CHAPTER 1

ENERGY AWARE APPROACH FOR HPC SYSTEMS

ROBERT BASMADJIAN[1], GEORGES DA COSTA[2], GHISLAIN LANDRY TSAFACK CHETSA[23], LAURENT LEFEVRE[3], ARIEL OLIEKSIAK[4], JEAN-MARC PIERSON[2]

[1]Passau University
[2]IRIT, Toulouse University
[3]INRIA, LIP Laboratory, Ecole Normale Superieure of Lyon
[4]Poznan Supercomputing and Networking Center

1.1 Introduction

With the *race to exascale* one of the major concern for actors involved in the development and operation of HPC systems is no longer the number of PFlops (petaflops) their system can achieve per second, but how many PFlops they can achieve per Watt. This novel fashion of evaluating supercomputers' performance place a great emphasis on their energy consumption. The emphasis on energy consumption can be justified by the fact that computer chips seem to have hit a wall, meaning that we can't make them go any faster. Consequently, supercomputer designers just have to add more chips to increase computing power. But this approach has a significative impact on energy usage.

However, tremendous efforts are being undertaken by HPC operators from multiple levels to make supercomputers greener. This is evidenced by the Green500 list[1];

[1]http://green500.org

its latest issue shows that the recent supercomputers are getting greener. The rise of graphics processors in massive server clusters and the acquisition of low power memories are probably the main reason of their sudden improvement in energy efficiency. Just to give a global picture, in 2009 Samsung claimed that more that 95TWh/year or $6B to $7B/year could potentially be saved if the memory in all 32Mu servers worldwide could be replaced with their Samsung Green DDR3 memory chip.

Similar efforts are being carried out regarding all other HPC subsystems from the processor to the network to the storage subsystems. However, significant efforts still need to be made if today's supercomputers want to meet the 20MW constraint for exascale.

Several directions are available to improve energy-efficiency of HPC systems with software:

- New and fitted programming models;
- Smart failure management for applications;
- Optimized data management;
- Improved runtimes;

There is a common belief that a considerable share of energy consumed by HPC systems during their operations could be potentially saved if user applications were programmed differently. Put another way, throughout their life cycle, user applications exhibit behaviours whose understanding allows implementing power reduction schemes which can significantly reduce the amount of energy they consume at runtime. This has been proven true by the literature [**?**, **?**, **?**, **?**, **?**, **?**].

From what precedes, making HPC applications more energy friendly requires: (i) rewriting the majority of existing applications with energy constraints in mind; (ii) and that new applications be coded with the same constraints. These alternatives may not always be possible. Rewriting some HPC applications is so costly that most people find paying the electrical bill worth (there is no evidence; however this issue has been in people's mind for a while, but to our knowledge no one has proposed an energy efficient version of an HPC application so far) whereas application developers usually don't pay much attention to how much energy their applications will consume. The main reason to this is that power saving schemes are platform specific. For example, let's consider the dynamic voltage and frequency scaling (DVFS) technology which allows scaling the processor's frequency according to the workload in some cases. So integrating DVFS into a program source code assumes that the developers know all the potential platforms that will run their applications which doesn't make sense.

The best place to improve energy-efficiency is at the runtime level. Contrary to application-level improvements, this has the potential to touch large classes of applications, some of which the source code is not accessible, on a large variety of hardware.

To achieve an energy-efficient runtime, a fine grained knowledge on hardware and application is necessary. This knowledge is necessary in order for the runtime to evaluate the possible gain of applying leverages.

Server power consumption is complex to evaluate and measure[**?**]. Servers are build of multiple sub-components of which the power consumption model is often non-linear. Servers are also in complex environment. During conception phase, care is also given to other power consuming elements such as the cooling, lights, network,... Cooling can consume up to the same power as the computing infrastructure. Usually cooling consumes between 20 and 30% of the computing infrastructure. This chapter addresses software modifications during the life of the HPC infrastructure, not during design, so cooling will be out of scope of this document.

Applications are also complex structures. Alongside with the spatial structure of hardware, applications have a temporal structure. Runtime must take into account this structure to make decision adapted to each phase of applications.

In the following, we will first provide a detailed explanation of servers power consumption in Section **??**, then Section **??** will describe application and phase detection and identification. Finally Section **??** will show available leverages in HPC systems.

1.2 Power consumption of servers

In [**?**], Rivoire establishes a comparison list between many different energy models. Two categories are defined: comprehensive models and high-level models called *black-box models*. Comprehensive CPU models were proposed in [**?**] and [**?**]. These models are very accurate in terms of CPU power consumption and rely on specific details of the CPU micro-architecture. High-level models are more general, and do not take into account specific low-level details of the operation and characteristics of the modelled infrastructure. Nevertheless, it increases portability of the model and simulation speed. These simple models sacrifice the accuracy of computation, but do not require to have a very detailed knowledge of the architecture of processors used.

In Section **??**, we introduce the most relevant energy-related attributes of a server in the form of UML class diagram. Then based on these attributes, in Section **??** we give the power consumption prediction models.

1.2.1 Server modeling

Figure **??** demonstrates the UML class diagram of servers depicting their most relevant energy-related dynamic and static attributes. The former denotes the fact that the attribute needs to be kept up-to-date by monitoring systems. By static attributes we mean those whose value remains constant; most of the time, the value of static attributes are obtained from the manufacturers' data sheet.

The `Server` class represents an abstraction for a generic server computing system, such that the different subclasses (e.g. `Tower Server`, `Rackable Server`, and `Blade Server` classes) are distinguished by their physical form factor. The attribute *name* is used to provide a unique identifier for servers, whereas *status* describes its dynamic status, which can be either ON or OFF.

