DNA-inspired Scheme for Building the Energy Profile of HPC Systems

Ghislain Landry Tsafack Chetsa, Laurent Lefevre, Jean-Marc Pierson, Patricia Stolf, Georges da Costa

DNA-inspired Scheme for Building the Energy Profile of HPC Systems

Ghislain Landry Tsafack Chetsa[1,2], Laurent Lefevre[1]
{ghislain.landry.tsafack.chetsa, laurent.lefevre}@ens-lyon.fr
Jean-Marc Pierson[2], Patricia Stolf[2], and Georges Da Costa[2]
{stolf,dacosta,pierson}@irit.fr

[1] INRIA, LIP Laboratory (UMR CNRS, ENS, INRIA, UCB)
Ecole Normale Superieure de Lyon, Université de Lyon, France
[2] IRIT (UMR CNRS), University of Toulouse
118 Route de Narbonne, F-31062 Toulouse CEDEX 9, France

Abstract. Energy usage is becoming a challenge for the design of next generation large scale distributed systems. This paper explores an innovative approach of profiling such systems. It proposes a DNA-like solution without making any assumptions on the running applications and used hardware. This profiling based on internal counters usage and energy monitoring allows to isolate specific phases during the execution and enables some energy consumption control and energy usage prediction. First experimental validations of the system modeling are presented and analyzed.

1 Introduction

Software solutions to improve the energy consumption of computing systems often fall into two categories, namely application-level solutions and system-level solutions. At the application level, a power-aware application design can use technologies such as Dynamic Voltage and Frequency Scaling (DVFS) for CPU energy reduction and the Low Power Idle (LPI) techniques for networks to reduce the application power usage at runtime with low performance penalty [12]. Code instrumentation can be used to adapt the processor's frequency and voltage according to the application's needs, so to reduce the application's overall power consumption [13, 9, 14]. Instrumented-code, inserted into the application source code, aims at dividing the application into regions having the same characteristics (memory intensive, CPU intensive, network intensive or disk intensive). The appropriate "power-saving" scheme (often the CPU frequency and voltage or P-state) is then selected for each defined region accordingly.

Software approaches could be very effective if a system designer (cloud or grid provider) has sufficient knowledge of the applications running on its system. For example an action applied to a single node may impact the power consumption of the remaining nodes running the same application and therefore the overall system energy consumption. Therefore, an effective use of these power

saving schemes on a multi-node system require a view of the system as a whole. The commonly used approach consists of instrumenting the source program to determine when to trigger an action on a computing node and how to handle that action's side effects on the remaining nodes running the same application. Instrumenting the program source code requires a specific expertise from the platform provider, which can not be assumed. Our work overcomes that necessity of instrumenting the application source code in order to apply power saving schemes.

A large body of work highlights the effectiveness of high performance computing (HPC) applications' power and energy consumption estimation via dedicated system registers known as hardware monitoring counters or performance counters. Hardware monitoring counters provide a means to understanding an application's resource utilization patterns. Power or energy estimation models based on hardware performance counters are often tailored for a specific type of applications: Indeed a power consumption model built for a workload having frequent memory accesses may not fit well a workload having infrequent memory accesses, and the same goes for frequent disk/network accesses workloads. Hence we believe that it is necessary to model phases of applications instead of applications, and therefore it is first needed to recognize and identify the successive phases of the system.

This paper studies how to represent the system into different phases considering their power consumption and hardware performance counters (including disk read/write and network bytes received/sent count) accessed throughout the phase. Observations led us to the following assumptions: (a) Hardware performance counters are accurate predictors of the system's energy consumption, (b) Any change in the set of performance counters relevant to power consumption prediction over a given duration reflects a change in the system's computational state over that duration. Based on these assumptions, we propose an approach to model the whole system's runtime (from a single node system to a cluster system) as a sequence of phases where each phase exhibits a computational behavior. This modeling paves the way to providing fine-grained control of HPC application power consumption without a priori knowledge of the application.

The major contributions of this paper are the following:

- We propose a DNA-like system model in order to better control large scale systems energy consumption. We focus on dividing applications into different phases. Its effectiveness is proven experimentally.
- We develop a cross platform energy consumption prediction model as a use-case of our system modeling approach.

This paper is organized as follows. Background and related work are presented in Section 2. We propose a representation model denoted as "DNA-like" representation of a system in Section 3. Based on this model, we propose an application mapping to the system representation and present some experimental results in Section 4. Section 5 concludes and introduces future works.

