Power and Thermal Characterization of POWER6 System

Víctor Jiménez [†] Carlos Boneti [‡][*] Francisco J. Cazorla [†]
Roberto Gioiosa [†] Eren Kursun [⋆] Chen-Yong Cher [⋆] Canturk Isci [⋆]
Alper Buyuktosunoglu [⋆] Pradip Bose [⋆] Mateo Valero [†]

[†] Barcelona Supercomputing Center, Spain
{victor.javier,carlos.boneti,francisco.cazorla,roberto.gioiosa,mateo.valero}@bsc.es

[‡] Schlumberger BRGC, Rio de Janeiro, Brazil
{cboneti}@slb.com

[⋆] IBM T. J. Watson Research Center, Yorktown Heights, USA
{ekursun,chenyong,canturk,alperb,pbose}@us.ibm.com

ABSTRACT

Controlling power consumption and temperature is of major concern for modern computing systems. In this work we characterize thermal behavior and power consumption of an IBM POWER6^{TM}-based system. POWER6 is a processor for high-end server systems presenting multiple levels of parallelism (SMT, CMP and SMP). Thus we believe a detailed characterization of such a system is of great value for the HPC and parallel architecture communities. We perform the characterization at several levels: application, operating system, and hardware level, both when the system is idle, and under load. At hardware level, we report a 25% reduction in total system power consumption by using the processor low power mode. We also study the effect of the hardware thread prioritization mechanism provided by POWER6 on different workloads and how this mechanism can be used to limit power consumption. At OS level, we analyze the power reduction techniques implemented in the Linux kernel, such as the tickless kernel and the CPU idle power manager. At application level, we characterize the power consumption and the temperature of two sets of benchmarks (METbench and SPEC CPU2006) and we study the effect of workload characteristics on power consumption and core temperature.

From this characterization we derive a model based on performance counters that allows us to predict the total power consumption of the POWER6 system with an average error under 3% for CMP and 5% for SMT. To the best of our knowledge, this is the first power model of a system including CMP+SMT processors. Finally, we show that the static decision on whether to consolidate tasks into the same core/chip, as it is currently done in Linux, can be improved by dynamically considering the low-power capabilities of the underlying architecture and the characteristics of the application (up to 5X improvement in ED^{2}P).

Categories and Subject Descriptors

B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; D.4.8 [**Operating Systems**]: Performance—*Measurements*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Experimentation, Measurement, Performance

1. INTRODUCTION

Processor architectures that leverage thread-level parallelism (TLP), as the means of improving system performance, have become the common trend in architecture design. In fact, most chip vendors already employ multithreaded processors in their designs that exploit TLP, using SMT and CMP architectures. As process technologies advance, the trend is to have more cores per chip, where each core can further increase the amount of concurrent threads via SMT, such as the IBM POWER5 [27] and POWER6 [18] and the Intel i7 [6]. While multithreaded processors provide better performance per watt ratios than monolithic architectures, the power dissipation continues to be a key performance limiter also for multithreaded architectures. Consequently, power and thermal characteristics of processors are one of the primary design constraints, and motivate an active research area.

Energy, power and thermal management are of paramount importance in many environments ranging from embedded and consumer devices to High Performance Computing (HPC) systems. In the former case, improvements in battery capacity simply have not kept pace with ever-more-powerful processors, limiting device use to short time periods. In the latter case, supercomputers and data centers provide huge amounts of computation power (necessary, for instance, for weather forecasting, climate research, molecular modeling and other areas), with very high associated energy costs. A study from the U.S. Environmental Protection Agency (EPA) estimates that national energy consumption by servers and data centers will reach more than 100 billion kWh annually and representing $7.4 billion in electricity cost [8]. In HPC systems, besides the heat dissipation issue, the increasing power consumption leads to additional problems in the power delivery and energy costs that account for a considerable percentage of the expenses of a data center. It is certain that managing and reducing the temperature and power consumption is a critical problem that must be addressed in all levels of computing systems, from the application layer to the hardware. As an example, most of the latest processors available in the market employ several techniques to reduce power consumption [10, 21]. From the OS perspective, the Linux kernel also implements features to reduce power [22, 26, 28].

POWER6 is a processor from IBM designed for high-

[*] This work was done while the author was at BSC.

PACT'10, September 11–15, 2010, Vienna, Austria.

end server systems. The processor is a dual-core processor where each core is an SMT capable of running two hardware threads. Moreover, the POWER6 provides power monitoring capabilities and temperature sensors to monitor the processor status and a special *nap mode* to control power consumption. In addition to the low-power nap mode, the POWER6 processor implements microarchitectural mechanisms for power management such as pipeline throttling, multiple voltage domains, memory controller dynamic modes and memory throttling [10]. POWER6 also includes a *Hardware Thread Prioritization* technique. While this is not designed for power management, it can also be used to improve the power and thermal behavior of the system [10].

This work specifically focuses on characterizing and modeling the power and thermal behavior of the POWER6 architecture. We explore power and thermal behavior with various power management techniques provided by POWER6 and evaluate their impact at multiple levels: application level, operating system level, and the hardware level. Such multi-level characterization is particularly important to understand the impact of the architectural and system-level management techniques across the computing system hierarchy. The experiments are performed on a JS22 blade, containing a Dual Chip Module (DCM) with two POWER6 processors. The characterizations are performed at the system level under multiple operating conditions, from idle to extreme load conditions using specifically designed microbenchmarks that stress particular processor resources, and the SPEC CPU2006 benchmarks. The main contributions of this work are the following:

- **Hardware level**: We present the first characterization of a real hybrid CMP/SMT implementation. We also demonstrate the impact of POWER6's hardware-thread prioritization mechanism on power consumption. Our results show that workload-aware manipulation of thread priorities improves the system's energy-delay product by as much as 25%. Finally, we show the power and thermal characteristics of the nap mode, and the combined effect of employing the nap mode and hardware thread priorities. These evaluations show very significant benefits, reducing up to 26% the core temperatures and 25% the total system power consumption.

- **OS level**: We explore the effectiveness of power and thermal management techniques present in modern OS for the POWER architecture, including tickless kernel and idle power managers. We demonstrate the benefits of these approaches and their dependence on other system components such as timer interrupt periods.

- **Application level**: We characterize system behavior with a set of specialized microbenchmarks and SPEC CPU2006 benchmarks. We correlate power and temperature with performance counters and derive an accurate model that represents the power and temperature impact of different workload performance characteristics. We show that high IPC applications have a bigger effect on the core temperature and memory-intensive applications do not heat the cores as much but cause the system to consume more power [19].

Based on that we derive several conclusions and we show two use cases to improve power behavior:

- **Power Model**: We develop a model, based on performance counters, to predict the power consumption of a POWER6 system. The model accurately predicts system power consumption with an average error under 4.5%. To the best of our knowledge, this is the first model for real CMP/SMT processor-based systems.

- **OS scheduling**: The JS22 system includes two POWER6 chips, each of which is a CMP/SMT chip. In such a system, thread placement affects both performance and power consumption. We show that by placing threads in a workload- and package-aware manner, we can achieve significant energy improvements, without incurring significant performance degradation, with a 3.7X reduction in energy-delay product.

The rest of this paper is structured as follows: Section 2 provides the necessary background and details on the processor and the Linux kernel features that we analyze. Section 3 describes the methodology, the infrastructure and the benchmarks used in this work. Section 4 shows the results of our experiments on a JS22 blade, both in idle conditions and under load. In Section 5 we present two use cases that follow from our characterization results: (i) a system-level power model and (ii) power-aware thread placement. Section 6 lists the related work and Section 7 concludes this study.