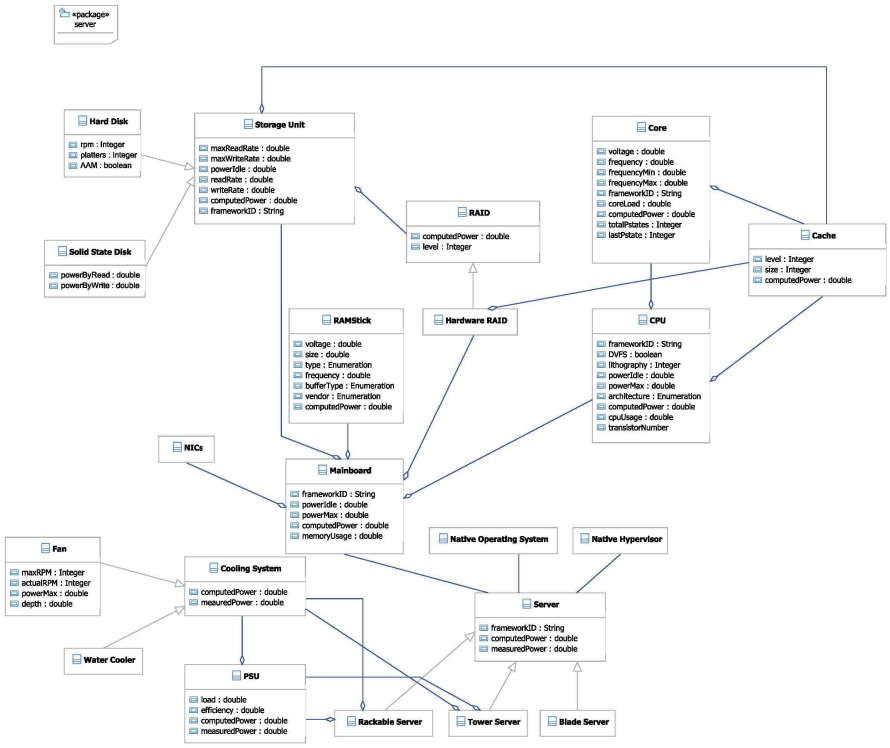

Figure 1.1 Server UML class diagram

Typically, a server has a `Mainboard` and runs baseline software components (e.g. `Native Operating System`). The mainboard provides most of the core hardware components of the system: Central Processing Units (`CPU` class), Random Access Memories (`RAMStick`), Network Interface Cards (`NICs`), Hardware RAIDs (`HardwareRAID`) and Storage Units (`HardDisk`). Its *memoryUsage* attribute denotes the total usage (in GB) of all the attached memory modules.

Since the advent of modern processors, a `CPU` is commonly composed of more than one `Core`. Each core can have its own `Cache`, depending on the cache *level* (e.g. Level 1). It is also possible that certain cores share the same cache (e.g. Level 3). The most relevant energy-related attributes of the `CPU` class are:

1. *architecture*: indicates the processor's vendor (e.g. Intel, AMD, etc.); it's relevant, since it translates into different power consumption behaviors.

2. *cpuUsage*: gives the utilization rate (load) of the processor.

3. *DVFS* (Dynamic Voltage and Frequency Scaling): used to indicate whether energy-saving mechanisms (e.g. Intel SpeedStep) are enabled or not.

4. *lithography*: denotes the size of the processor in nanometres.

5. *transistorNumber*: presents the number of transistors expressed in the order of millions.

Each `Core` operates at a certain *frequency* (in GHz) and *voltage* (in Volt). *coreLoad* expresses the utilization rate of the corresponding core. The `RAMStick` class has several attributes relevant to power consumption estimation:

1. *voltage*: indicates the supply voltage with which the memory module operates; it is highly dependent on the memory *type* (e.g. DDR_3).

2. *size*: denotes the size of the memory module (in GB).

3. *frequency*: shows the memory module's operational frequency (in MHz).

4. *vendor*: presents the manufacturer (e.g. KINGSTON, HYNIX, etc.).

5. *bufferType*: type of buffer technology (e.g. fully buffered).

`Hard Disk` class' energy-related attributes are the following: *maxReadRate* and *maxWriteRate* indicate respectively the maximum read and write rates of the disk which are computed in terms of transferred data size per second (MB/s). *readRate* and *writeRate* indicate respectively the actual read and write rates of the disk which are expressed in terms of MB/s. *powerIdle* is the power consumed by the hard disk when it is inactive.

`Tower Servers` and `Rackable Servers` are equipped with their own `PSUs` and `Fans`. The most relevant energy-related attribute of a `PSU` is the *efficiency*, which indicates (in percentage) the amount of loss for the power supplied to server components. Inside the `Fan` class, *depth* denotes the depth (in meter), whereas *maxRPM* and *powerMax* indicate respectively the maximum nominal values of rotations per minute and power consumption of the fan. *actualRPM* shows the actual instantaneous rotation speed of the fan.

1.2.2 Power prediction models

In this section, we first introduce the idle power consumption of a server and then present its dynamic power.

1.2.2.1 Idle power consumption Basically, the idle power of a server does not fluctuate significantly (e.g. ± 1 Watt) and depends strongly on its configured hardware (e.g. number of processors, memory modules). Over the last years, there have been several efforts to reduce the idle power consumption of servers by means of several software solutions (e.g. C-states for processors) in combination with the hardware.

1.2.2.1.1 Processor Based on Joule's and Ohm's laws [**?**], the power consumption of a processor can be expressed by:

$$P = I * V, \tag{1.1}$$

where P denotes the power (Watt), I represents the electric current (Ampere) and V indicates the voltage (Volt).

With the advent of multi-core processors, a processor is in idle state when all of its constituent cores are also inactive. Consequently, Equation (**??**) can be considered to present the idle power of each core:

$$P_i = I_i * V_i, \tag{1.2}$$

Furthermore, the idle power consumption of each core i depends on its number of transistors. Hence, Equation (**??**) can also be adopted to express the idle power of transistors (in the order of million):

$$P_{ji} = I_{ji} * V_{ji}, \tag{1.3}$$

where I_{ji} and V_{ji} denote respectively the current and voltage of the j^{th} transistor of the core i.

Basically, the processors' operating voltage is between 0 and 2V. By studying the characteristic of voltage-current relationship within the above mentioned range, the author of [**?**] showed that this relationship is almost linear. Hence, the current leakage I_{ji} of transistors is modeled using the curve-fitting methodology by means of a second order polynomial:

$$I_{ji} = \alpha V_{ji}^2 - \beta V_{ji} + \gamma, \tag{1.4}$$

where $\alpha = 0.114312$, $\beta = 0.22835$ and $\gamma = 0.139204$ are the coefficients.