2 Background

Many applications have recurring phases during execution. Many works investigate methods to use these phases for architectural and system adaptations. Weissel et al. propose to adapt processor execution by monitoring memory boundedness of applications [17] . Dhodapkar et al. investigate dynamic hardware reconfiguration [7]. Murali Annavaram and al. [1] also point out the use of sequential and parallel phases in parallel applications for efficient distribution of threads on an asymmetric multiprocessor. These works open the door to any kind of system's optimization based on phases recognition including controlling and limiting the power consumption of large-scale systems.

A common approach to minimize the power consumption of large-scale systems is to identify phase-based applications [13]. The core idea is to schedule the processors to a higher frequency for CPU (Central Processing Unit) intensive phases, and to a lower frequency for non-CPU bound phases and workload [13, 6]. Focusing on highly iterative applications which allow them to predict the future behavior of an application based on its past, Freeh et al. identify nodes having been assigned small computations to reduce their frequency [8]. These approaches are quite effective, however to our knowledge the source code of the application needs to be analyzed carefully to determine the different phases thus the CPU frequency at which each phase should run. Therefore, detecting phases is not only application specific but also requires extensive knowledge given the complexity of today's high performance applications. In comparison, our work does not attempt to estimate or reduce the system's power or energy consumption, instead, we investigate the possibility of detecting and characterizing application's execution phases at runtime for using insights gathered form hardware monitoring events and the system's energy consumption. The particularity of our approach is that it almost does not require any knowledge of the application under consideration.

3 DNA-like System Modeling

As mentioned earlier in this paper, a fine-grained power control of a whole cluster system requires a view of that system as a whole. In this section, we introduce a very simple model for describing a system's runtime behavior. We assume a system or a cluster system is a set of computing nodes with their applicative states (the system state during application execution); network devices such as routers and switches are not taken into account. The strength of our model is that it provides insights about the system's computational state as well as its power/energy consumption. Our model represents a cluster system runtime as a state graph whose initial and final states are the system's idle state (or configuration). A transition from one state S1 to S2 is weighted by the conditional probability that the system goes to S2 given that it is in S1.

We represent each system state including the idle state as a column vector of size n; where n is the number of computing nodes in the system, and whose

entries describes the system behavior over a fixed period of time. An entry of a system's state vector is defined considering what we call the "DNA-like" structure of the system, which is a succession of computational behaviors exhibited by the system over time. We call such a computational behavior a "letter" and the set of possible computational behaviors the "system description alphabet".

A letter itself is modeled as a column vector of hardware monitoring counters including disk read/write and network bytes (respectively packets) sent/received counts to capture non memory- or CPU-intensive behavior. Details on their construction are provided later in this paper. On some platforms, it may not be possible to measure the system power/energy consumption using the power distribution unit. To avoid this limitation, one of our design constraint is that a letter in addition to describing the computational system, should provide a way to estimate the system power/energy consumption over a time period.

3.1 Letter Model

Observation Efforts to model HPC applications power/energy consumption via performance monitoring counters have shown that performance monitoring counters relevant to power consumption estimation depend on the application itself. Thus performance counters relevant to power consumption estimation of a CPU intensive application may differ from those relevant to power consumption estimation of a memory intensive application.

As performance monitoring counters relevant to power consumption estimation depends on the computational state of the application, we state that *any change in the set of performance counters relevant to power consumption estimation of an application over a time period T reflects a change in the application computational state over the same time period T.*

Based on the above statement, we propose Algorithm 1 which takes as input an application footprint as a matrix and outputs the partition of the application into computational phases according to changes in the set of performance monitoring counters relevant to the power consumption estimation. Each row r of the input matrix contains values of hardware monitoring counters recorded over the sample interval timestamped r.

Relevant hardware Counters Definition The steps to find out which performance counters are relevant to power estimation are the following: (1) the most straightforward approach is to only take into account the first k performance counters highly correlated to power consumption, (2) the second approach relies on the power model described by $Power \sim \sum_{i=1}^{n} \alpha_i * C_i$ or a similar one (linear or not). In this equation α_i and C_i are model coefficients and hardware counters respectively. We conduct a multi-variable linear regression to obtain coefficients α_i and retain counters C_j exhibiting a 5% level of statistical significance to power consumption estimation given the power model.

