is so much more as compared to MG that both HTBP and *history window* predictors have very low phase prediction accuracy (20% and 10%, respectively). It can be inferred from Fig. 4 that HTBP performs better than the *history window* predictor does so even when the phase behavior of a benchmark is highly variable.

[†]Wattsup power meter (`https://www.wattsupmeters.com`) records the *total* power for the computing system to which it is connected.

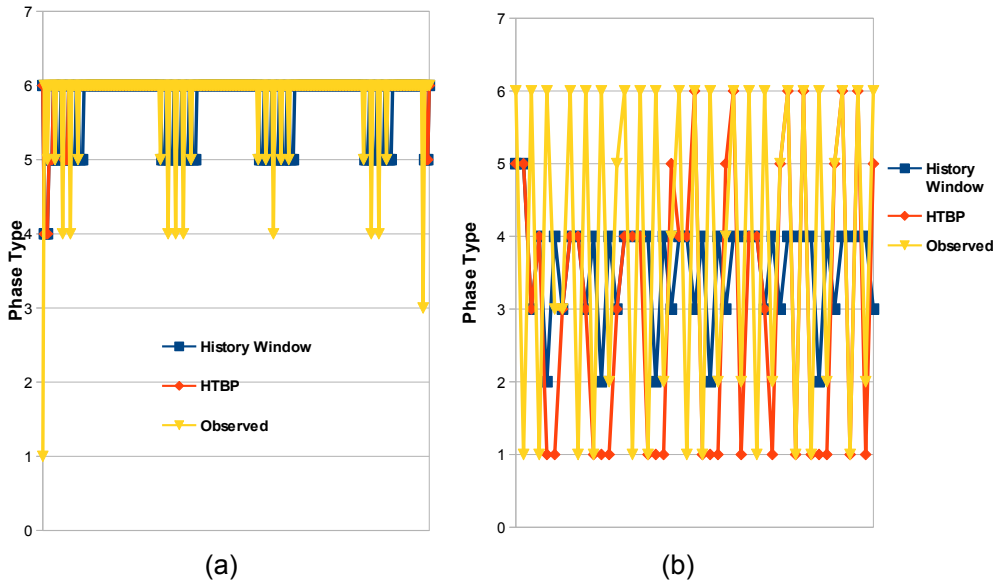

Figure 4. Phase type behavior of (a) MG and (b) FT benchmarks.

Setup for the PL-based experiments. The CPU Miser package [7], which is a state-of-the-art software for implementing DVFS in applications, was chosen to experiment with the schemes that assign an *a priori* value of the performance loss (denoted here as PL-based). In particular the CPU Miser works as follows. It divides the execution of an application into intervals of a particular duration (typically 250 μs) and predicts the execution characteristics, such as memory stalls, of the upcoming interval based on recent intervals similar to the *history window* predictor. The CPU Miser primarily depends on the memory accesses, even though it may use the I/O and idle times (provided by the `/proc/stat` file in Linux) to choose a suitable frequency for a given time slice. CPU Miser provides for a user to define the performance loss, given which it attempts to save energy; and it has been shown to attain significant energy gains [7]. Note that, similarly to the proposed work, CPU Miser requires no modification to the application source code. However, it does not consider the instantaneous power consumption of the unit under test (UUT) for choosing a suitable frequency. The CPU Miser technique is chosen to evaluate the PL-based approach for the three values of performance loss, namely, 5%, 10%, and 20%.

4.1. Performance

For the seven NAS benchmarks operated under the three PL-based schemes (denoted as *5% PL*, *10% PL*, and *20% PL*) and the proposed scheme (denoted as *Proposed*), Fig. 5 depicts the performance loss compared to the full-power execution where the benchmarks are executed at the highest available frequency of the processor.