Let t_i denote the total number of transistors (in the order of millions) of a core i, then its power consumption is given by:

$$P_i = \sum_{j=1}^{t_i} I_{ji} * V_{ji}. \tag{1.5}$$

Recently, several energy-saving mechanisms (e.g. Intel SpeedStep [**?**] and AMD Cool'n'Quiet [**?**]) have been emerged in order to reduce the power consumption of processors. To model such an impact, the following equation is used based on Equation (**??**):

$$Pr_i = \delta_i * P_i, \tag{1.6}$$

where δ_i is the reduction factor in the power consumption P_i of core i, whereas Pr_i represents the reduced power consumption of a core i. Note that δ_i can vary depending upon the corresponding energy-saving mechanisms, where each of such mechanism has its own particular behavior. For different modeling of δ_i, interested readers can refer to [**?**]. However, it is worth pointing out that for AMD processors

both frequency and voltage scaling play a major role in reducing the power consumption whereas for Intel processors only the voltage is an important factor. Hence, the idle power consumption of a multi-core processor having n cores is the sum of the power of each of its constituent cores:

$$P_{CPU} = \sum_{i=1}^{n} Pr_i \tag{1.7}$$

1.2.2.1.2 Memory Similar to the processor, memory's power consumption can be expressed as:

$$P = I * V. \tag{1.8}$$

As mentioned previously, it was shown in [**?**] that there is a linear relationship between the supplied voltage V (between 0 and 2V) and current I. Since the modern memory module technologies (e.g. DDR_2, DDR_3) operate within the above mentioned range, then the current I can be given by:

$$I = c * V, \tag{1.9}$$

where c is a constant and takes a value of 0.00043 and 0.00013 for DDR_2 and DDR_3 respectively. The idle power consumption of a memory module for a given *frequency* f (MHz) and size s (GB) can be expressed as:

$$P(f, s) = c * V^2. \tag{1.10}$$

In order to reflect the impact of frequency as well as the size of a memory module, Equation (**??**) can be rewritten as:

$$P = s * f * c * V^2. \tag{1.11}$$

Given a set of n memory modules, their idle power consumption is given by:

$$P_{RAM} = \sum_{i=1}^{n} s_i * f_i * c * V_i^2, \tag{1.12}$$

where s_i, f_i and V_i denote respectively the size, frequency and the voltage of a specific memory module i, whereas c is a constant whose value is given above.

1.2.2.1.3 Hard Disk The hard disk's *idle mode* (e.g. no operation) consists of the following three states[2]: *idle*, *standby* and *sleep*. Furthermore, the power consumption of standby and sleep states is quite identical and it is in average 10% of the idle state power consumption. The main reason for such a behavior is that during standby and sleep states, the disk's mechanical parts are stopped. The *idle mode* power consumption of the hard disk is given by:

$$P_{HDD} = y * P_{idle}(\alpha + 0.1 * \beta + 0.1 * \gamma), \tag{1.13}$$

[2]The hard disk changes its state sequentially from idle to standby and then to sleep.

where $y \in [0, 1]$ denotes the probability that the disk is in idle mode whereas P_{idle} is the idle state power consumption provided by the manufacturer's data sheet. The parameters $\alpha, \beta, \gamma \in [0, 1]$ indicate respectively the probability that the disk is in idle, standby and sleep states. To derive values for α, β and γ, such that $\alpha + \beta + \gamma = 1$, the following probabilistic approach can be adopted:

1. If $0 < y$ [3] ≤ 0.3, then $\alpha = 0.9$, and $\beta = \gamma = 0.05$.
2. If $0.3 < y \leq 0.6$, then $\alpha = 0.5$ and $\beta = \gamma = 0.25$.
3. If $0.6 < y \leq 1$, then $\alpha = 0.1$ and $\beta = \gamma = 0.45$.

From above equations, one can observe that the more the disk is in idle mode (i.e. $y \simeq 1$), the higher is the probability that it will remain in standby and sleep states. Note that whenever the state transition of the disk is accurately detected, then the parameters α, β, γ can be configured appropriately such that always two of such parameters have a value of zero.

1.2.2.1.4 Mainboard The idle power consumption of the mainboard is the sum of the power consumptions of its constituent components:

$$P_{Mainboard} = \sum_{i=1}^{l} P_{CPU} + P_{RAM} + \sum_{j=1}^{m} P_{NIC} + \sum_{k=1}^{n} P_{HDD} + c, \qquad (1.14)$$

where P_{CPU}, P_{RAM}, and P_{HDD} are given respectively by Equations (**??**), (**??**), and (**??**), whereas c is a constant related to the mainboard's own power draw. Thus, the value of c can be configured through *powerIdle* attribute of the `Mainboard` class either from the manufacturer's data sheet or by means of observations. Finally, P_{NIC} denotes the idle power consumption of the network interface cards whose value can be found in the manufacturer's specifications.

1.2.2.2 Dynamic Power Consumption Unlike the idle power of servers, the dynamic one fluctuates significantly based on the running applications (e.g. jobs) and their workloads. As for the idle power, there have been several efforts to reduce the dynamic power consumption of servers by means of energy-saving mechanisms (e.g. DVFS) in combination with the hardware.

1.2.2.2.1 Processor Based on CMOS circuits [**?**] power draw, the dynamic power consumption of a core i is given by the following utilization-based model:

$$P'_i = V_i^2 * f_i * C_{eff} * \frac{L_i}{100}, \qquad (1.15)$$

where P'_i denotes the dynamic power (Watt) of a core i having a utilization L_i, V_i and f_i indicate respectively the voltage and frequency of the corresponding core i, whereas C_{eff} presents the effective capacitance.

[3] Values of y can be found in Section **??**.

Given a processor of n cores with no specific energy-saving mechanisms enabled, its dynamic power consumption is given by:

$$P'_{CPU} = \sum_{i=1}^{n} P'_i, \tag{1.16}$$

In [**?**], the authors showed that the power consumption of a multi-core processor is not the pure sum of that of its constituent cores as illustrated in Equation (**??**). Consequently, they decomposed a multi-core processor into three component levels: (1) chip, (2) die and (3) core level, and modeled the power consumption of each of the corresponding component. Furthermore, they showed that Equation (**??**) always overestimates the power consumption and proposed a model that takes into account both resource sharing and energy-saving mechanisms. Readers interested in the detailed modeling can refer to [**?**].

1.2.2.2.2 Memory Concerning the dynamic power consumption due to memory access, there is always only one active operating rank per channel regardless of the number of memory modules or module ranks in the system. As a matter of fact, such a power is always constant during access and independent of the operation type (e.g. read or write) and size:

$$P'_{RAM} = \gamma * \beta, \tag{1.17}$$

where $\beta = 7$ W, 17 W, and 10 W for DDR_2 unbuffered, DDR_2 fully buffered and DDR_3 unbuffered memory modules respectively, whereas $\gamma \in [0, 1]$ denotes the probability that a memory access is performed. Such a parameter is useful for systems when the application profile of accessing the memory module is not known in advance. The following technique can be adopted to set values for γ:

1. If the processor is in idle state performing no activity, then it can be equally assumed that the memory modules are also in idle state: i.e. $\gamma = 0$.