Figures 1, where similar patterns refer to the same computational behavior, shows the output of our algorithm considering a constant number (4 in this case)

Data: A: a matrix representing the application footprint, $P = \emptyset$
Result: P: a set containing subsets R_i representing application phases
Initialization: consider a sample interval S_n of upper bound n, n represents the first n rows of the application matrix A such that $n > p + 1$, where p is the number of performance counters (including disk read/write and network packets (bytes) sent/received count).
$P = P \cup R_0$
while *row(A)* $> n$ **do**
  Shift the upper bound of S to $n = upper\_bound(S) + n$
  Compute the set R_i of relevant counters from S_n
  if $R_{i-1} \neq R_i$ **then**
    Find $j \in [upper\_bound(S) - n, upper\_bound(S)]$ such that $R_j == R_{i-1}$, where R_j is the set of relevant counters from Sj;
    $P = P \cup R_j$
    Delete the first j rows of A
    Go to 1 (Initialization)
  end
end

Algorithm 1: Algorithm to detect application phases

of hardware counters. The system under consideration comprises two computing nodes running the NAS LU (class C problem on sixteen processes) benchmark [2]. The second approach showed bad results as the power estimation error using the power model is sometimes too high. We can also noticed that four phases was detected on the first computing node (on the x-axis from 0 to 60, form 60 to 90, from 90 to 120, and finally from 120 to 160) and five on the second computing node.

Letter Encoding and Representation In this section, we present a formal representation of letters of our system description model. To simplify we consider that a letter has four hardware monitoring counters i.e., the number of hardware monitoring counters representing a letter is fixed to 4. We also limit the overall number of hardware monitoring counters to sixteen. Although it is possible to monitor more than 64 hardware counters on a single node of our cluster, only a few of them are relevant to the system's power consumption estimation [5, 15].

Let's assign each hardware monitoring counter (including disk reads/writes and network (packets) sent/received bytes) to a four-bit aggregation or half byte. Our quadruplet is therefore of the form (X_1, X_2, X_3, X_4); where each X_i values is a half byte. Now, deleting commas in between the X_i gives a sixteen-bit aggregation which converted into decimal is an unsigned integer. The unsigned integer obtained from the above transformation is then the final representation of a letter.

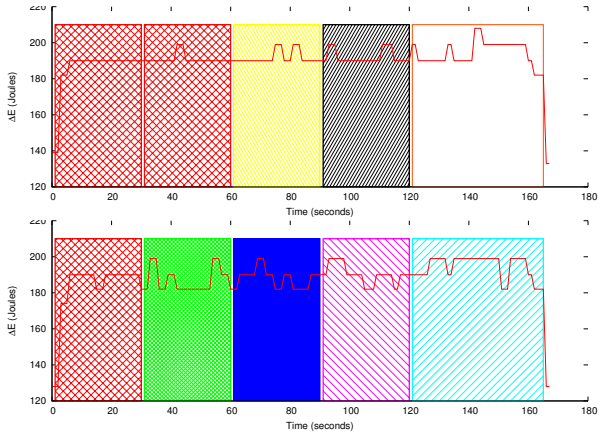

Fig. 1. Example of phase detection on a two-nodes cluster system running the NAS LU benchmark (from top to bottom, first and second computing node).

3.2 System Model Use-cases

This section proposes two simple use-cases of our system modeling approach. We first show how the system's energy profile can be derived from its model. Next, we present an approach to predict the energy consumption of an application using our system model and partial execution.

Energy Profiling The concept of energy profile is very well defined in fields such as physical chemistry, but in computer science it is not easy to find an unanimous definition of this concept. The following definitions can be considered: For an application, the energy profile (respectively power profile) can be roughly defined as its energy (respectively power) footprint during its execution. This definition assumes that the application under consideration is the only application running on the system. A more complete definition presents an application's energy profile as its part of the energy footprint variation of the overall system during its execution.

In studies such as [5, 11, 4, 16, 3, 10, 18] efforts have been devoted to model power usage of system components (CPU, memory, disk, network) via hardware performance counters or hardware monitoring events. From those studies, the energy profile of an application can also be defined as a set of measurements (hardware monitoring events collected per-process or system-wide) over a time interval T such that there exists a combination of them estimating the power/energy used by the application or the whole system over T. The last definition describes the energy consumed by the application or the whole system via a set of hardware monitoring counters. This is particularly interesting as hardware monitoring counters in addition to providing insight about the system's (application's) computational state give way to compute its power consumption.

