Power Characterisation of Shared-Memory HPC Systems†

JAVIER BALLADINI[1], ENZO RUCCI[2], ARMANDO DE GIUSTI[2,4],
MARCELO NAIOUF[2], REMO SUPPI[3], DOLORES REXACHS[3]
AND EMILIO LUQUE[3]

[1] Department of Computer Engineering, Universidad Nacional del Comahue
Buenos Aires 1400, 8300 Neuquén, Argentina
javier.balladini@fi.uncoma.edu.ar
[2] III LIDI, Facultad de Informática, Universidad Nacional de La Plata
Calle 50 y 120, 1900 La Plata (Buenos Aires), Argentina
{erucci, degiusti, mnaiouf}@lidi.info.unlp.edu.ar
[3] Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona
Campus UAB, Edifici Q, 08193 Bellaterra (Barcelona), Spain
{remo.suppi, dolores.rexachs, emilio.luque}@uab.es
[4] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

Abstract. *Energy consumption has become one of the greatest challenges in the field of High Performance Computing (HPC). Besides its impact on the environment, energy is a limiting factor for the HPC. Keeping the power consumption of a system below a threshold is one of the great problems; and power prediction can help to solve it. The power characterisation can be used to know the power behaviour of the system under study, and to be a support to reach the power prediction. Furthermore, it could be used to design power-aware application programs. In this article we propose a methodology to characterise the power consumption of shared-memory HPC systems. Our proposed methodology involves the finding of influence factors on power consumed by the systems. It is similar to previous works, but we propose an in-deep approach that can help us to get a better power characterisation of the system. We apply our methodology to characterise an Intel server platform and the results show that we can find a more extended set of influence factors on power consumption.*

Keywords: *power characterisation, shared memory systems, microbenchmarks, green computing.*

1. Introduction

High Performance Computing (HPC) has had for decades the only goal of increasing the processing speed of computationally complex applications such as scientific applications. Supercomputers were designed exclusively with the aim of increasing the number of floating point operations per second

(FLOPS). This is reflected in the TOP500 list [13], which uses the FLOPS metric to determine the ranking of supercomputers. The performance and the trade-off price/performance were the most important objectives.

Thus, this led to the appearance of supercomputers that consume vast amounts of electrical power and produce so much heat that large cooling facilities must be constructed to ensure proper performance. According to the Lawrence Livermore National Laboratory (LLNL), for every watt (W) of energy consumed, 0.7 W is spent in cooling to dissipate the energy. The energy consumption of current supercomputers is so high that it produces a huge economic impact. In 2005, annual spending in electrical energy at LLNL was of 14.6 million dollars [8]. Currently, the fastest supercomputer in the world (according to TOP500) has a power of 7.7 MW. The energy consumption not only has an economic impact, it also affects the ecology and society due to the lack of exploitation of renewable and clean energy.

In 2007 the first list of the Green500 [9] was published, ranking the most energy-efficient supercomputers in the world. Thus, the new era of green computing began, avoiding the focus of performance-at-any-cost. Today, the TOP500 is not the only interesting ranking, but also the Green500.

Keeping the power consumption of a system below a threshold is a great challenge for HPC, motivated by the following reasons, among others. As an energy deficit can lead to service disruptions, the energy consumption below the available energy must be maintained. In addition, in order to improve system load factor, energy suppliers often provide electricity in low-load periods at a relatively low cost. They may also provide incentives, through conservation and load management programs, to encourage elimination or shifting of peak loads [5]. In case a computing centre is supplied by a intermittent renewable energy source (for example: wind or solar farms), the energy output from the power plant increases or decreases over time and the demand of the computing centre must change accordingly. All these reasons justify the necessity to accurately predict how changes in computing system parameters and utilisation will impact future power consumption.

It may be possible to predict the power using any of the following two approaches. One approach would be to perform an initial training phase in which the application is running at various system parameters and utilisation while the power is measured and recorded. This information can then be used to predict power in new application program executions. Another approach could be to identify different application phases and to search for historical power data of microbenchmarks that match with these identified phases. If the identified phases do not exist in the history, new microbenchmarks are added together with their power information.