2. If the processor is carrying out some computational activities (utilization of more than 1%), then a probabilistic approach can be adopted in modeling γ, such that the more total memory is in use, the higher the probability that a memory access is performed:

$$\gamma = \frac{memoryUsage}{\sum_{i=1}^{n} s_i},$$

such that s_i and n are defined in Equation (**??**), whereas $memoryUsage$ is introduced in Section **??**.

1.2.2.2.3 Hard Disk It was shown in [**?**] that the power to perform read or write operations on hard disk drive is in average 1.4 times more than the idle state power consumption, whereas the startup power consumes in average 3.7 times more than the one for idle state:

$$P'_{HDD} = x * 1.4 * P_{idle} + z * 3.7 * P_{idle}, \tag{1.18}$$

where $x, z \in [0, 1]$ denote respectively the probability that the disk is in accessing and startup modes respectively such that $x + y + z = 1$, whereas P_{idle} and y are introduced in Section **??**. The following technique can be adopted in order to set values for x, y and z:

1. If the average operation size (MB/s) of reads and writes per second is zero (i.e. *readRate* or *writeRate*=0), then it can be assumed that the disk is in idle mode ($x = z = 0$ and $y = 1$).
2. If the average number of read/write operations per second is not zero, we adopt a probabilistic approach in modeling the mode changes such that:
   - If *readRate or writeRate* > 0, then $x = \frac{readRate+writeRate}{maxReadRate+maxWriteRate}$,
   - If *writeRate* $= 0$, then $x = \frac{readRate}{maxReadRate}$,
   - If *readRate* $= 0$, then $x = \frac{writeRate}{maxWriteRate}$,

whereas $y = 0.9 * (1 - x)$ and $z = 0.1 * (1 - x)$.

1.2.2.2.4 Mainboard As for the idle part, the dynamic power consumption of the mainboard is given by:

$$P'_{Mainboard} = \sum_{i=1}^{l} P'_{CPU} + P'_{RAM} + \sum_{j=1}^{m} P'_{NIC} + \sum_{k=1}^{n} P'_{HDD}, \tag{1.19}$$

where P'_{CPU}, P'_{RAM}, and P'_{HDD} are given respectively by Equations (**??**), (**??**), and (**??**), whereas P'_{NIC} denotes the dynamic power consumption of the network interface card whose value is at most 5% of the idle power consumption.

1.2.2.3 Total Power Consumption

1.2.2.3.1 Fan Based on [**?**], the power consumption of a fan is expressed as:

$$P = d_p * q, \tag{1.20}$$

where P represents the power consumption (Watt), d_p indicates the total pressure increase in the fan (Pa or N/m^2), and q denotes the air volume flow delivered by the fan (m^3/s). Consequently, Equation (**??**) can be rewritten as:

$$P = \frac{F}{A} * \frac{V}{t}, \tag{1.21}$$

where F, A, V and t denote respectively the force (N), area (m^2), volume (m^3) and time (seconds). Since the ratio volume V to area A presents the depth d (m), then Equation (**??**) can be expressed as:

$$P = \frac{F * d}{t}. \tag{1.22}$$

It was shown in [**?**] that F is proportional to the square of the revolutions per minute (RPM):

$$F = c_{fan} * RPM^2. \tag{1.23}$$

Hence, the power consumption of a fan having a depth d and an actual instantaneous revolution per minute RPM is given by:

$$P_{Fan} = \frac{c_{fan} * RPM^2 * d}{3600}, \tag{1.24}$$

such that c_{fan} is a constant for a given fan that can be computed as:

$$c_{fan} = \frac{3600 * P_{max}}{RPM_{max}^2 * d}, \tag{1.25}$$

where P_{max} and RPM_{max} denote respectively the maximum power consumption and rotations per minute of a fan whose values can be extracted, in addition to the depth d, from the manufacturer's data sheet.

1.2.2.3.2 Power Supply Unit The parameter that plays the major role in the power consumption of a PSU is its *efficiency*. To this end, the PSU manufacturers provide the efficiency range with respect to a given load. The power consumption of a PSU having an *efficiency* of e (in percentage) is:

$$P_{PSU} = (100 - e)\frac{P_{Mainboard} + P'_{Mainboard} + P_{Fan}}{n * e}, \tag{1.26}$$

such that $P_{Mainboard}$, $P'_{Mainboard}$ and P_{Fan} are introduced in Equations (**??**), (**??**) and (**??**) respectively, whereas n denotes the number of PSUs and e their efficiency (assuming that it's identical for all the installed PSUs).

1.2.2.3.3 Total Power Given a server composed of a mainboard, fans and power supply units as depicted in Figure **??**, then:

1. For *Blade* type servers, the power consumption is:

$$P_{Blade} = P_{Mainboard} + P'_{Mainboard}. \tag{1.27}$$

2. For *Tower* or *Rackable* type servers, the power consumption is given by:

$$P_{Tower\_Rackable} = P_{Mainboard} + P'_{Mainboard} + \sum_{i=1}^{l} P_{Fan} + \sum_{j=1}^{m} P_{PSU}, \tag{1.28}$$

such that $P_{Mainboard}$, $P'_{Mainboard}$, P_{Fan} and P_{PSU} are respectively given by Equations (**??**), (**??**), (**??**), and (**??**).

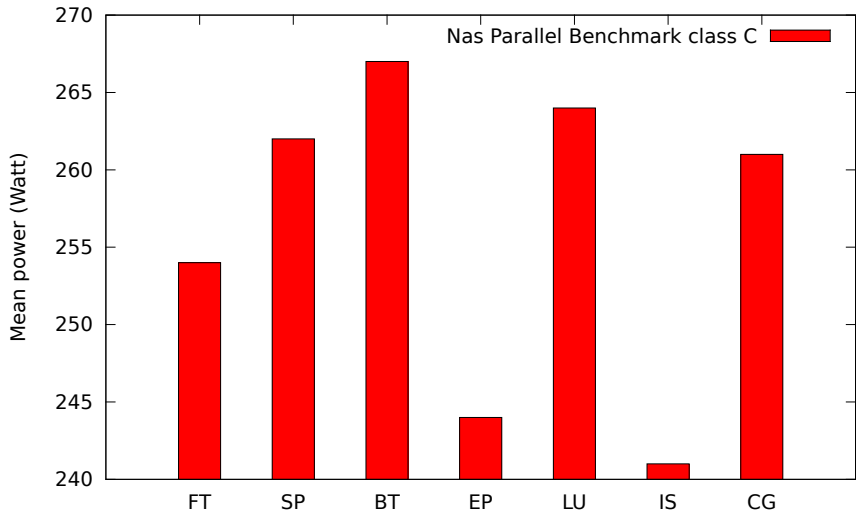

Figure 1.2 Mean power consumption of Nas Parallel Benchmark on Sun Fire X2200 M2 with AMD Opteron 2218 processors.