2. BACKGROUND

2.1 The IBM POWER6 Processor

POWER6 is a dual-core chip where each core can be run in a two-way SMT mode. POWER6 is an in-order processor with limited out-of-order execution for floating point operations. Each core has a 64KB L1 I-cache and D-cache. The cores have a 4MB private L2 cache connected to the L3 controller and to the memory controller through the symmetric multiprocessor (SMP) interconnect fabric. The optional off-die L3 cache is shared by both cores. Depending on the configuration, each chip has one or two memory controllers that interface to the DRAM memory.

POWER6 systems integrate a thin hypervisor layer that abstracts the real hardware and provides the capability of running several virtual machines simultaneously on the same physical resources. This virtualization mechanism is completely transparent and does not require any modification of the guest OS. However, collaboration between the guest OS and the hypervisor has significant benefits for improving chip utilization and throughput as well as for effective power management.

In this paper we are particularly interested in the interaction between the guest OS and the hypervisor for effective power and performance management. POWER6 implements specific methods for the guest OS to release hardware threads and cores when there are no runnable processes available. This is done by invoking the cede_processor hypervisor call, which enables the hypervisor to dispose of the hardware thread and assign the resources to another virtual machine or to employ power management techniques on the unused resources. In our environment, we run only one virtual machine, thus, the hypervisor performs one of the following operations when the cede_processor is invoked from a given hardware thread: *(i)* If the other hardware thread on the same core is under use, the hypervisor turns off the hardware thread and puts the core in Single Thread (ST) mode. This effectively assigns more hardware resources to the running process, thus improving single-thread performance. Moreover, while this mode does not directly target at reducing power, as several functional units are not utilized during ST mode, overall power consumption also decreases. *(ii)* If hypervisor has already turned off the other hardware thread in the core (i.e., the core is already in ST mode), the hypervisor puts the core in *nap mode.*

Nap Mode: POWER6 implements a low-power mode per core called *nap mode.* This mode turns off the internal clocks and restricts the operation of the functional units in the core. Reducing active power consumption by turning off the clocks reduces the temperature as well, which further reduces leakage power. In this paper we show the effect of nap mode on both system power consumption and core temperature.

Thread Priorities: POWER6 processor employs a *thread priority* mechanism, through software/hardware co-design, that controls the instruction decode rate for each hardware thread with eight priority levels [1, 4]. The software-controlled priorities range from 0 to 7, where 0 means the thread is switched off and 7 means the thread is running in Single Thread mode (i.e., the other thread is off). The thread's software-controlled priority is enforced by the hardware at decode stage, which determines the actual number of decode cycles assigned to the hardware thread. In general, the higher the priority of a thread with respect to the other thread on the same core, the higher the number of decode cycles assigned to the thread. Consequently, the thread with a higher priority receives more resources and can obtain higher performance.

The main motivation of the software-controlled priority is to address instances where biasing thread performance is desirable because one thread is not really progressing or because it requires some level of Quality of Service. For example, as shown in Figure 1, the POWER6 Linux kernel reduces the priority of the idle process or of any process spinning on a lock in order to give more hardware resource to the other running thread. Moreover, depending on the application, software-controlled priorities can significantly improve both throughput and execution time [4]. In this paper, we also show how software-controlled priorities can be used for power management, to improve the performance-per-watt ratio.

2.2 Linux Kernel

The Linux guest OS, running on POWER6 implements several mechanisms to reduce power consumption [22, 28, 26]. These mechanisms are grouped under *CPU Idle Power Manager (PM) Support* and *tickless kernel.*

CPU Idle PM Support: When a CPU is idle (i.e., there is no process available to run on that CPU other than the idle process) the OS runs the Idle Process which basically loops over the code shown in Figure 1. In general, the Idle Process tries to take advantage of the low-power mechanisms provided by the underlying architecture if available, otherwise the process simply executes a long latency instruction.

In POWER6, the guest OS does not have direct access to the physical hardware and therefore gives the thread up to the hypervisor (cede_processor, line 24 in Figure 1) when idle. The hypervisor, in turn, decides to put the core in ST or in low-power mode. It is highly probable that another process becomes runnable shortly after the Idle Process has been scheduled on the CPU. Therefore, the kernel does not directly go into low power mode but waits for some (programmable) amount of time (*snooze loop*, lines 8-13) to improve responsiveness.

Since the snooze loop does not perform any useful work, the kernel reduces the priority of the Idle Process while in the snooze loop down to the minimum (HMT_low and HMT_very_low). In this way the process running on the other hardware thread receives more hardware resources and its performance may increase. At the same time, since HMT_very_low also reduces the number of instructions decoded per second, the OS also saves power. Thus, by using hardware thread priorities, the OS reduces power consumption while improving system throughput. If no process becomes runnable after the snooze delay, the OS invokes the hypervisor call cede_processor (line 24) and releases the hardware thread to the hypervisor.

```
 1 if (__get_cpu_var(smt_snooze_delay)) {
 2  start_snooze = get_tb() +
 3   __get_cpu_var(smt_snooze_delay) *
 4   tb_ticks_per_usec;
 5  local_irq_enable();
 6  set_thread_flag(TIF_POLLING_NRFLAG);

 8  while (get_tb() < start_snooze) {
 9   if (need_resched() || cpu_is_offline(cpu))
10    goto out;
11   ppc64_runlatch_off();
12   HMT_low();        /* priority 2 */
13   HMT_very_low();   /* priority 1 */
14  }

16  HMT_medium();  /* priority 4 */
17  clear_thread_flag(TIF_POLLING_NRFLAG);
18  smp_mb();
19  local_irq_disable();
20  if (need_resched() || cpu_is_offline(cpu))
21   goto out;
22 }

24 cede_processor();

26 out:
27 HMT_medium();  /* priority 4 */
```

Figure 1: Code snippet from the idle loop for the POWER6 processor. This piece of code is continuously called from the function cpu_idle while the system is idle.

In Section 4.1.1 we evaluate the Linux CPU Idle PM Support implementation for POWER6 processors and analyze the idle power behavior of the system with different kernel implementations.

Tickless Kernel: As explained in the previous section, when a core is idle, the OS tries to put it in low power mode. One challenge in this power management scheme is the interrupt behavior of the system. Any interrupt received in the low-power state forces the CPU to go back to the active state to handle the interrupt. Timer interrupts are the most common interrupts received by a CPU and local timers fire interrupts periodically. Moreover, if the CPU is idle, the timer interrupt handler does not perform any operation, while still forcing the CPU to wake up. Thus, reducing the amount of interrupts delivered to the idle cores increases the time the cores stay in low power mode and improves overall system power efficiency.

The tickless kernel mechanism (kernel version $\geq$ 2.6.21) reduces the effect of timer interrupts by disabling the periodic timer interrupts when a CPU is idle [26]. In practice, instead of programming the local timer to expire in 1/HZ second (periodic timer), the kernel programs the timer to expire in the next, non-periodic, timer event (e.g., a software timer programmed by a task that has called the sleep() system call). In Section 4.1.2 we evaluate the effect of the tickless kernel on the power consumption of the POWER6 system.