The first approach does not seem to be difficult to achieve. However, unfortunately, the production systems do not enable (accurate and fine grained) power measurements (for the moment) because it increases costs due to board space constraints and the need for additional components. In absence of direct power measurement, the commonly used alternative is to make power models. The basic idea behind power modelling is to take as input some Performance Monitoring Counters (PMCs) and software counters and use those to calculate power consumption. Previous works [4,10,7,6,12] calculate the total system power consumption using several learning techniques such as linear regression, recursive learning automata, and stochastic power models. There are many different hardware counters that can be tracked, while only a few can be tracked simultaneously. Thus, we have the problem to choice the best counters that will result in an accurate model. A power characterisation of computing system could be used to judiciously select these counters.

In the second approach, microbenchmarks must be generalised in order to match them with a major number of applications phases. A power characterisation of the system is also necessary to make this generalisation. Furthermore, a power characterisation could be used to design power-aware application programs.

In this article we propose a methodology to characterise the power consumption of shared-memory HPC systems. Our proposed methodology involves the finding of influence factors on power consumed by the systems, that is, a sensitivity analysis of workload properties and system parameters on the power behaviour. The workload considers the computation and communication aspects of applications while disk input/output operations are excluded due to being a large issue to be discussed beside computation.

Our methodology is similar to previous works [4,10], but we propose an in-deep approach of the impact of workload properties on power consumption. This study can help us to get a better power characterisation of system computation.

The remaining of this article is organised as follows. Section 2 exposes the methodology overview used to characterise the power consumption of shared-memory HPC systems. Sections 3, 4, 5 and 6 describe the methodology's phases and present a case of study. Finally, section 7 discusses the conclusions and future works.

2. Methodology Overview

Our methodology explores influence factors of workload properties and system parameters. The methodology consists of four phases:

1. *Identification of system architecture components and parameters*. The use of different parts of hardware and system parameters normally produce different power consumption, so it is necessary to first determine what are the components that make up the architecture and the configurable parameters of the system under study.
2. *Development of microbenchmarks*. Building of small synthetic applications, called microbenchmarks, whose operations stress and evaluate special features of each architectural component.
3. *Test Cases Creation and Electrical Power Measurement*. Instrumentation of the HPC system with a power meter, and measurement of the power used to compute the microbenchmarks at different system parameters.

4. *Finding of Power-Influence Factors*. Identification of influence factors based on measurement result analysis.

The phases are further explained and supported by a case of study in sections 3, 4, 5 and 6.

3. Phase I: Identification of System Architecture Components and Parameters

A computer consists of a set of components or modules of three basic types that communicate with each other: processor, memory and Input/Output (I/O). So, as we are focused on computation but not the I/O phases of programs, we can determine three system aspects (excluding I/O) to analyse:

1. CPU's functional units. Multi-core processors are composed of several cores (or CPUs, Central Processing Unit), and a core include several independent functional units such as Integer Unit, Floating-Point Unit, Branch Processing Unit, etc. These units inside a core are candidates to consume different power.
2. Data access. A shared memory system provides a global physical address space accessed from any core, and a design key issue of these systems is in the organisation of the memory hierarchy. The cores may have access to a central shared memory (UMA -Uniform Memory Access-), or may participate in a memory hierarchy with both local and shared memory (NUMA -Non-Uniform Memory Access-). Common memory organisations use shared caches, buses, and interconnection networks, and we need to evaluate the influence on power consumption of these parts.
3. System Parameters. The parameters of the system that can modify the power are mainly divided in two types: Resource Hibernation and Dynamic Voltage Scaling.
    a. Resource Hibernation: The computer components consume power even when idle. Thus, the technique of resource hibernation turns off or disconnects components in idle moments. The components that can be hibernated depend on each system and can include: hard disks, cores, network interface cards, and memories.
    b. Dynamic Voltage Scaling (DVS): Reducing the supply voltage reduces power consumption. However, it increases the delay of logic gates, so that the clock frequency should be reduced to allow the circuit to work properly. Current systems allow us to change the CPU's voltage/clock-frequency and it is clearly a factor to analyse.

In particular, as a case of study, we evaluate the parallel platform Intel Server System SC5650BCDP, a dual socket with dual core Intel Xeon E5502 [1] processors and 16GB of main memory (8GB per socket). Figure 1 shows the architecture and memory hierarchy, including memory sizes, of the parallel system. It is a NUMA system, each processor has an integrated memory controller, and the interconnection system between processors is the Intel QuickPath Interconnect (QPI), which provides high-speed, point-to-point links. The available CPU's clock frequencies are: 1.6, 1.73 and 1.86 GHz. For the best of our knowledge, the system does not support physical hot-plug of memories.