1.3 Classification and energy profiles of HPC applications

In the HPC world, the details of applications are often discarded. Most application are considered to be identical and to use 100% of the servers resources. Actually, it is far from being the case.

For instance in Figure **??**, all the benchmarks run with 100% load. However their power profiles are different. This suggest that having the load and memory access profiles is insufficient for an effective evaluation of the power consumed by an application. For example, IS and FT have the same average resource utilisation for the CPU and the memory, but there is a difference of 16W in their power consumption on the same hardware. The two just mentioned applications (IS and FT) from Nas Parallel Benchmark (NPB) suite have different patterns of access to the memory and the network subsystems. We believe the difference in their total energy consumption can be attributed to the fact that they have different memory and network profiles.

Depending on the behaviour of the application, several policies are often possible. Nevertheless, similar applications can be treated in the same way. The similarity criterion is often relative and may not always be the same. For example, applications can be said to be similar if they have the same resource utilization pattern (i.e., the same behaviour in terms of resource utilization). Knowing that many scientific applications often exhibit different behaviours (also known as phases) throughout their life cycle, in the following, a phase can be either an application (if it only has a single behaviour) or a specific phase of an application.

As just mentioned, comparing two applications or telling whether they are similar or not can be very difficult. Several authors propose to analyse applications off-line. In [?] authors introduce metrics to characterize parallel applications. Motivated by the need of choosing the appropriate platform for a given application, authors rely upon static code analysis to classify applications. This approach offers the benefit of not being platform dependent; however, does not allow on-line applications/phase classification or phase detection. A similarity-based taxonomy approach is proposed in [?], but authors do not emphasise on how applications are differentiated. Approaches described in [?] and [?] show that parallel applications can be classified in a limited number of classes.

In [?], authors manually analyse the communication patterns of 27 applications from different HPC benchmarks based on MPI communication library. Their purpose was to study the possible deterministic communication patterns in order to exploit them in fault tolerance algorithms. Code analysis is time consuming and does not allow for runtime analysis. However it proves the potential and the value of communication pattern discovery. The authors of [?] proposes a tool for assessing the code quality of HPC applications which turns to static pattern analysis while for instance [?] proposes MAQAO to tune performance of OpenMP codes.

In [?], authors present the Integrated Performance Monitor (IPM). This tool allows for MPI application profiling and workload characterization. It allows for post-mortem analysis of the application behaviour to understand the computation and communication phases. Vampir [?], Tau [?], Sun Studio [?] are other examples of such performance analysis tools. In [?] authors use Periscope to automatically detect memory access patterns, after the program ends. Similarly, Scalasca [?] searches for particular characteristic event sequences in event traces automatically. From low-level event traces, it classifies the behaviour and quantifies the significance of the events by searching for patterns of inefficient behaviours. It relies on a number of layers to create an instrumented code, to collect and measure via measurement libraries linked with the application, to trace the running application to finally analyze a report produced after the run. Authors of [?] use an I/O stress test benchmark, namely IOR, to reproduce and predict I/O access patterns. Analysis of the results shows that a simple testbed can be used for the characterization of more complex applications, but a manual tuning of the benchmark parameters has to be operated, which leads to impractical usage. In [?] authors examined and compared two input/output access pattern classification methods based on learning algorithms. The first approach used a feed-forward neural network previously trained on benchmarks to generate qualitative classifications. The second approach used Markov models trained from previous executions to create a probabilistic model of input/output accesses.

Works done in [?], [?] use on-line techniques to detect applications execution phases, characterize them and accordingly set the appropriate CPU frequency. They rely on hardware monitoring counters to compute runtime statistics such as cache hit/miss ratio memory access counts, retired instructions counts, etc. which are then used for phase detection and characterization. Policies developed in [?], [?] tend to be designed for single task environment. As the infrastructure in HPC systems tends

to be a shared environment, usually the monitoring infrastructure must gather information at different levels: process/application/virtual machine level, and not only at the host level. On-line recognition of communication phases in MPI application was investigated by Lim et al. in [**?**]. Once a communication phase is recognized, authors apply the CPU DVFS to save energy. They intercept and record the sequence of MPI calls during the execution of the program and consider a segment of program code to be reducible if there are high concentrated MPI calls or if an MPI call is long enough. The CPU is then set to run at the appropriate frequency when the reducible region is recognized again.

1.3.1 Phase detection

A classical methodology relies on the concept of execution vector (EV) which is similar to power vectors (PV) in [**?**]. An execution vector is a column vector whose entries are system's metrics including hardware performance counters, network byte sent/received and disk read/write counts. For convenience, those metrics will be referred to as *sensors*. Each sensor related to hardware performance represents the access rate to a specific hardware register over a given time interval, whereas sensors related to the network and the disk monitor network and disk activities respectively. There are several type of sensors related to hardware performance counters, these include: number of instructions, last level cache accesses and misses, branch misses and predictions, etc. The sampling rate corresponding to the time interval after which each sensor is read depends on the granularity. While a larger sampling rate may hide information regarding the system's behaviour, a smaller sampling rate may incur a non negligible overhead. A sampling rate of one second is often a good trade off. Execution vectors are also timestamped with the time at which it was sampled.

By definition, the manhattan distance between two points in an n-dimensional space is the distance between them if a grid-like path is followed. It offers the advantage that it does not depend on the translation of the coordinate system with respect to a coordinate axis, i.e., it weights more heavily differences in each dimension. In order to detect system's phase changes at runtime, the manhattan distance between consecutive execution vectors is used as the resemblance or similarity criterion. This similarity is used to cluster EVs along the execution time-line as follow: two consecutive EVs along the execution time-line belong to the same group or are similar if the manhattan distance between them is bellow fixed similarity threshold. The similarity threshold can be a fixed or varied percentage of the maximum known distance between all consecutive execution vectors.

For example, given a fixed similarity threshold of 10%, two consecutive EVs belong to the same group if the manhattan distance between them is less than 10% of the similarity threshold.