3. METHODOLOGY

System Infrastructure: In our experiments we use an IBM JS22 BladeCenter, with two dual-core, 2-way SMT POWER6 chips running at 4.0GHz. Thus, the system presents 8 logical CPUs to the hypervisor and the OS layer. Our system does not include an off-chip L3 cache. Therefore, the last level of cache in our system is the 4MB L2 cache private to each core. Only one memory controller per processor is available in our configuration. The amount of DRAM memory is 15GB. The system runs SUSE Linux Enterprise 10 SP2 with kernel version 2.6.28 patched with perfmon2 3.8 in order to access the performance counters. Our JS22 blade is located in a server room where the temperature is kept constant.

We use the EnergyScale architecture [2] to measure power and temperature in the JS22 via a standard service processor, called the Flexible Support Processor (FSP). The EnergyScale implementation includes an optional plug-in card, containing a micro-controller called Thermal and Power Management Device (TPMD). The FSP and TPMD are controlled by the BladeCenter (chassis) management module. The sensors incorporated into this device are also used to protect the system from thermal emergencies and to cap power consumption in case of necessity. Thus, we can assure that they are calibrated and accurate enough to perform our study. Our observations during the experimental process confirm this claim as well.

We develop an external framework that monitors each core's temperature and total system power consumption through the management module. Although some details on the physical location of the temperature sensors can be seen in [10], we mostly treat the information read from the management console in a black-box fashion. This monitoring framework allows us to gather temperature measurements at 1s granularity and power measurements at 1min granularity. Although the architecture performs finer-grain power measurements, the monitoring interface only exposes 1min averages for the power. While these granularities are sufficient for characterization purposes, we also develop a power model that provides fine-grain power estimations by monitoring performance counters, as one of the contributions of this paper. This power model can be used for dynamic power management policies that operate at smaller time scales.

The power and temperature measurements shown in this paper are, unless stated otherwise, normalized to the ones obtained when the system is idle. Because of clearance we cannot report raw measurements.

Benchmarks: Real applications present phases and significant dynamic variations during their execution, which complicates fine-grain architectural characterization. Microbenchmarks with well-defined characteristics simplify this problem by allowing us to understand the behavior of the different architectural components in isolation, where we can independently quantify the effects of the mechanisms we want to study. Therefore, we develop a set of synthetic microbenchmarks that stress different parts of the POWER6 microarchitecture. Several of these microbenchmarks can be concurrently executed as MPI processes. This allows us to create multiprogrammed workloads and mix of workloads with different characteristics. We use six of these microbenchmarks in our evaluations: cpu_int, cpu_fp, ld_l1, ld_l2, ld_mem and st_mem.

The microbenchmarks are composed of a main loop that iterates for a configurable number of iterations. Table 1 describes the loop body for the ones used in this study. cpu_int mainly executes integer arithmetic operations accounting for 85.5% of the total executed instructions. cpu_fp is an FP-intensive benchmark, where 94% of the instructions are floating point arithmetic operations. ld_l1, ld_l2 and ld_mem are benchmarks mostly performing memory load operations (approximately 95% of the instructions are loads). They stress different levels of the memory hierarchy. ld_l1 always hits in L1 cache. ld_l2 always hits in L2 cache but always misses in L1 cache. Finally, ld_mem always misses in both L1 and L2, and brings the data from main memory. All of them perform a pointer traversal across an array which is configured to produce the desired memory access behavior. st_mem continuously performs memory writes missing in all levels of the cache hierarchy. This high rate of misses forces to evict lines from the L2 cache and effectively access the main memory.

Name	Loop Body
cpu_int	$a = a + (it * (it-1)) - x_i * it : x_i \in \{1, 2, \ldots, 54\}$
cpu_fp	Sequence of approximately 200 FP operations (fadd, fsub, fmul) using SW pipelining for increasing ILP.
ld_l1 ld_l2 ld_mem	`p = *p; // repeated 100 times`
st_mem	Memory store operations.

Table 1: Loop body of the different microbenchmarks.

We also use SPEC CPU2006 benchmarks [16] for our characterizations and to evaluate our power model. Other benchmarks aiming to stress system's I/O could have been used. However, I/O subcomponents are not typically energy-proportional and thus, their power consumption is approximately the same across different workloads [3]. We use IBM XL (XLC 10.1 and 12.1 XLF) to compile all the benchmarks (except for bwaves, gamess, zeusmp, tonto and xalancbmk, which are compiled with gcc 4.1.2 due to compile-time issues with XL).

4. RESULTS AND ANALYSIS

In this section we present and analyze our experimental results for the POWER6 power and thermal characterization. This characterization is divided into two parts. The first part analyzes the system when it is idle (no processes are running besides services and background processes). The second part studies the behavior of the system when it is under load.

4.1 Idle System Characterization

Several techniques have been proposed to reduce temperature and power consumption when the system is idle, both at hardware and software level. This section evaluates their effectiveness for our POWER6 system.

4.1.1 Low Power Mode

POWER6 employs several power reduction techniques for idle cores. Here we quantify the effects of these capabilities. Specifically we look at the effects of thread prioritization and enabling *nap* mode via cede_processor. We consider four power management policy combinations: *(i) No power saving* represents the baseline behavior without any power management. In this case all calls to cede_processor and HMT_xxx[1] have been disabled. *(ii) HMT enabled* only enables hardware thread prioritization. Enabling the calls to HMT_xxx allows the snooze loop to be executed with low priority. *(iii) CEDE enabled* only enables the calls to cede_processor so that the cores can go into nap mode, disabling the clock of most of the circuits inside the core. This policy does not manipulate hardware thread priorities. *(iv) Both enabled* enables all calls to both cede_processor and HMT_xxx, and thus represents the most aggressive power management policy.

Table 2 shows power and temperature characteristics observed for the idle system with these four policies. With

[1] Prioritization functions such as HMT_medium, HMT_low and HMT_very_low

		1	2	3	4
		No power saving	HMT enabled	CEDE enabled	Both enabled
Temp savings (%)	core 0	0	8.2	24.6	26.2
	core 1	0	6.8	22.0	23.7
	core 2	0	8.6	22.4	24.1
	core 3	0	9.4	21.9	23.4
Power savings (%)		0	8.7	23.3	24.3

Table 2: Temperature and power savings when the system is idle using different low-power-saving mechanisms in the processor. Values are normalized to the first configuration.

		tickless 100	non-tickless 100	tickless 1000	non-tickless 1000
total ticks/s		30	399	39	3993
temp increase %	core 0	0	0	0	2.3
	core 1	0	0	0	0
	core 2	0	0	0	2.3
	core 3	0	0	0	2.1
power increase %		0	0.46	0.46	2.75

Table 3: Timer interrupts for tickless and non-tickless kernel configurations (HZ=100 and HZ=1000). Power and temperature are normalized to the first configuration.

no power saving policy, all the cores reach the highest temperature and the highest total system power consumption. We will consider these values as the baseline for this section, showing the reduction compared to this baseline for the rest of configurations. *HMT enabled* mode considerably reduces the activity within the core and both power consumption and temperature are considerably reduced. The core temperatures and the system power decrease 7-9% and 8.7% respectively using only hardware thread prioritization. We see much more dramatic improvements with the *CEDE enabled* policy. In this case, although we prevent the processor from reducing thread priorities in the snooze loop, higher power savings are achieved by enabling POWER6 nap mode. Compared to the baseline configuration, the core temperatures and the system power consumption are reduced by 22-24% and 23.3%, respectively. Finally, applying both power management approaches in the *both enabled* policy further reduces system power consumption by 1%. This shows limited improvements for the idle system with hardware thread priorities when the nap mode is enabled. However, the nap mode can only be enabled when both threads in a core are idle, whereas hardware thread prioritization does not have such restrictions. When only one thread is idle, by using prioritization, more resources can be given to the other thread, increasing both performance and energy efficiency. Overall, combining nap mode and hardware thread prioritization significantly reduces the energy consumption when the processor is in idle mode.