According to the hardware events-oriented definition of the energy profile, our system model implicitly defines its energy profile. For example, let's consider the energy profile in Figure 1 (a), which can be written as a sequence of the form $L_i \ \ldots \ X_i \ \ldots \ L_k$ denoted as its DNA-like structure; where each system phase is replaced by a letter L_i or XXX which refers to any system configuration or state we do not have enough information about (XXX may typically represent an idle state of the system). It is obvious that the system's energy profile is easy to obtain considering the application, for each known letter is associated to a specific power consumption.

Cross Platform Energy Consumption Prediction This work is probably the very first attempt to predict energy consumption of an application on a given platform. The design of an energy prediction model is mainly motivated by the idea that users often have more than one candidate platforms for running their jobs, therefore choosing the less energy consuming platform can be beneficial to both users and platform providers.

Key concepts of our prediction model include *DNA-like structure of the application*, and *relative energy*. To simplify, we assume that the application under consideration is the only application running on the system, so the system's DNA-like structure is that of the application.

The model implicitly uses two sets of data, one from a reference platform provided by the DNA-like structure of the application, and one from a target platform which is the platform on which we want to estimate the overall energy consumption of the application. The aforementioned reference platform is the platform on which the pattern (DNA-like structure) matching with the application currently running was found.

As mentioned earlier, an application's run matches with a DNA-like structure if a significant percentage of the DNA-like structure matches with the already executed part of the application. Let E_{tar} be the energy consumed by the already executed part of the application and E_{ref} the energy that the matched part of the DNA-like structure had consumed on the reference platform. Denoted as E_{rel}, the relative energy consumption between the two platforms is given by the following: $E_{rel} = \frac{E_{tar}}{E_{ref}}$

With the above relative energy, the estimated energy consumption of the application is given by:

$E_{est} = \int_0^{X\%} P(t)_{i,tar}\, dt + E_{rel} * \int_{X\%}^{end} P'(t)_{j,ref}\, dt$

Where E_{est} is the estimated energy consumption on the target platform; E_{rel} the relative energy consumption between the target platform and the reference platform. In the above equation, $\int_0^{X\%} P(t)_{i,tar}\, dt$ represents the energy consumed by the application before a match is found with a pattern from the profile database. Either measured or estimated, $P(t)_{i,tar}$ is the instantaneous power usage of the application. $P'(t)_{j,ref}$ is the instantaneous power usage of the application on the reference platform and can be obtained from its DNA-like structure.

We define the estimation accuracy as the ratio between the estimated and measured energy on the target platform.

4 Experimental Results: Model Development and Validation

We design a cluster composed of 2 nodes with 2 Intel Quad-core Xeon CPUs each (16 cores in total), 12 GB of RAM. The nodes are connected by a Gigabit Ethernet switch. The Linux kernel 3.1.1 is installed on each node where perf event is used to read the hardware monitoring counters. MPICH is used as MPI library. MG, LU, IS, CG, EP, SP, and BT from NPB-3.3 are used for the experiments. MG demonstrates the capabilities of a very simple multi-grid solver, MG nodes communicate with their neighbors, and with other "distant" nodes. IS calculates the large-scale integer sorting. The CG kernel communication graph involves lots of neighbor communication. EP is embarrassingly parallel code that implements the random-number generator. Each application has several types of computations, memory access, and communication phases. Class C of these benchmarks are used (they are compiled using the default compiler's options). Each node's power usage is monitored with one sample per second using a power distribution unit.

4.1 Idle System State

We conduct experiments to observe the degree to which diverse hardware events affect the system's idle power consumption. Results show that they are almost never the same from a time interval T to another and that the system's idle power cannot be correlated with hardware events. Hence, to characterize the system's idle state, we monitor it again under different workload characteristics. The number of cache misses are chosen as the main system characteristic.

As shown in Figure 2, the number of cache misses normalized to the elapsed cycles count (yielding the event rate) clearly discriminates among all workloads (each scatter of points represents one NAS banchmark). Based on this observation, we define a threshold β under which the system is said to be idle. We assign the letter XXX to the system's idle state to reflect the fact that we are unable to estimate its power usage via hardware performance counters.

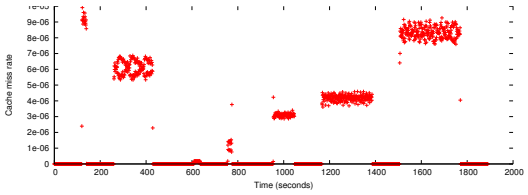

Fig. 2. NAS benchmarks characterized by their cache miss rate.