The EP benchmark is executed at the highest frequency by *Proposed*, *5% PL*, *10% PL* and at 2.5 GHz by *20% PL* scheme with resultant performance loss of 1% and 26% for the corresponding frequencies. Both the *5% PL* and *10% PL* schemes incurred considerable performance losses of ∼20% and ∼60% for the compute-intensive LU and BT benchmarks by executing them at 2.5 and 1.8 GHz, respectively. The reason for the performance overhead in the PL-based schemes as implemented in the CPU Miser is that it attempts to determine the precise number of stalls in an OOO execution engine, which is more complex compared to estimating the change in the application performance using micro-operations retired as

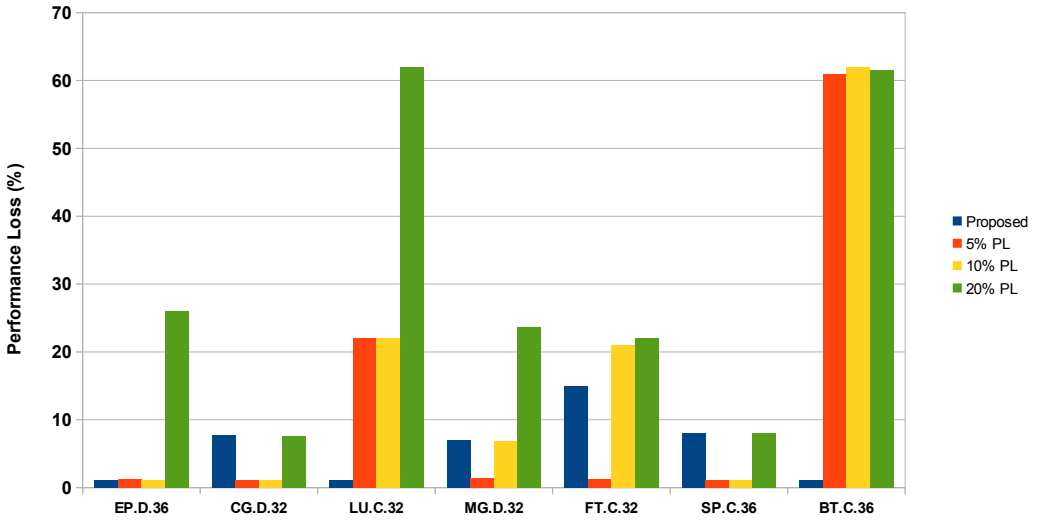

Figure 5. Performance loss for the seven NAS benchmarks under different frequency scaling schemes.

in Eq. (9). Also, CPU Miser does not consider contention among the cores while modeling the DRAM-access delay. Therefore, it either overestimates or underestimates the number of stalls in an application, and thus, chooses a relatively low or high frequency depending on the performance loss. The CG, SP, MG, and FT benchmarks are shown in the reducing order of their memory-access intensity. Recall that CG and SP depict linear phase behavior while MG and FT exhibit variable phase behavior. Therefore, the resultant performance loss values are near identical for CG and SP for all the four frequency scaling schemes whereas they differ in the case of MG owing to their variable phase behavior and the phase mispredictions. The average performance losses for the *Proposed*, *5% PL*, *10% PL*, and *20% PL* schemes are 5.8%, 11.2%, 14.51% and 28.5% for the eight NAS benchmarks, respectively. Hence, one may conclude that the *Proposed* scheme results in a lower performance degradation compared to the PL-based schemes as implemented in the CPU Miser.

4.2. Energy

Figure 6 depicts the change in energy savings with respect to the full-power execution for the seven NAS benchmarks, where a negative value in the graph indicates an increase in overall energy consumption. In Fig. 6, under the *5% PL* scheme and the highest operational frequency as discussed in Section 4.1, a slight energy increase of about 1% is observed for all the benchmarks, except for LU, which resulted in some energy savings of about 2.5%, and for BT, which actually increased the energy consumption because of long execution time at 1.8 GHz. Although the *10% PL* and *20% PL* schemes have rather large freedom to decrease the frequency owing to their higher pre-set value of performance loss, they do so without considering the instantaneous power consumption of the UUT. Therefore, these schemes end up executing the benchmarks at the operating frequency of either 2.5 or 1.8 GHz and, according to the instantaneous power measurements of the UUT, increase or decrease too much the overall power consumption. The average energy savings for the *Proposed*, *5% PL*, *10% PL*, and *20% PL* schemes are 6.1%, −2%, 1.8%, and 0.8%, respectively, for all the benchmarks. Hence, the *Proposed* scheme produces the highest energy savings.