The processor supports low power states (C-states) at individual core. On the contrary, Intel Turbo Boost technology is not supported by E5502 processors.

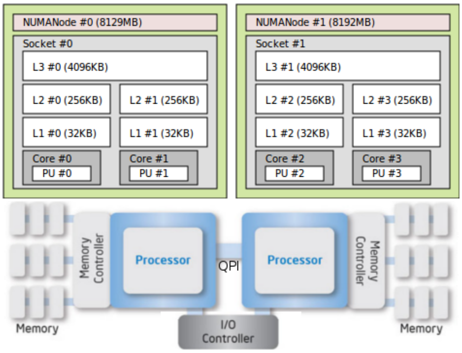


Fig. 1. *Architecture and memory hierarchy of the system under study.*

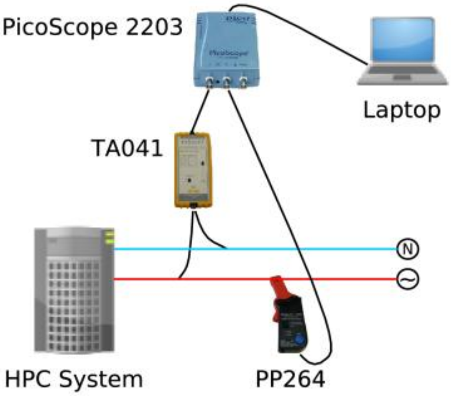


Fig. 2. *Measurement connection diagram.*

4. Phase II: Development of Microbenchmarks

After identifying what are the components that make up the architecture under study, we developed a set of microbenchmarks that allow us to characterise it. The microbenchmarks were developed using language C, gcc compiler version 4.6.3, under GNU/Linux with Pthreads library for threads management. To guarantee that the compiler does not affect the microbenchmark's purpose, assembler codes generated by it were verified using the *objdump* command (with -d option).

Each microbenchmark launches four threads, where each thread runs a loop executing between 1 and 2.5 Gigaoperations. Taking into account the hardware characteristic of the support architecture, we consider the next factors at the time of developing the microbenchmarks:

Operation and Data Type. Different operations on different data types have different complexity hardware implementation. We developed microbenchmarks for add, multiply and division operations. Also, four special microbenchmarks were developed. The first one evaluates the cost of performing no specific operation. The rest of them evaluate the cost of performing a complex operation that involves other simpler ones. Data is read from three vectors accessed sequentially (stride = 1). The data types used are: int (32-bit integer), float (32-bit floating point) and double (64-bit floating point).

Operands Accessing Mode. Load and store instructions have different computational cost. It is interesting to study what occurs with power. We developed two microbenchmarks: one that only executes load instructions and one that only executes store instructions.

Non-Uniform Memory Access. The support architecture has a common memory address space, but the accesses may be local or foreign. The foreign access uses QPI interconnection link. Thus, we developed two microbenchmarks that write data in main memory: one with local access and one with foreign access.

Resources Usage Efficiency. Systems are often scheduled incorrectly and, as a result, resources usage efficiency decreases. We developed two microbenchmarks to study the power influence of this factor: an efficient one and an inefficient one. Both microbenchmarks write data in memory but they differ in the number of used cores. The efficient microbenchmark uses all the cores of the architecture used, while the inefficient microbenchmark use only one of them.

Parallel Programming Model. Programming models differ on the way of dealing process communication and synchronisation, either as shared memory or distributed memory. OpenMP is the most widely used model on shared memory, while MPI is the corresponding for distributed memory. We used the NAS Parallel Benchmarks (NPB) [2], implemented both in OpenMP and MPI, to evaluate the parallel application programming model influence on parallel architectures power. We selected the CG, IS and EP benchmarks which are computation bound, and the MG benchmark which is communication bound [11]. We selected only these benchmarks because the others (NPB) do not provide additional information for our objective. We determined the benchmarks problem size so that its main memory requirements for execution are met and memory swapping never happens. Particularly, we chose a class C problem size defined by the specification of the NPB.