Knowing that the behaviour of the system is relatively stable during a phase (a phase is a time period during which the behaviour of the system is stable) and assuming that this stability is translated into execution vectors sampled during that phase, a phase is defined as any behaviour delimited by two successive manhattan distances exceeding the similarity threshold. Therefore, considering Figure **??** (where the x-

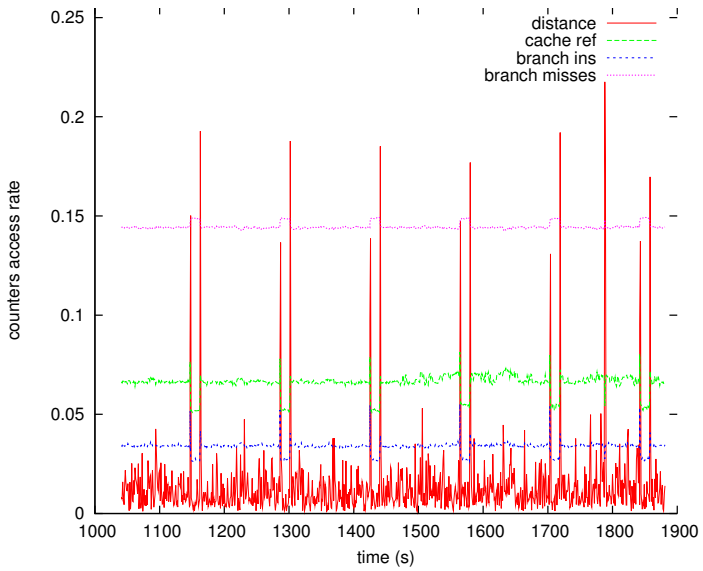

Figure 1.3 Phase identification using the similarity between consecutive execution vector as phase identification metric. This is a zoomed-in view of the traces collected on one node when the system was running Molecular Dynamics Simulation[**?**]; “distance" represents the variation of the distance between consecutive execution vectors.

axis represents the execution time-line) along with a similarity threshold of 50% and assuming that the maximum manhattan distance between two consecutive execution vectors along the execution time-line is 0.2; seven phases separated by six micro-phases or transition phases (0.1 is 50% of 0.2; referring to the definition of a phase, phases are delimited by distances greater than 0.1) can be detected. These phases correspond to variations reported in the access rate of plotted performance counters.

1.3.2 Phase identification

The need to reuse system reconfiguration information for reoccurring phase being essential for many cases (for example to avoid the characterisation overhead). It is therefore not surprising that several methods for phase identification have been developed. The cost associated with phase identification may depend on the methodology employed. Since data collected during a phase are often for large for efficient representation and comparison on the hardware, phases are often represented with only a few items of information. The representation of a phase is usually tight to the phase detection mechanism. Herein, a phase is represented with a special EV known as the *representative vector* of that phase. It is actually the arithmetic average of all EVs collected during that phase.

A phase being represented with a single vector, identifying two phase with each other boils down to comparing their representative vectors. However, the cost as-

sociated with this increases with the number of archived phases, which can lead to performance degradation during runtime.

For phase identification to guide reuse of system reconfiguration information, one can associated each phase with a label which implicitly indicates the subsystem (including the processor, the memory, storage and network) that was mainly used during the phase (For example, a compute intensive phases are likely to stress the processor more that the storage subsystem, in which case the phase can be labelled as compute intensive). In an energy efficient context, this can help determine with subsystem can be switched off to save energy. Referring to HPC workloads, five labels can be defined: *compute intensive*, *memory intensive*, *mixed*, *network intensive* and *I/O intensive*. They are self explanatory with the exception of "mixed". In a few words, workloads/phases labelled as mixed are both memory and compute intensive, which means that they alternate between memory intensive and compute intensive behaviours; however, the granularity at which this occurs is low to the point that they cannot be considered as phases.

Another approach consists of using a decision tree like in [**?**]. In this work, Authors uses a decision tree (Figure **??**. It is a tree-shaped diagram that determines a course of action. Each branch of the decision tree represents a possible phase class. The tree structure shows how one choice leads to the next. The decision tree can be produced by multiple algorithms that identify various ways of splitting a data set into classes. Their representation of acquired knowledge in a tree form is intuitive and easy to understand by humans. What is more, the learning and classification steps of decision tree induction are simple and fast.

These classifications produce classes of applications that share the same resource consumption profiles. Using those classification and models it is possible to estimate the power profile of an application. Taking into account time in addition to power enables evaluating energy consumed by an application. Having an application that consume power at a steady state has a different impact compared to an application that consume power in a random way, even if the two applications have the same total energy and the same length. Their different way to consume energy lead to different starting time of fans, and to different final temperature of elements.

Once complete status is available, at the same time from the hardware and from the application or phase point of view, green leverages can be applied.

1.4 Policies and leverages

There is a large body of work addressing the issue of power consumption in high performance computing (HPC) systems. These work can be roughly divided into off-line and on-line approaches. Off-line approaches necessitating human intervention involve several steps including: source code instrumentation for performance profiling; execution with profiling; determination of the appropriate CPU-frequency for each phase; and source code instrumentation for inserting dynamic voltage and frequency scaling (DVFS) instructions.

Off-line methods usually use a two level approach:

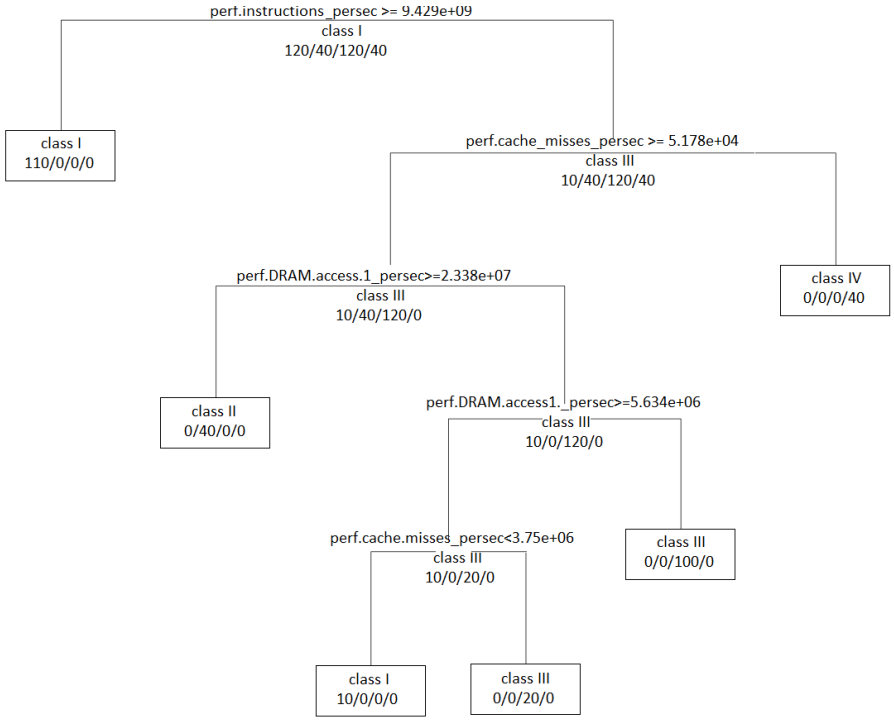

Figure 1.4 Figure 6 Classification Tree for AMD Opteron 275

- Profiled test-runs and/or source code instrumentation;

- Source code modification to add leverages.