4.2 Computational States

Firstly, we execute each of the above listed benchmarks 30 times and record the number of occurrences of each performance counter in Table 1. These performance counters are recorded every second along with the power consumption of each node of our cluster. Next, we run Algorithm 1 for defining regions for each node in order to create the system description alphabet. Once the alphabet of each node computed, we use the run history to create a transition matrix for each node. The transition matrix typically gives the probability to move from one letter to another.

As evaluation, we first investigate how close to reality is our approach for partitioning an application into different computational behaviors (or simply phases). For this purpose, we run successively two applications opposite from their computational point (i.e. IS and EP). IS is communication intensive whereas EP is mainly computing. Figure 3 shows the effectiveness of our approach. The couple of integers appearing in each region of the figure gives for each region the corresponding letter coded as an unsigned integer and its duration. For example, (17692 21) means the hardware events highlighted in bold in Table 1 are relevant to power consumption estimation over 21 seconds for the second computing node. We observe from the experiments that two phases were detected and identified with different letters for the two nodes nodes, and that their energy consumptions are different.

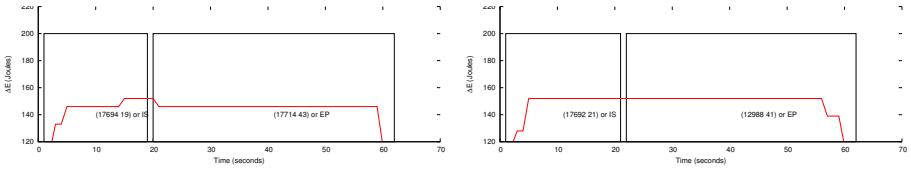

Fig. 3. Dividing NAS benchmarks IS and EP ran successively into two different computational behaviors using their power usage and hardware counters (left: First node; right: Second node).

Table 1. List of hardware monitoring counters

PERF_COUNT_HW_CPU_CYCLES	**PERF_COUNT_HW_INSTRUCTIONS**
PERF_COUNT_HW_CACHE_REFERENCES	PERF_COUNT_HW_STALLED_CYCLES_FRONTEND
PERF_COUNT_HW_CACHE_MISSES	PERF_COUNT_HW_BUS_CYCLES
PERF_COUNT_HW_BRANCH_INSTRUCTIONS	**PERF_COUNT_HW_BRANCH_MISSES**
netSENTbyte	**netSENTpkt**
netRECVbyte	netRECVpkt
Write I/O	Read I/O

Use Case Evaluation: Energy Profile The goal of our approach is to use the energy profile to predict the energy consumption and optimize the resources used. So, if we are able to predict the next system configuration, it will no

longer be necessary to instrument the application source code prior to applying power saving schemes. For example if the predictor tells that the next state is a communication phase with an acceptable error, then power-saving schemes such as DVFS could be triggered. To observe to what degree our system model fits that expectation, we follow this generic methodology:

- Run the application and create its alphabet using Algorithm 1. We assume that all letters have the same duration of fifteen seconds, i.e., each letter covers fifteen seconds of the application's lifetime.
- Take the first two letters of the same application and attempt to determine the rest of the sequence using the system transition matrix.

We discuss the results for LU because it runs a long time. The DNA-like structure (or description sequence) for one node running LU benchmark is provided in Figure 4, where durations are omitted (all 15s). We apply our predictor to the second letter of the LU sequence (293,15) to determine its energy profile or the rest of the sequence (recall that an application's energy profile is derived from its DNA-like structure or description sequence). The predictor simply looks at the transition matrix and chooses the next state with the highest probability. Using this approach, we are only able to predict a few parts of the entire application – less than 5 letters in this case. This can be attributed to the simplicity of the predictor. We are currently investigating more complex approaches to enhance our prediction mechanism.

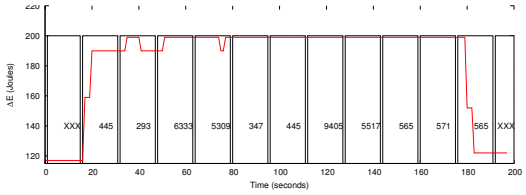

Fig. 4. DNA-like representation with fixed letter duration for a node running the LU benchmark.