Cache Friendliness. The access to the memories of parallel machines (implemented with different technologies and placed at different locations) is a candidate to be a power influence factor. The developed microbenchmarks can be classified into two groups. The first group microbenchmarks are cache friendly, that is, they have a good cache hit rate. The second group microbenchmarks are not cache friendly, that is, they have bad cache performance at all the levels. The cache friendly microbenchmarks work with a data set smaller than the L1 cache size. The data set size of the no cache friendly microbenchmarks is bigger than the L3 cache size.

In a first stage, we focused on intensive use of the CPU, so we have left the analysis of the C-states (as a power influence factor) for the future. Table 1 shows a description of some of the developed microbenchmarks. Input parameters are: Characteristic Memory Access Pattern, Data Type, Characteristic Basic Operation, Cache Friendliness, Parallelism Level and Clock Frequency. The Characteristic Memory Access Pattern indicates the data structures used by the microbenchmark and how they are accessed by it. The Data Type parameter indicates the data type of the data structures that were defined in previous column. The Characteristic Basic Operation parameter represents the operation performed by each thread of the microbenchmark. The Cache Friendliness parameter indicates whether the microbenchmark is cache friendly or not. The Parallelism Level parameter is adjusted to the number of cores of the support architecture (one thread per core). The Clock Frequency parameter depends on the available processors clock frequencies. Different executions for different k_1, k_2 and f_i values were done. In all the cases, each thread performs the same basic operation on its own data set.

Table 1. *Some developed microbenchmarks.*

Microbenchmark	Charac-teristic Memory Access Pattern	Data Type	Characteristic Basic Operation	Cache Friend-liness	Para-llelism Level	Clock Fre-quency
addFloatCache	a,b,c: stride-1	Float	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = a_j + b_j$	Yes	4	f_i
divDoubleMem	a,b,c: stride-1	Double	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = a_j / b_j$	No	4	f_i
storeDoubleCache	a,b: stride-1	Double	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ a_j = b_j = 0$	Yes	4	f_i
loadDoubleCache	a,b: stride-1	Double	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ x = a_j, y = b_j$	Yes	4	f_i
noOpIntCache	a,b, c: stride-1	Int	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ x = a_j, c_j = b_j$	Yes	4	f_i
multiOpIntCache	a,b,c: stride-1	Int	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = 8 * a_j + b_j + a_j * a_j * \frac{5}{b_j} + b_j * b_j * a_j$	Yes	4	f_i
multiAddIntCache	a,b,c: stride-1	Int	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = a_j + b_j + a_j + a_j + 5 + b_j + b_j + a_j$	Yes	4	f_i
umaFloatCache	a,b,c: stride-1	Float	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = a_j + b_j + c_j, a_j = c_j, b_j = c_j$	Yes	4	f_i
efficientFloatCache	a,b,c: stride-1	Float	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = a_j / b_j$	Yes	4	f_i
inefficientFloatCache	a,b,c: stride-1	Float	$for_{i=0}^{k_1} for_{j=0}^{k_2}\ c_j = a_j / b_j$	Yes	1	f_i
…	…	…	…	…	…	…

5. Phase III: Test Cases Creation and Electrical Power Measurement

Once the microbenchmarks are developed, the next step is the creation of test cases. Test cases are written using bash scripts that execute combinations of microbenchmarks and system parameters. Once the HPC system is instrumented with the power meter, the test cases are run and measured.
Following, we explain how to scale the clock frequency using a bash command, and later we expose the power measurement methodology.