4.1.2 Linux Tickless Kernel

In this section we measure the effect of the tickless mechanism on the temperature and power, monitoring interrupts and events on an idle system. As the system is idle, the number of *external* interrupts is negligible and thus, the system is mainly disrupted by *timer* interrupts (tick events).

We develop four kernel versions to evaluate the impact of the tickless mechanism, as shown in columns of Table 3. We build tickless and standard tick-based kernels with different tick rates (timer events per second) of 100Hz (default value for a server configuration) and 1000Hz. We measure idle core temperatures and system power in all these configurations. For this section and the rest of the paper we choose configuration 4 in Table 2 as the baseline for all power and temperature results. For that configuration the system is idle and both low-power mechanisms analyzed in the previous section are active, leading to the minimum power consumption and core temperature.

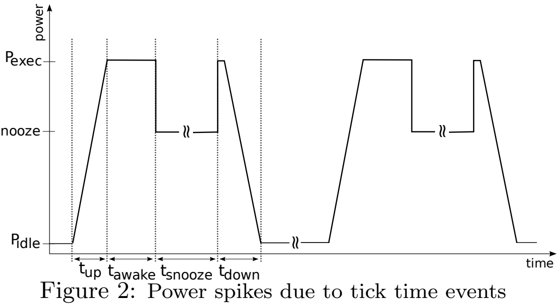


Figure 2: Power spikes due to tick time events

In terms of number of tick timer events, there is a significant difference (row 1 of Table 3). The number of events per second in a non-tickless system is much higher than in a tickless system, increasing by 13X (from 30 to 399) for the 100Hz kernel and 102X (from 39 to 3993) for 1000Hz kernel. These results show the effect of employing a tickless kernel, reducing the number of times that the cores have to wake up from their idle state to handle each of these interrupt requests.

Temperature and power results shown in Table 3 demonstrate the power and the thermal effects of the tickless kernel. It is interesting to notice that the first three configurations (tickless-100, non-tickless-100 and tickless-1000) do not show any significant variation. However, the last configuration, non-tickless-1000, has a power consumption 2.75% higher than the rest, with a slight increase in temperature. The reason for this increase is that the number of timer events per second is much higher in this configuration than the rest (10X compared to non-tickless-100 and 54X compared to tickless-1000). As the number of timer events per second grows, cores are more disrupted and cannot stay much in the nap mode. This is not as significant for the non-tickless-100 kernel due to the smaller number of ticks generated by the lower resolution timer.

Figure 2 depicts the interrupt timing behavior in more detail. Each of the spikes in the figure represents an expiration of the tick timer. When the system is idle it consumes P_{idle} (configuration 4 in Table 2) and on every tick timer expiration the following actions are carried out:

- The core wakes up from nap mode to active mode. This transition takes t_{up} μs. In [2] it is shown that t_{up} fits in the context switch delay, that is, in the order of few microseconds. Our results show that for the POWER6 processor, t_{up} equals 4 μs. As we have seen in Table 2 (configuration 1), during this period the system power consumption is the highest among the four configurations shown. P_{exec} represents the absolute power for that configuration.

- Once in active mode, we have to account for the time it takes the interrupt handler to run and to go from user mode to kernel mode and vice-versa, t_{awake}. In the interrupt handler, the OS checks whether there is any job to do. As shown in [11, 12] both steps take in the order of few microseconds (1-3μs). We assume 3 μs in this paper. During this period, the power consumption remains at P_{exec}.

- In an idle system most of the time the OS just continues in the idle loop and enters the snooze delay loop checking if a context switch is needed. As the hardware priority is reduced when entering the snooze delay loop, the system power consumption goes down to

P_{snooze} (Table 2 shows a 8.7% reduction over P_{exec}). This phase lasts for t_{snooze}, which by default is 100 μs in Linux. Changing the hardware priorities requires executing an OR operation, so we assume a delay of 0 μs.

- Finally, the system goes back to nap mode in a transition that takes t_{down} μs. Our results show that for the POWER6 processor, t_{down} equals 4 μs. During this period, the power consumption increases again up to P_{exec} and gradually decreases to P_{idle}.

The effect of ticks on the power consumption is represented by Equation 1, where t_{total} is the observation period and $\#ticks$ is the number of ticks occurred during that period.

$$P = \Big([(t_{up} + t_{awake} + t_{down}) \times (P_{exec} - P_{idle}) \\ + t_{snooze} \times (P_{snooze} - P_{idle})] \times \#ticks \qquad (1) \\ + t_{total} \times P_{idle} \Big) / t_{total}$$

We now apply Equation 1 to understand the low impact of the tickless mechanism (especially for HZ=100). In the previous section, Table 2 displays the power consumption for the idle loop using different configurations. These measurements are conducted when all four cores are in the same state, therefore for the rest of this analysis we will assume that all cores treat the tick-timer expiration at the same time. If we considered expirations independently, their number would be higher but system power consumption would be significantly lower as only one core would be active at a time. Thus, both analysis would lead to very similar results.

For non-tickless-100 we have approximately 100 tick-timer expirations per second in the whole system. Using Equation 1, the computed power consumption under this scenario is 0.24% over P_{idle}. For the case of non-tickless-1000, there are approximately 1000 wake-ups per second, which leads to a power consumption of 2.3% over the baseline. Both results are close in absolute terms to the actual measurements in Table 3.

Overall, we conclude that the tickless mechanism does not significantly reduce the power consumption for a standard tick resolution (HZ=100) as the number of times the cores exit the nap mode is not enough to noticeably increase the power consumption during the period of one second. This observation may change for other systems with the following characteristics: 1) the time to go from/to the low-power (t_{up} and t_{down}) mode is high. This may happen in processors in which low-power modes introduce changes in the supply voltage, in which case t_{up} can be much higher; 2) the difference between P_{exec} and P_{idle} is high; or 3) t_{snooze} is relatively long. Such formulation of the interrupt behavior can help evaluate different kernel configurations for the POWER6 system without the need to deploy them in an actual system.

4.2 System Under Load Characterization

Next, we analyze the power and thermal behavior of our POWER6 system when it is under varying load levels. We also demonstrate the impact of dynamically varying the number of active cores on power consumption. In this section, because of space constraints, we use a subset of SPEC CPU2006 with distinct representative behavior (e.g., integer/floating point, high/low IPC, high/low memory access count, etc.).

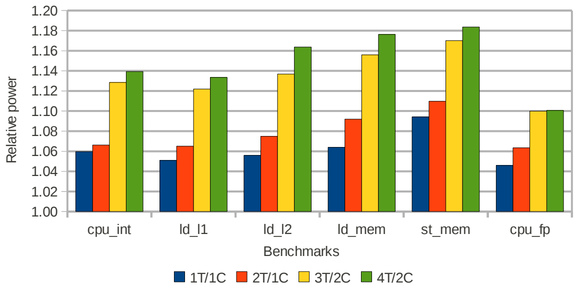


Figure 3: METbench power consumption for different number of threads (T) and cores (C).

	cpu_int, cpu_fp	cpu_int,ld_l1	cpu_int,ld_mem
Cores	1	1	1
T_{avg} (%)	18.2	19.6	17.1
P_{avg} (%)	6.6	7.2	8.2
Aggregated Performance Counters			
IPC	1.77	1.56	1.31
L1 MPKC	0.00	0.00	1.97
L2 LD MPKC	0.00	0.00	1.95
L2 ST MPKC	0.00	0.00	0.00

Table 5: METbench temperature/power results for 2 threads (mixed workloads). Power and temperature are relative to the idle system.