For off-line approaches, Freeh *et al.* [**?**] exploit PMPI to time MPI calls to insert DVFS scheduling calls based on duration while Cameron *et la.* [**?**] profile MPI communications to create scenarii that will be followed when running application at full scale. Kimura *et al.* [**?**] instrumented program source code to insert DVFS directives according to the program's behaviour in order to reduce the program's energy consumption without significant performance degradation.

On-line approaches use the methods described in the previous section to detect program execution phases to apply DVFS accordingly. In [**?**, **?**] characterize phases and set the appropriate CPU frequency accordingly. Policies developed in [**?**, **?**] tend to be designed for single task environments. It can be considered as a limitation as HPC systems tends to allocate parts of the computing infrastructure to different applications.

Once a communication phase is recognized, one possibility is to apply CPU DVFS to save energy. In [**?**], authors intercept and record the sequence of MPI calls during program execution and consider a segment of program code to be reducible if there are high concentrated MPI calls or if an MPI call is long enough. The CPU is then set to run at the appropriate frequency when the reducible region is recognized again.

Power saving schemes presented above are effective in the sense that they permit to reduce application's energy consumption without significant performance degradation; however, those techniques are not scalable. In addition, they can hardly be used by non expert, since they require deep understanding of the applications.

More recently [**?**, **?**] showed that the energy consumption (the energy used when operating) of HPC systems can be significantly reduced through system reconfiguration mechanisms such as using DVFS to scale the processor's frequency down/up according to the workload. Those work can be seen as instances of the generic methodology presented in this chapter. Phase identification is the ability to identify recurring phases, or more generally to identify phases with each other. It is a desirable property for phase detection techniques, since it can be used in tuning algorithms to reuse previously found optimal configurations for recurring phases.

Phase identification is often used in conjunction with phase prediction. The earlier a phase is detected, the more the reconfiguration will save energy.

Table **??** summarizes possible reconfiguration decisions that can be taken given a specific workload/phase label. Decisions are selected so as to guarantee that they do not result in significant performance degradation; they lie on the fact that some specific workloads might not need certain resources. Note that some elements in the table are counter-intuitive: switching on memory banks when running I/O intensive workloads is indeed efficient. An increase in RAM size reduces the dependency on disk which in turn improves the overall performance. If the system has several disks, some can be switched off instead of sending them to sleep, the reverse operation is performed if necessary when running I/O intensive workloads. Also notice that the disk (respectively the NIC) automatically changes to active when it is accessed.

Table 1.1 Phase labels and associated energy reduction schemes.

Phase label	Possible reconfiguration decisions
compute intensive	switch off memory banks; send disks to sleep; scale the processor up; put NICs into LPI mode
memory intensive	scale the processor down; decrease disks or send them to sleep; switch on memory banks
mixed	switch on memory banks; scale the processor up send disks to sleep; put NICs into LPI mode
communication intensive	switch off memory banks; scale the processor down switch on disks
I/O intensive	switch on memory banks; scale the processor down; increase disks, increase disks (if needed)

1.5 Conclusion

HPC systems require energy during their full lifecycle from production, design, to transport, usage and recycling/dismantling. This chapter focuses on the usage aspect of HPC and how adapted and optimized software solutions can improve energy efficiency. Measuring and understanding energy consumption and energy efficiency improvements are challenging tasks that can generate some misunderstandings [**?**]. This chapter proposes some solutions for modeling the power consumption of servers. This allows to design power prediction models needed in order to take decisions. Meanwhile due to the complexity of applications and their non constant usage of resource, runtime support based for phase detections and phase identification are presented. These approaches allow the deployment and usage of a set of available green leverages which permit energy reduction.

REFERENCES

1. R. Meade and R. Diffenderfer, *Foundations of electronics, circuits and devices*, 4th edition. Thomson Delmar Learning, 2003.
2. H. J. van der Bijl, *Theory and Operating Characteristics of the Themionic Amplifier*, In Proceedings of the IRE (Institute of Radio Engineers), pp. 97–126, 1919.
3. V. Pallipadi, *Enhanced Intel SpeedStep Technology and Demand-based Switching on Linux* `http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux`, 2009.
4. *AMD Cool'n'Quiet Technology* `http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx`.
5. R. Basmadjian, F. Niedermeier, and H. De Meer. *Modelling and Analysing the Power Consumption of Idle Servers*. In Proceedings of 2nd IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2012), 2012.
6. A.P. Chandrakasan, R.W. Brodersen. *Minimizing Power Consumption in CMOS Circuits*. 4th edition. Thomson Delmar Learning, 2003.
7. R. Basmadjian and H. De Meer. *Evaluating and Modeling Power Consumption of Multi-Core Processors*. In Proceedings of the 3rd International Conference on Energy-Efficient Computing and Networking (e-Energy 2012), 2012.
8. R. Basmadjian, N. Ali, F. Niedermeier, H. De Meer and G. Giuliani. *A Methodology to Predict the Power Consumption of Servers in Data Centres*. In Proceedings of the

Please enter `\offprintinfo{(Title, Edition)}{(Author)}`
at the beginning of your document.

2nd International Conference on Energy-Efficient Computing and Networking (e-Energy 2011), 2011.