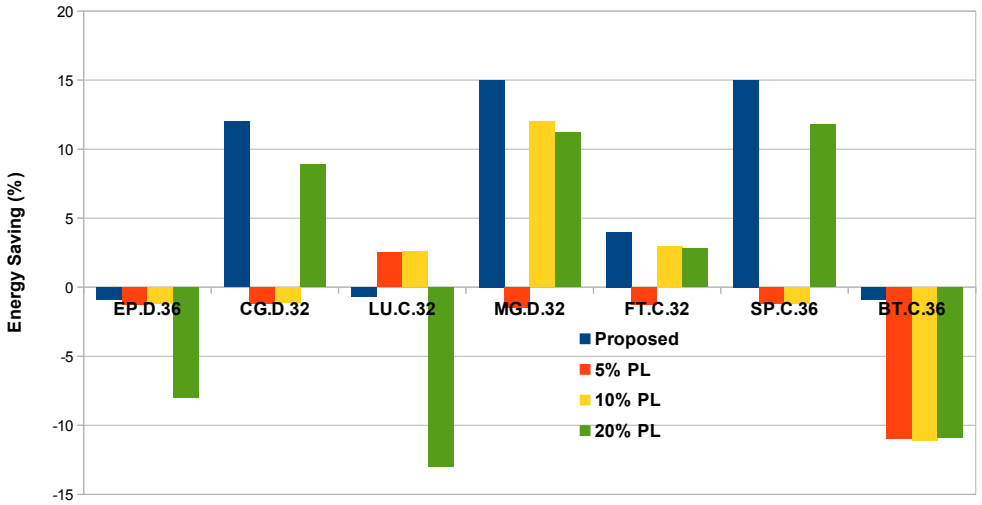

Figure 6. Energy savings for the seven NAS benchmarks under different frequency scaling schemes.

Energy-delay product. The energy-delay product (EDP) [9] has been widely considered as a metric that is used to couple the energy consumption with performance of various architectures and technologies. By accounting for the execution time as well power, EDP has a clear advantage over measuring a "raw" power consumption because, for different operating frequencies, both energy consumption and execution time vary even for the same benchmark. In other words, EDP is a *fused* metric that quantifies the energy-performance efficiency.

It has been observed in the experiments that EDP is marginally increased by ∼1% under the *Proposed* scheme for the EP, LU, and BT benchmarks because they are executed at the highest frequency and suffer only from the profiling overhead. The *5% PL*, *10% PL*, and *20% PL* end up increasing EDP substantially for nearly all the benchmarks because of significant performance degradation. On average, for the NAS benchmarks, the *Proposed* scheme decreases the EDP by 1.2%, whereas the *5% PL*, *10% PL*, and *20% PL* schemes increase their EDP by 8%, 9%, and 25%, respectively.

5. CONCLUSIONS AND FUTURE WORK

In this paper, a model is proposed, which aims to (1) predict the micro-operations retired at different frequencies and the memory accesses per per micro-operation (MAPM) values by using a linear regression analysis, (2) record the instantaneous power consumption by using the Intel RAPL technology, (3) determine the energy consumption of the platform at different frequencies based on the predicted performance and recorded power consumption, and (4) chooses the scaling frequency at which the energy consumption is at the minimum. The model operates on the timeslices grouped into phases according to the proposed here prediction mechanism, which considers *all* the previous workload history traces, rather than just the immediate history, to determine the future phases. This paper shows that such a mechanism increases the prediction accuracy significantly. When compared with approaches that choose the performance loss *a priori*, the proposed here power-aware scheme, incorporating the new model and phase prediction mechanism, delivers better overall energy savings and a smaller performance loss. Future work will extend the proposed scheme to heterogenous computing

platforms comprising both CPUs and GPUs by using their built-in capabilities to get the instantaneous power consumption.