4.2.1 *Effect of Workload Characteristics on Power and Thermal Behavior*

It is well established that power and thermal behavior of computing systems strongly depend on dynamic characteristics of the running workloads. To characterize the effect of workload characteristics on POWER6, we conduct several experiments with different applications from METbench and SPEC CPU2006 benchmark suites. While, in general, power and thermal behavior change with the amount of activity in the system, there is not a single characteristic factor that directly reflects the power consumption of the system. It is rather a combination of application features such as its IPC and memory intensity. We present measured power and thermal characteristics for METbench and SPEC CPU2006 benchmarks in Table 4. The table shows measured average temperature, T_{avg} (percentage over the baseline), average system power, P_{avg} (percentage over the baseline), IPC, L1 misses per kilo-cycle (L1 MPKC), and L2 load and store misses per kilo-cycle (L2 LD MPKC and L2 ST MPKC) for each benchmark.

The results presented in Table 4 show the strong influence of different workload characteristics on power and thermal behavior. We observe strong deviations among benchmarks in terms of their power and thermal behavior and their associated performance metrics. Below we look at specific benchmark categories and derive the relations between major workload features and their impact on power and temperature.

CPU-bounded benchmarks: We see that high-IPC and highly-CPU-bounded benchmarks generally lead to higher core temperatures. Among the METbench microbenchmarks, cpu_int has the highest IPC and a core temperature that is 7-9% higher than the other microbenchmarks. Within SPEC CPU2006, the benchmarks that cause higher core temperatures are h264ref, bzip2 and cactusADM. These three benchmarks also present the highest IPC among the SPEC CPU2006 benchmarks used in this work [2].

Memory-bounded benchmarks: Interestingly, while

[2] METbench microbenchmarks are designed to exercise a single resource in the system at a time. In contrast, SPEC CPU2006 stress different parts of the system at once.

	cpu_int	ld_l1	ld_l2	ld_mem	st_mem	cpu_fp	h264ref	bzip2	gcc	dealII	lbm	cactusADM	mcf	milc	soplex
Types	INT	INT	INT	INT	INT	FP	INT	INT	INT	FP	FP	FP	INT	FP	FP
T_{avg} (%)	19.8	12.5	13.0	10.2	14.6	10.2	22.7	20.7	16.8	19.8	15.5	21.4	14.8	15.5	15.5
P_{avg} (%)	6.0	5.1	5.6	6.4	9.4	4.1	7.8	7.4	7.3	7.6	13.1	10.0	7.7	9.4	8.3
Aggregated Performance Counters															
IPC	1.32	0.26	0.034	0.0020	0.018	0.47	1.16	0.79	0.44	0.66	0.39	0.85	0.12	0.19	0.32
L1 MPKC	0.0	0.0	32.5	1.94	3.62	0.0	11.0	8.9	5.8	5.9	29.5	29.3	5.58	8.3	8.2
L2 LD MPKC	0.0	0.0	0.0	1.94	0.0	0.0	0.00	0.05	0.66	0.25	0.25	0.06	1.20	1.84	0.95
L2 ST MPKC	0.0	0.0	0.0	0.0	3.61	0.0	0.02	0.13	0.16	0.02	5.6	0.51	0.05	0.46	0.37

Table 4: METbench and SPEC CPU2006 temperature/power results when executing 1 thread. Power and temperature are relative to the measured values when the system is idle.

CPU-bounded benchmarks achieve higher temperatures, they do not consume the most power. As Table 4 shows, memory intensive benchmarks generally consume more power. This is because of the accesses to main memory, which carry significant power cost. Among the microbenchmarks, ld_mem and especially st_mem are the workloads with the highest power consumption. Although ld_mem power consumption does not differ significantly from the other microbenchmarks when only one process is used, Figure 3 shows the increasing power gap with increasing number of threads. For the SPEC CPU2006 benchmarks we see a similar trend. Memory-intensive benchmarks like milc and, especially, lbm consume more power than the rest of the benchmarks. For instance, relative to the baseline, lbm consumes 5.3% more than h264ref, with significantly lower temperature in comparison. The core temperatures are generally low for memory-intensive benchmarks as they spend most of the time waiting for data from the main memory.

mcf is a low-IPC benchmark with a considerable amount of L2 cache misses per kilo-cycle and with similar characteristics to milc. However, the power consumption of mcf is considerably smaller (1.7% less). The most significant difference between them is the number of L2 store misses per kilo-cycle, which is 10X higher for milc. As we have seen before in Figure 3, accessing main memory because of a store operation leads to a higher power consumption. Accordingly, lbm, which has the highest number of L2 store misses, also shows the highest power consumption among the evaluated benchmarks.

FP benchmarks: An interesting application in this category is cpu_fp. Despite having a medium IPC (0.47), it achieves the lowest core temperature. We believe that the reason for this behavior is related to the fact that the floating point unit (FPU) occupies a bigger area than other structures. Therefore, a fully utilized FPU has a lower power density than other fully-utilized parts of the core, such as the fixed point unit (FXU), which is stressed by cpu_int. Hence cpu_fp yields a lower temperature.

Comparing the execution of METbench and SPEC CPU2006, we also notice that SPEC applications tend to consume more power and reach higher temperatures. METbench microbenchmarks are very specific and they "light up" fewer parts of the processor than SPEC CPU2006 benchmarks do.

In Table 5 we also look at the impact of heterogeneous workload mixes. Here, we see that co-scheduling a computation-intensive benchmark and a memory-intensive one leads to both high core temperatures and a high power consumption. For example, considering cpu_int and ld_mem, one thread continuously performs arithmetic operations while the other exercises the memory subsystem. In contrast, a more homogeneous mix, such as cpu_int and cpu_fp leads to lower power consumption. In this case the core temperature is higher as the core is more stressed.

Figure 3 also alludes to an important characteristic of the POWER6 processor. As we increase the number of used cores from one to two, we see a significant jump in power consumption. This is due to the fact that a second core has to exit the nap mode to serve the threads. We demonstrate in the following sections that every core that leaves the nap mode adds a constant power increment of approximately 5% to the system power consumption. We will refer to this increment as P_{AC} in the following sections.

Using the relations derived in this section, we develop a power model for the POWER6 system in Section 5.

4.2.2 *Effect of Core Usage on Power and Thermal Behavior*

In this section we execute several copies of a microbenchmark from the METbench suite in an incremental way. First we execute 2 copies on contexts 0 and 1 (one core), then 4 copies on threads 0, 1, 2 and 3 (two cores), and so on until using the 8 threads (four cores). We name each of these steps an *execution step*. Each execution step is roughly 9 minutes long (360 iterations[3]), considerably longer than the granularity that we are allowed to read the power information.

cpu_int: Figure 4 shows the results with cpu_int. In the figure we notice that power remains quite stable in the intervals between execution steps. Power noticeably increases when two more copies of cpu_int are started and a new core is used. We observe the first increment around one minute after the program is started. This is because of the 1min access granularity of the TPMD for power measurements.

From this experiment we estimate that for every two new copies of cpu_int that are running on the system, the system power consumption increases approximately 7.6%. Another observation is the interaction between cores within the same chip. In the figure, around $t = 50$ seconds, we see that the temperature of core 1 increases approximately 8% when core 0 starts executing the benchmark. Later, around $t = 600$ seconds the temperature of core 0 further increases 7% when core 1 starts running. This is due to the lateral heat conduction between the cores within the chip. On the other hand, as the two chips are physically separated, we do not see any inter-chip effect in temperature.