9. http://www.engineeringtoolbox.com/fans-efficiency-power-consumption-d_197.html
10. M. Jarus, A. Oleksiak, T. Piontek, J. Weglarz. *Runtime power usage estimation of HPC servers for various classes of real-life applications*, Future Generation Computer Systems, 2013, to appear.
11. Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. *A view of the parallel computing landscape*. ACM, 52:5667, October 2009.
12. Denis Barthou, Andres Charif Rubial, William Jalby, Souad Koliai, and Cedric Valensi. *Performance tuning of x86 openmp codes with maqao*. In Parallel Tools Workshop, pages 95113, Dresden, Germany, September 2009. Springer-Verlag.
13. Franck Cappello, Amina Guermouche, and Marc Snir. *On communication determinism in parallel hpc applications*. In Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on, pages 18, August 2010.
14. Karl Furlinger, Nicholas J. Wright, and David Skinner. *Performance analysis and workload characterization with ipm*. In Matthias S. Mller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, Tools for High Performance Computing 2009, pages 3138. Springer Berlin Heidelberg, 2010.
15. Markus Geimer, Felix Wolf, Brian J. N. Wylie, Daniel Becker, David Bohme, Wolfgang Frings, Marc-Andre Hermanns, Bernd Mohr, and Zoltan Szebenyi. *Recent developments in the scalasca toolset*. In Tools for High Performance Computing 2009, Proc. of the 3rd Parallel Tools Workshop, Dresden, Germany, chapter 4, pages 3951. Springer, 2010.
16. Michael Gerndt and Edmond Kereku. *Automatic memory access analysis with periscope*. In Proceedings of the 7th international conference on Computational Science, Part II, ICCS 07, pages 847854, Berlin, Heidelberg, 2007. Springer-Verlag.
17. Marty Itzkowitz and Yukon Maruyama. *Hpc profiling with the sun studio performance tools*. In Parallel Tools Workshop, Dresden, Germany, September 2009. Springer-Verlag.
18. T.M. Madhyastha and D.A. Reed. *Learning to classify parallel input/output access patterns*. Parallel and Distributed Systems, IEEE Transactions on, 13(8):802 813, August 2002.
19. W. E. Nagel, A. Arnold, M. Weber, H.-Ch. Hoppe, and K. Solchenbach. *Vampir: Visualization and analysis of mpi resources*. Supercomputer, 12:6980, 1996.
20. Thomas Panas, Dan Quinlan, and Richard Vuduc. *Tool support for inspecting the code quality of hpc applications*. In Proceedings of the 29th International Conference on Software Engineering Workshops, pages 182, Washington, DC, USA, 2007. IEEE Computer Society.
21. Hongzhang Shan, Katie Antypas, and John Shalf. *Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark*. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC08, pages 42:142:12, USA, 2008. IEEE Press.
22. Sameer S. Shende and Allen D. Malony. *The tau parallel performance system*. Int. J. High Perform. Comput. Appl., 20:287311, May 2006.

23. K. 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*. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 24, no. 1, pp. 1828, Jan. 2005.

24. Kirk W. Cameron, Rong Ge, and Xizhou Feng, *High-performance, power-aware distributed computing for scientific applications*, Computer **38** (2005), no. 11, 40–47.

25. 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*, Trans. Comp.-Aided Des. Integ. Cir. Sys. **24** (2006), no. 1, 18–28.

26. Vincent W. Freeh, Nandini Kappiah, David K. Lowenthal, and Tyler K. Bletsch, *Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs*, J. Parallel Distrib. Comput. **68** (2008), no. 9, 1175–1185.

27. Vincent W. Freeh and David K. Lowenthal, *Using multiple energy gears in mpi programs on a power-scalable cluster*, Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming (New York, NY, USA), PPoPP '05, ACM, 2005, pp. 164–173.

28. Rong Ge, Xizhou Feng, and Kirk W. Cameron, *Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters*, Proceedings of the 2005 ACM/IEEE conference on Supercomputing (Washington, DC, USA), SC '05, IEEE Computer Society, 2005, pp. 34–.

29. Canturk Isci, Gilberto Contreras, and Margaret Martonosi, *Live, runtime phase monitoring and prediction on real systems with application to dynamic power management*, Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (Washington, DC, USA), MICRO 39, IEEE Computer Society, 2006, pp. 359–370.

30. Canturk Isci and Margaret Martonosi, *Runtime power monitoring in high-end processors: Methodology and empirical data*, Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (Washington, DC, USA), MICRO 36, IEEE Computer Society, 2003, pp. 93–.

31. Canturk Isci and Margaret Martonosi, *Identifying program power phase behavior using power vectors*, in Workshop on Workload Characterization, 2003.

32. Russ Joseph and Margaret Martonosi, *Run-time power estimation in high performance microprocessors*, Proceedings of the 2001 international symposium on Low power electronics and design (New York, NY, USA), ISLPED '01, ACM, 2001, pp. 135–140.

33. Hideaki Kimura, Takayuki Imada, and Mitsuhisa Sato, *Runtime energy adaptation with low-impact instrumented code in a power-scalable cluster system*, Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (Washington, DC, USA), CCGRID '10, IEEE Computer Society, 2010, pp. 378–387.

34. Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal, *Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs*, Proceedings of the 2006 ACM/IEEE conference on Supercomputing (New York, NY, USA), SC '06, ACM, 2006.

35. Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis, *A comparison of high-level full-system power models*, Proceedings of the 2008 conference on Power

aware computing and systems (Berkeley, CA, USA), HotPower'08, USENIX Association, 2008, pp. 3–3.

36. Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch, *Adagio: making dvs practical for complex hpc applications*, Proceedings of the 23rd international conference on Supercomputing (New York, NY, USA), ICS '09, ACM, 2009, pp. 460–469.

37. Ghislain Landry Tsafack, Laurent Lefevre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa, *Beyond cpu frequency scaling for a fine-grained energy control of hpc systems*, SBAC-PAD 2012 : 24th International Symposium on Computer Architecture and High Performance Computing (New York City, USA), IEEE, oct 2012, pp. 132–138.

38. Ghislain Landry Tsafack, Laurent Lefevre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa, *A runtime framework for energy efficient hpc systems without a priori knowledge of applications*, ICPADS 2012 : 18th International Conference on Parallel and Distributed Systems (Singapore, Singapore), IEEE, Dec 2012, pp. 660–667.

39. `http://ringo.ams.sunysb.edu/index.php/MD_Simulation:_Protein_in_Water`

40. Mohammed Diouri, Manuel Dolz, Olivier Gluck, Laurent Lefevre, Pedro Alonso, Sandra Catalan, Rafael Mayo, Enrique Quintan-Orti, *Solving some Mysteries in Power Monitoring of Servers: Take Care of your Wattmeters!* , EE-LSDS 2013 : Energy Efficiency in Large Scale Distributed Systems conference , Vienna, Austria, April 22-24, 2013

41. Anne-Ccile Orgerie, Laurent Lefvre, and Jean-Patrick Gelas, *Demystifying Energy Consumption in Grids and Clouds*, The Work in Progress in Green Computing (WIPGC) Workshop, in conjunction with the first IEEE sponsored International Green Computing Conference, Chicago, USA, August 2010