5.1 System Parameter: Clock Frequency Scaling

Modern general purpose processors can scale the frequency of each core individually. Access is through the Advanced Configuration and Power Interface (ACPI). It is possible to know, for a given core, the available frequencies and the frequency currently in use, respectively, reading the following two files in GNU/Linux:

```
  /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_
frequencies
  /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

To change the frequency it is possible to use *cpufreq-selector* command. For example, running the command "`cpufreq-selector -c 0 -f 1000000`" the core number 0 is set to 1 GHz.

5.2 Power Measurement Methodology

This section explains some definitions about power and energy, and the methodology typically used to measure the electrical power of a whole system, detailing the instruments utilised by us.

Power is the rate at which the system consumes electrical energy. The watt (W) is the unit of real power, equivalent to 1 joule by second (1 J/s), and it is the product of current times voltage. Energy is the total amount of electrical energy that the system consumes over time, and is measured in joules or watt-hour (Wh).

We are interested in power; and energy can be calculated by integrating power over time. We measure the power consumption of the whole shared-memory HPC system. For this, we use the oscilloscope PicoScope 2203, the TA041 active differential oscilloscope probe, and the PP264 60 A AC/DC current clamp, all products of Pico Technology. The electrical signals captured by the dual-channel PicoScope 2203 are transmitted in real-time via USB to a laptop. The voltage is measured using the TA041 probe that is connected to one oscilloscope's input channel. The current of the phase conductor is measured using the PP264 current probe that is connected to the other input channel of the oscilloscope. Then, power is calculated as the product of measured voltage and current. The sample rate for the experiments was of 1000 Hz. Figure 2 shows the measurement connection diagram.

6. Phase IV: Finding of Power-Influence Factors

After running and measuring test cases, we proceed to find power-influence factors. Figure 3 shows the average power for the microbenchmarks developed to evaluate "Operation and Data Type" and "Cache Friendliness" factors. The microbenchmarks run at the maximum clock frequency. It can be observed that microbenchmarks with bad cache performance produce higher average power. Regarding the "Operation and Data Type" factor, it can be seen that the operation to perform has no significant impact on int data type, except when the operation is composed of multiple ones (multiOp, multiAdd and multiDiv). While floating point data types (float and double) have different size, their power behaviour are similar. Also, it can be seen that the operation to perform impacts on the average power produced, being these values generally lower than those for int microbenchmarks.

The influence of "Data Type", "Operands Accessing Mode" and "Cache Friendliness" factors are analysed in Figure 4. This chart shows the average power for the microbenchmarks developed to evaluate those factors at the maximum clock frequency. It can be observed that average power for load and store microbenchmarks are similar when the cache has a good performance but this similarity does not maintain when cache miss rate increases. It can be seen that store instructions performed in main memory produces higher average power, particularly when using int and double data types.

Figure 5 shows the average power for “Non-Uniform Memory Access” and “Cache Friendliness” microbenchmarks at the maximum clock frequency. It can be observed that average power hardly varies when the cache performance is good. In the opposite situation, that is when the cache has a bad performance, the use of the QPI interconnection link decreases average power.

Figure 6 allow us to evaluate the power influence of “Resources Usage Efficiency” and “Cache Friendliness” factors. This chart shows the average power for the microbenchmarks developed to evaluate those factors at the maximum clock frequency. It can be seen that average power increases when resources are used efficiently, beyond the cache performance.

The “Parallel Programming Model” influence factor is analysed in Figure 7. This chart shows the average power for each NAS benchmark executed at the maximum clock frequency. It can be observed that the programming model practically has no impact on the average power of each benchmark tested.

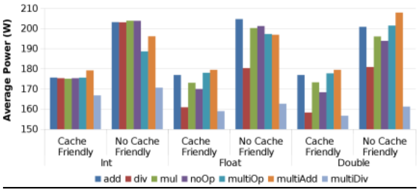
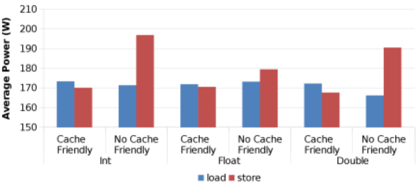


Fig. 3. *Influence of “Operation and Data Type” and “Cache Friendliness”.*



”.

Fig. 4. *Influence of “Operands Accessing Mode” and “Cache Friendliness”.*

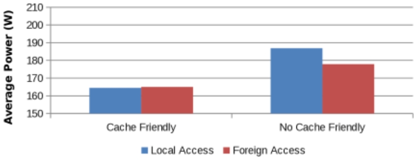


Fig. 5. *Influence of “Non-Uniform Memory Access” and “Cache Friendliness”.*