As cpu_int is not using any shared resources between the cores, the aggregated throughput does not reduce as we increase the number of used cores. This can be seen in Table 6a. The IPC is stable around 0.85 per thread and the aggregated throughput increases linearly with the number of threads being executed.

ld_mem: This benchmark continuously executes load instructions always missing in all levels of the cache hierarchy. Therefore, it always needs to go to main memory to get the data. As shown in Table 6b, its IPC is much less than for cpu_int. As we have previously stated, memory-intensive workloads typically consume more power than computation-intensive loads. This can be seen again comparing the incre-

Therefore, some SPEC CPU2006 benchmarks consume more power than METbench.

[3] METbench can iterate a benchmark a certain number of times in order to obtain better stability in the results.

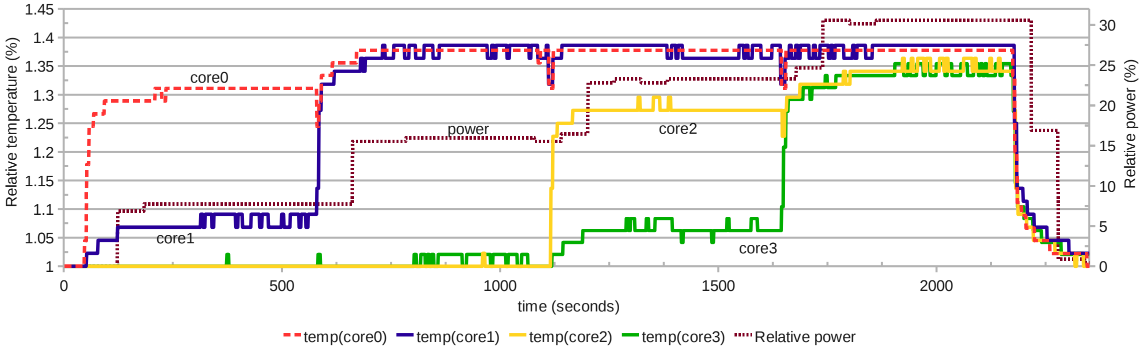


Figure 4: Several copies of cpu_int are used to create an incremental execution (2, 4, 6 and 8 hardware threads). The values are relative to the power and temperature measurements when the system is idle.

#threads	IPC per thread								aggregated IPC	P_{avg} (%)
	chip0				chip1					
	core0		core1		core2		core3			
	0	1	2	3	4	5	6	7		
2	0.8482	0.8483							1.6965	8.3
4	0.8480	0.8480	0.8489	0.8489					3.3938	16.5
6	0.8478	0.8478	0.8489	0.8489	0.8485	0.8485			5.0904	23.9
8	0.8480	0.8480	0.8481	0.8481	0.8487	0.8487	0.8486	0.8486	6.7868	31.2

(a) cpu_int

#threads	IPC per thread								aggregated IPC	P_{avg} (%)
	chip0				chip1					
	core0		core1		core2		core3			
	0	1	2	3	4	5	6	7		
2	0.0017	0.0017							0.0034	10.6
4	0.0011	0.0011	0.0011	0.0011					0.0044	17.0
6	0.0010	0.0010	0.0010	0.0010	0.0016	0.0016			0.0072	26.2
8	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011	0.0088	33.0

(b) ld_mem

Table 6: IPC, aggregated IPC and power consumption for the incremental execution of multiple processes. The power consumption values are normalized with respect to the ones obtained when the system is idle.

mental executions of cpu_int and ld_mem where power consumption is always higher for the latter in every execution step.

It is interesting to notice the reduction in IPC as more ld_mem threads are run. For instance, by comparing the cases where two threads (on the same core) and four threads (on the same chip) are executed on the system, the IPC for the first thread in the core 0 decreases approximately 36%. This suggests that there is contention in the shared hardware resources between cores. Looking at the results for six threads, we observe that the IPC for contexts four and five is approximately the same as it was in the case of two threads for contexts zero and one. The drop in IPC occurs within a chip when going from two to four contexts. Thus, the contention occurs within the chip, probably in the SMP interconnect fabric, as both L1 and L2 cache are private to each core and each core has its own memory controller and channels interfacing the main memory.

4.2.3 Effect of Hardware Thread Priorities on Power Consumption

It has been previously demonstrated that the hardware prioritization mechanism in POWER processors can improve system throughput [4]. Here we look at hardware prioritization from a power management angle. We show the effect of applying this mechanism in a power-aware manner and present use cases where thread prioritization can improve not only system throughput, but also system power consumption. Although there are multiple priority levels in POWER6, we present only a subset of them, as we are more interested in showing their possible use to improve energy efficiency, rather than doing an extensive characterization.

In Section 4.1.1 we showed that by using hardware thread prioritization, the power consumption for an idle system can be reduced up to 9%. In this case, since the system was solely running the idle loop, performance was not a major concern. In the case of a system that is executing workloads, hardware thread prioritization cannot be blindly used to reduce power consumption in a performance-agnostic manner. Careful consideration of power-performance trade-offs is needed to choose the appropriate priority levels. We show that by exploiting workload characteristics, we can use hardware thread prioritization to reduce power consumption and increase system throughput.

Table 7a shows the results of executing a high-IPC application (h264ref) together with a low-IPC, memory-intensive one (lbm). With the standard priority configuration, (4,4), the system has a power consumption 16% over the baseline. lbm is the main contributor to that consumption. If the priority configuration is changed to (5,4), so that the priority of the high-IPC workload is increased, the system power consumption is slightly reduced as less memory requests are performed by lbm. Moreover, the aggregated IPC is increased as more computational resources are given to h264ref, thus obtaining a better relative energy-delay product (EDP) [5]. It is important to notice that in this case the individual IPC for lbm is not drastically reduced (approximately by 11%). In the most extreme configuration (6,1) the power is further reduced and the performance is increased again. However, this comes at the expense of significantly reducing the performance of the memory-intensive workload (lbm).

Priorities	3,4	4,4	5,4	6,1
IPC				
h264ref	0.32	0.55	0.72	1.15
lbm	0.36	0.35	0.31	0.01
Aggregated	0.68	0.9	1.03	1.16
P_{avg} (%)	15.1	16.1	15.1	8.7
EDP (relative)	1.73	1	0.75	0.56
ED^2P (relative)	2.29	1	0.65	0.43

(a) Mixed workload (h264ref and lbm)

Benchmarks	cpu_int		ld_mem	
Priorities	1,1	4,4	1,1	4,4
Aggr. IPC	0.07	1.80	0.0030	0.0034
P_{avg} (%)	3.9	6.9	8.7	9.6
EDP (relative)	642.9	1	1.2	1
ED^2P (relative)	16508.8	1	1.5	1

(b) Effect of priority (1,1)

Table 7: Power results using prioritization for a single core. Power values are normalized to consumption when idle. EDP and ED^2P normalized to configuration (4,4).

We consider *priority one* as a special case for power management. Table 7b characterizes the effects of this priority mode, where it shows the results of executing a CPU-bound (cpu_int) and a memory-bound (ld_mem) workload with priorities (4,4) and (1,1). We notice that the effect of hardware thread prioritization depends on the characteristics of the workload. For instance, running ld_mem with priority (1,1) does not significantly affect its IPC, as it is an extreme low-IPC memory-bound benchmark. The power consumption is also not significantly affected as this benchmark consumes most of the power in the memory subsystem. For cpu_int, a high-IPC workload, the behavior is completely different. The power consumption is decreased 3%, at the expense of reducing the IPC from 1.8 to 0.07. In general, the higher the core activity, the higher the power reduction obtained with priority one and the higher the performance impact.