Evaluation for Energy Consumption Prediction We evaluate our model for energy consumption prediction considering a very stable application. For simplicity, we only predict the energy consumption of one node. The application iteratively does two things: It first computes the inverse of a 10x10 square matrix and then it copies a large file from a remote host. We choose such an application to ensure that its behavior is stable enough over time. The advantage of having a stable application resides in the fact that we can successfully predict the entire execution of the application.

We consider two scenarios. For the first scenario, the target and reference platforms are the same node but running at respectively 1.6GHz and 2.13GHz. For the second scenario, the reference platform is a Dell Power Edge server and the target platform a Sun Fire V20z. For partial execution, 20% of the

application is executed. Results are summarized in Table 2. We compute for both scenarios the expected energy consumption based on the equations given in Section 3.2. We can see from those results that the accuracy is very good. Notice that the accuracy is higher because it is computed considering the average energy consumption instead of the peak energy consumption. However it must be noted that the considered application is very simple and stable.

Table 2. Energy consumption prediction results summary

Scenario	Estimated Energy (in Joules)	Accuracy	Peak energy consumption
1	797143.5	1.01	811932
2	2068515.6	1.02	2088515.6

5 Conclusions and Future Work

In this paper we propose to optimize large-scale systems energy consumption by triggering specific actions to reduce energy consumption depending on the behavior of the application being executed. We introduce a system modeling approach and some use cases. as a state graph and present some relevant use cases of the model. Our approach for dividing applications into phases is fast and does not require extensive knowledge of applications running on the system. We show through experiments that our model can effectively be used for predicting the system's energy profile. We also show that energy consumption estimation using our system model and partial execution can successfully predict the entire system power usage.

Future work will combine our model with power saving schemes. We will also improve the system model using more sophisticated mechanisms in order to guarantee accurate energy profile prediction. We also plan on using our energy consumption prediction model for predicting the energy consumption of real workloads on multi-nodes systems

Acknowledgments This work is supported by the INRIA large scale initiative Hemera focused on "developing large scale parallel and distributed experiments".

References

1. M. Annavaram, E. Grochowski, and J. P. Shen. Mitigating amdahl's law through epi throttling. In *ISCA*, pages 298–309. IEEE Computer Society, 2005.
2. D. H. Bailey, E. Barszcz, J. T. Barton, R. L. Carter, T. A. Lasinski, D. S. Browning, L. Dagum, R. A. Fatoohi, P. O. Frederickson, and R. S. Schreiber. The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 5:63–73, 1991.
3. D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. A simple power-aware scheduling for multicore systems when running real-time applications. In *IPDPS*, pages 1–7. IEEE, 2008.

4. G. Contreras. Power prediction for intel xscale processors using performance monitoring unit events. In *In Proceedings of the International symposium on Low power electronics and design (ISLPED*, pages 221–226. ACM Press, 2005.
5. G. D. Costa and H. Hlavacs. Methodology of measurement for energy consumption of applications. In *GRID*, pages 290–297. IEEE, 2010.
6. M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In G. K. Egan and Y. Muraoka, editors, *ICS*, pages 157–166. ACM, 2006.
7. A. S. Dhodapkar and J. E. Smith. Managing multi-configurable hardware via dynamic working set analysis. In *In 29th Annual International Symposium on Computer Architecture*, pages 233–244, 2002.
8. V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch. Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. *J. Parallel Distrib. Comput.*, 68(9):1175–1185, 2008.
9. R. Ge, X. Feng, and K. W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society.
10. R. Joseph, M. Martonosidepartment, and E. Engineering. Run-time power estimation in high performance microprocessors. In *In International Symposium on Low Power Electronics and Design*, pages 135–140, 2001.
11. I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE '01, pages 28–31, New York, NY, USA, 2001. ACM.
12. A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36:26–31, August 2008.
13. H. Kimura, T. Imada, and M. Sato. Runtime energy adaptation with low-impact instrumented code in a power-scalable cluster system. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 378–387, Washington, DC, USA, 2010. IEEE Computer Society.
14. 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*, SC '06, New York, NY, USA, 2006. ACM.
15. C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron. Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems. *Computer Science - Research and Development*, pages 1–9, aug 2011.
16. K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37:46–55, July 2009.
17. A. Weissel and F. Bellosa. Process cruise control-event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002)*, Grenoble, France, 2002.
18. W. Wu, L. Jin, J. Yang, P. Liu, and S. X. D. Tan. A systematic method for functional unit power estimation in microprocessors. In *Design Automation Conference*, 2006.