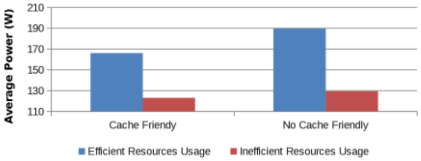


Fig. 6. *Influence of "Resources Usage Efficiency" and "Cache Friendliness".*

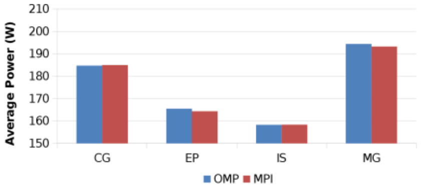


Fig. 7. *Influence of "Parallel Programming Model*

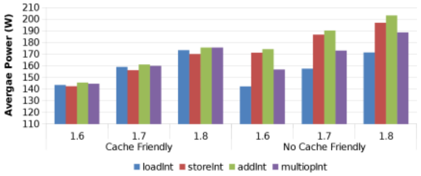


Fig. 8. *Influence of "Voltage and Frequency Scaling" and "Cache Friendliness".*

Figure 8 allow us to assess the influence of "Cache Friendliness" and "Voltage and Frequency Scaling" factors. This chart shows the average power for int microbenchmarks with good and bad cache performance at different clock frequencies (for readability only four microbenchmarks are shown). It can be seen that average power increases when the clock frequency increases, regardless the cache performance.

From previous analysis, we can confirm the influence of the studied factors on the average power of the support architecture:

- When working with floating point data types (float or double), the operation to perform must be taken into account because it influences the produced average power. It does not occur the same with int data type. Beyond data type used, the ratio of the number of mathematical operations to the number of data read from main memory must be a factor to consider.
- When the cache has a good performance, local and foreign accesses have similar power behaviours. Nevertheless, when cache

performance is bad, foreign accesses produce lower average power than local accesses.

- Increase the efficiency in resources usage produces higher average power.
- The parallel programming model is not a power influence factor (although it is an energy influence factor as we analysed in [3]).
- The cache performance has no influence on the average power produced by load instructions. However, store instructions produces higher average power when they are performed in main memory, particularly when int and double data types are used.
- Beyond the operands accessing mode, the operation to perform, the data type, the parallel programming model and the cache performance, the average power increases when the clock frequency increases.

7. Conclusions and Future Works

In this work we present a methodology to characterise the power consumption of shared-memory HPC systems. The power characterisation can be used to know the power behaviour of the system under study in order to design power-aware application programs, and to be a support to reach the power prediction. We apply our methodology to characterise an Intel server platform and the results show that we can find an extended set of influence factors on power consumption.

As future works, we will analyse the influence of C-states on power consumption of our platform. Later, we plan to find a way to automatically characterize a system, following our methodology. Furthermore, we will continue working on power prediction of HPC systems using the information obtained with the power characterisations.

References

1. Intel E5500 datasheet - Vol 1 (Accessed on 2012), http://www.intel.com/content/www/us/en/processors/xeon/xeon-5500-vol-1-datasheet.html.
2. NAS Parallel Benchmarks (Accessed on 2012), http://www.nas.nasa.gov/publications/npb.html.
3. Balladini, J., Suppi, R., Rexachs, D., Luque, E.: Impact of parallel programming models and cpus clock frequency on energy consumption of hpc systems. In: AICCSA. pp. 16-21 (2011).
4. Bircher, W.L., John, L.K.: Complete system power estimation using processor performance events. IEEE Transactions on Computers 61, 563-577 (2012).
5. Capehart, B.L. (ed.): Encyclopedia of Energy Engineering and Technology. CRC Press (2007).

6. Contreras, G.: 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. pp. 221-226. ACM Press (2005).
7. Economou, D., Rivoire, S., Kozyrakis, C.: Full-system power analysis and modeling for server environments. In: In Workshop on Modeling Benchmarking and Simulation (MOBS (2006).
8. Feng, W.C.: The importance of being low power in high-performance computing. Cyberinfrastructure Technology Watch Quarterly 1 (3) (August 2005).
9. The Green500 website (Accessed on 2012), http://www.green500.org/.
10. Jiménez, V., Cazorla, F.J., Gioiosa, R., Valero, M., Boneti, C., Kursun, E., Cher, C.Y., Isci, C., Buyuktosunoglu, A., Bose, P.: Power and thermal characterization of power6 system. In: Proceedings of the 19th international conference on Parallel architectures and compilation techniques. pp. 7-18. PACT '10, ACM (2010).
11. Jin, H., Hood, R., Chang, J., Djomehri, J., Jespersen, D., Taylor, K.: Characterizing application performance sensitivity to resource contention in multicore architectures. Tech. rep., NASA Advanced Supercomputing (NAS) Division (2009).
12. Qiu, Q., Wu, Q., Pedram, M.: Stochastic modeling of a power-managed system: construction and optimization. In: Proceedings of the 1999 international symposium on Low power electronics and design. pp. 194-199. ISLPED '99 (1999).
13. The TOP500 website (Accessed on 2012), http://www.top500.org/.