References

1. M. Annavarami, E. Grochowski, and J. Shen. Intel software programmer's guide-combined volumes 3a and 3b. Washington, DC, USA, 2014. IEEE Computer Society.
2. S. Cho and R. G. Melhem. On the Interplay of Parallelization, Program Performance, and Energy Consumption. *IEEE Transactions on Parallel and Distributed Systems*, 21:342–353, 2010.
3. Kihwan Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(1):18 – 28, 2005.
4. M. Curtis-Maury, A. Shah, F. Blagojevic, D.S. Nikolopoulos, B.R. de Supinski, and M. Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 250–259, New York, NY, USA, 2008. ACM.
5. M. Etinski, J. Corbalan, J. Labarta, M. Valero, and A. Veidenbaum. Power-aware load balancing of large scale mpi applications. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, May 2009.
6. V.W. Freeh and D.K. Lowenthal. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 164–173, 2005.
7. R. Ge, X. Feng, W. Feng, and K.W. Cameron. CPU MISER: A performance-directed, run-time system for power-aware clusters. In *Parallel Processing, 2007. ICPP 2007. International Conference on*, page 18, Sep. 2007.
8. R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 826–833, Oct 2013.
9. R. Gonzales and M. Horowitz. Energy dissipation in general purpose processors. *IEEE Journal of Solid State Circuits*, 31:1277–1284, 1995.
10. C.H. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 1, nov 2005.
11. S. Huang and W. Feng. Energy-efficient cluster computing via accurate workload characterization. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 68–75, May 2009.
12. N. Ioannou, M. Kauschke, M. Gries, and M. Cintra. Phase-Based Application-Driven Hierarchical Power Management on the Single-chip Cloud Computer. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 131–142, oct. 2011.
13. C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 359–370, Dec 2006.
14. K. Kandalla, E.P. Mancini, S. Sur, and D.K. Panda. Designing power-aware collective communication algorithms for InfiniBand clusters. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 218–227, 2010.
15. M.Y. Lim, V.W. Freeh, and D.K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.
16. J. Liu, D. Poff, and B. Abali. Evaluating high performance communication: a power perspective. In *Proceedings of the 23rd International Conference on Supercomputing*, pages 326–337, 2009.
17. J. Park, D. Shin, N. Chang, and M. Pedram. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In *2010 International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 419–424, 2010.
18. David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
19. B. Rountree, D.K. Lownenthal, B.R. de Supinski, M. Schulz, V.W. Freeh, and T. Bletsch. Adagio: making dvs practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.
20. Scientific grand challengies: Crosscutting technologies for computing at the exascale. In *U.S. Department of Energy sponsored workshop*, Washington, DC, Feb. 2010.
21. S. Song, C. Su, B. Rountree, and K. W Cameron. A simplified and accurate model of power-performance efficiency on emergent gpu architectures. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 673–686, Washington, DC, USA, 2013. IEEE Computer Society.
22. V. Sundriyal, M. Sosonkina, A. Gaenko, and Z. Zhang. Energy saving strategies for parallel applications with point-to-point communication phases. *J. Parallel Distrib. Comput.*, 73(8):1157–1169, August 2013.
23. V. Sundriyal, M. Sosonkina, and Z. Zhang. Automatic runtime frequency-scaling system for energy savings in parallel applications. *The Journal of Supercomputing*, 68(2):777–797, 2014.
24. A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji. Designing Energy Efficient Communication Runtime Systems for Data Centric Programming Models. In *Proceedings of*

the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, GREENCOM-CPSCOM '10, pages 229–236, Washington, DC, USA, 2010. IEEE Computer Society.

25. T. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, MICRO 24, pages 51–61, New York, NY, USA, 1991. ACM.