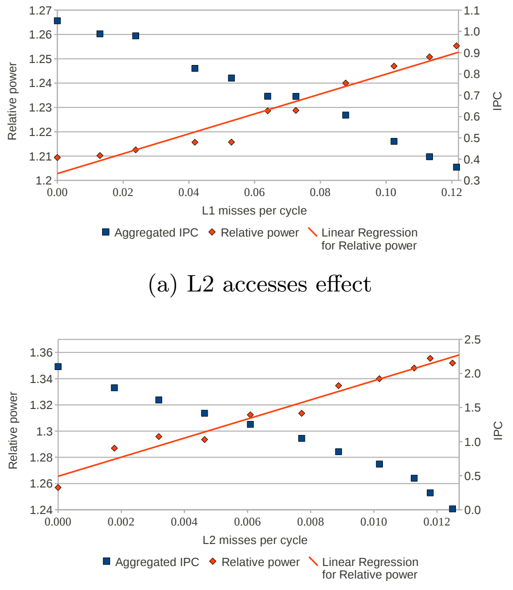
One major advantage presented by this priority-based power-performance management scheme is the ability to make "small" changes to the system behavior to achieve desired power-performance targets. Unlike most dynamic adaptation schemes that expose drastically different operating points, the prioritization based approach can provide small shifts in power and performance with very small impact to runtime behavior. Another advantage of this mechanism is its very short latency until the applied power management actions take effect. The response time of this mechanism is dramatically faster compared to external mechanisms such as dynamic voltage and frequency scaling (DVFS). Therefore, hardware thread prioritization can be used as a fast and flexible initial response in the case of a thermal/power emergency [10].

5. APPLICATION OF THE RESULTS

In this section we apply some of the learning from the characterization work that is presented in the previous sections. Section 5.1 discusses an analytical power model that relies on performance counters. Section 5.2 analyzes the impact of the power and thermal behavior of the system on the OS scheduler.

5.1 Power model

The possibility to obtain temperature and power measurements is a useful feature that is provided by POWER6 based systems. However, some configurations may not include the external microcontroller responsible to obtain these measurements (TPMD). Moreover, in some systems, it may not be possible to access the console that provides the temperature and power measurements. Typically, the console is password protected and plain users do not have access to it. Clearly, it is beneficial for users to understand the power and thermal behavior of their applications. In another context, since direct access to TPMD from the OS is not possible, OS cannot use power and thermal information to improve its decisions such as better scheduling in terms of power consumption.



(a) L2 accesses effect

(b) Memory accesses effect

Figure 5: Effect of accesses to L2 cache and main memory on system power consumption. Power values are relative to the idle system consumption. Several instances of the same benchmark are executed to create a higher power delta. The regression line is just an approximation to show the increasing trend.

In this section, we present a model based on performance counter (PMC) data to estimate power consumption of the system that is under study. Since performance counter data is available and accessible by the OS, an analytical model in this form can alleviate all the shortcomings highlighted previously. This model follows a similar thinking as presented in [3]. To the best of our knowledge, this is the first simple analytical power consumption model for a POWER6-based system. The model presents a good accuracy and it only relies on the performance counters as the hardware support. Moreover, since the set of performance counters required is minimal, it is easier to implement it in run time systems that take decisions based on performance counters data.

Similar to [3] we select a group of PMC that captures the activity in different components of the system such as CPU, memory, disk etc. In our case, we concentrate on the CPU and memory parts since, as shown in [3], there is not a significant variation in power consumption due to activity in the other components (95% of the dynamic power consumption is due to activity in CPU and memory). The selection of the right set of performance counters for the model relies on a hybrid scheme, where expert knowledge and pruning techniques based on statistical analysis are utilized. The scheme leads to a set of performance counters that obtain significant accuracy in prediction POWER6 system power consumption.

The power consumption due to activity in the chip is modeled by using IPC and the number of L1 load misses per cycle (L1LDMPC). The memory system contribution to the power consumption is modeled by using the number of L2 misses per cycle (L2LDMPC and L2STMPC). As the system does not have a L3 cache, every miss to the second level cache goes to the main memory, thus L2 misses per cycle are good indicators of memory power consumption. In [3], authors do not differentiate between load and store misses. However, as discussed in Section 4.2.1, in Table 4, benchmarks with a high count of L2 store misses consume more power than other type of workloads. Thus, the analytical model includes L2 store misses to improve accuracy.

As we only have total system power consumption, it is important to understand the power behavior of different components and whether a linear model of those components is sufficient. For this purpose, we use two microbenchmarks which can vary the miss rate both for L1 and L2 from zero to cache/memory saturation point. Figure 5a displays the power consumption variation as the L1 miss ratio grows which provides an insight on the L2 cache power contribution to the system power consumption. Figure 5b shows a similar information for L2 misses reflecting the memory power contribution to the system power. In both figures, we observe that the power consumption values grow linearly as the number of misses increase. Thus, we define the model as a linear combination of these different factors that contribute to the power consumption.

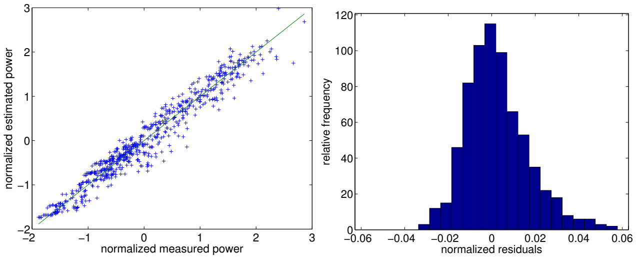
Equation 2 shows the model with its several components that account for the total power consumption of the system. The power is predicted as a percentage over the baseline when the system is idle (i.e., no user-process is being executed and the cores spend most of the time in the nap mode). From the characterization step in Section 4.2.1, we observe that for each core that exits the nap mode there is an increment in the power consumption (P_{AC}). N_{AC} is the number of active cores, so multiplying it by P_{AC} gives the power consumption of all the cores in the system which are not in nap mode.

$$\begin{aligned} P = N_{AC} \times P_{AC} + \alpha \times IPC + \beta \times L1LDMPC \\ + \gamma \times L2LDMPC + \sigma \times L2STMPC \end{aligned} \tag{2}$$

We conduct several descriptive statistic tests for the parameters in the data set (e.g., normality test for residuals, and non-presence of non-random patterns in the residuals). We also look at the significance of the parameters and their correlation to the response variable.

It is important to note that the coefficients that are found by regression are subject to change if size or type of the components of the system are changed (e.g., memory). Motivated by this fact, we follow two different approaches to train the model. (i) The first model (*METbench training*) creation efforts rely on METbench data to train the model. Since METbench runs five time faster than SPEC CPU2006, the amount of time to collect training data for a new model is considerably reduced. Thus, we only use METbench results to train the model, and we test the model with SPEC CPU2006 data. In the case of a new model requirement for a different system configuration, we can do it quickly by just using METbench to collect the data and later to train the model. (ii) The second model (*shared training*) creation effort combines all the data (METbench and SPEC CPU2006) into a pool, and then relies on this dataset to train the model. A more general and accurate model is possible when a heterogeneous set of workloads are used. To cross-validate the model, we use *leave-one-out cross-validation* technique. Leave-one-out cross-validation is a standard statistic technique to estimate the accuracy of a regression model [15].

METbench training. Figure 6 shows the relative error in percent between our model estimation and the actual power measurement. Different bars for a benchmark correspond to different CMP configurations (1T1C, 2T2C and 4T4C), and hybrid CMP+SMT ones (2T1C and 4T2C). As shown in Figure 6, most of the benchmarks are predicted with an error equal or less than 5%. The relative error for 1T1C configuration is below 1% for almost all benchmarks. The maximum error occurs for cactusADM when it is run as four processes in SMT mode. We compute the average error using the geometrical mean. The error is under 4% for all the configurations.



(a) (b)

Figure 7: Model validation using all the available data (METbench and SPEC CPU2006). (a) shows the normalized measured vs. estimated power. The values are normalized by subtracting the mean of the error and dividing by the standard deviation of the error. (b) shows the residue error distribution with the cross-validation process. The residuals are normalized by dividing to the actual measure value.

In general, the estimation error increases as more processes run on the system. We observe this effect for both CPU-bound and memory-bound workloads. We attribute it to the accumulation of the errors that are made to predict the power consumption for each core in the system. When both SMT and CMP capabilities are used, the estimation error grows with respect to the CMP case. Specifically, the error is more evident in high-IPC workloads such as h264ref, cactusADM and dealII. These benchmarks present a higher degree of interaction when they are co-scheduled on the same core. For memory-intensive benchmarks such as lbm, mcf and milc there is no significant error increase when both SMT and CMP capabilities are used. Nonetheless, the average error for the CMP+SMT case is between 2.5% and 5% for two and four processes, respectively.

Shared training. By combining data from METbench and SPEC CPU2006 to train the model, we capture wider resource usage patterns, and thus we obtain a model that could potentially predict unobserved data points in a more accurate way. Figure 7a shows the normalized measured vs. estimated power consumption. Model predictions are considerably close to the real measurements for most of the data points. The residual distribution with the cross-validation process, shown in Figure 7b, resembles a normal distribution, with mean, $\mu = -7.2 \cdot 10^{-5}$, and only 4.6% of the individuals are out of the confidence interval $[\mu - 2\sigma, \mu + 2\sigma]$. The error is under 6% for all the cross-validation steps.

Overall, both of the approaches that are used to construct the model obtain quite accurate results, with errors less than 6%. This level of accuracy is sufficient for users to study the power consumption behavior of their applications. In addition, this level of accuracy is also attractive for OS to implement such a model to deploy optimization policies.

5.2 Thread Placement

With the arrival of SMT and CMP architectures, ensuring fairness between the different running processes has become an important issue. Several techniques such as scheduling

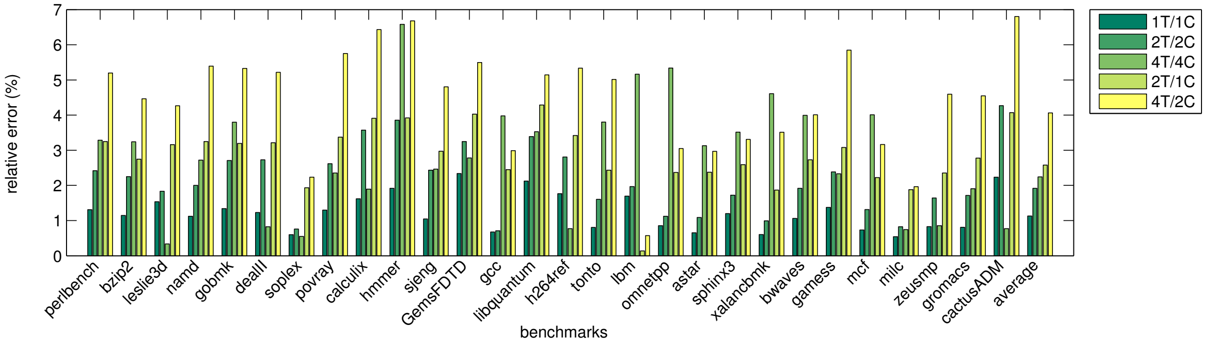


Figure 6: Estimation accuracy for the power model trained with METbench data only, for different number of threads (T) and cores (C). For instance, 4T/2C means 4 threads are run on 2 cores (using SMT capabilities). The error is computed as: $\frac{|measured-predicted|}{measured} \times 100$.

	1	2	3	4	5	6
Pattern	1100 0000	1010 0000	1000 1000	1111 0000	1100 1100	1010 1010
HW threads Cores	2 1	2 2	2 2	4 2	4 2	4 4
P_{avg} (%)	9.6	13.3	12.8	20.2	19.3	29.4
IPC	1.75	2.33	2.34	3.51	3.51	4.68
EDP	1.74	1.01	1	1.65	1.64	1
ED^2P	2.32	1.02	1	2.18	2.14	1

(a) h264ref

	1	2	3	4	5	6
Pattern	1100 0000	1010 0000	1000 1000	1111 0000	1100 1100	1010 1010
HW threads Cores	2 1	2 2	2 2	4 2	4 2	4 4
P_{avg} (%)	15.1	17.9	22.0	22.0	29.4	34.9
IPC	0.41	0.44	0.76	0.42	0.83	0.88
EDP	3.24	2.88	1	3.97	1.08	1
ED^2P	6.01	4.98	1	8.33	1.14	1

(b) lbm

Table 8: Effect of core configurations on power and performance. Power is normalized to the idle power. EDP and ED^2P values are normalized to the best configuration within each group (2 or 4 threads).

domains, load balancing and cache affinity have been implemented in actual operating systems.

Job scheduling techniques have also been used in order to reduce power consumption. For instance, Linux provides a setting, sched_mc_power_savings, that attempts to save power consumption by grouping several processes into a single chip, therefore leaving other chips idle. An analogous setting, sched_smt_power_savings, exists to consolidate several processes into a single core [28].

In this section we study the effect of thread placement on the power consumption. Given a set of processes, there are different possible ways of assigning them to hardware threads, considerably varying the impact on power and performance. In order to analyze this impact we conduct several experiments where multiple processes are executed with different core usage patterns. The second row in Tables 8a and 8b shows the core usage pattern used. For instance, the binary pattern 1000 1000 means that the first hardware thread in the first core in every chip is used to execute one process.

CPU-bounded workload: Table 8a shows the effect of thread placement for 2 and 4 instances of the CPU-bounded benchmark h264ref. The first thing we notice is that SMT configurations (columns 1, 4 and 5) present lower power consumption with respect to the other scheduling options using the same number of threads. For example, the configuration on column 1 reduces power consumption by 3.2% $(1-\frac{1.096}{1.133})$ with respect to the configuration in column 2. Analogously configuration 5 is 7.8% better than configuration 6. However, as h264ref is CPU-bound, running both processes in SMT mode on the same core affects the performance (24.9% and 25%, respectively). The energy-delay product is worse for these configurations as the small power reduction does not make up for the loss in performance. Similar conclusions were obtained in [19].

More interestingly, the power consumption remains the same between using 2 cores in a single chip (configuration 2) and using one core in each chip (configuration 3). We expect that in configuration 2, the second chip would be in low power mode most of the time, leading to a power consumption reduction. However, the POWER6 saves power at the core level, without any extra reduction when a whole chip is idle. Therefore, what really matters is the number of idle cores and not whether they are in the same chip or not. The same behavior can be observed when using 4 threads in configurations 4 and 5. If the processor were able to reduce the power consumption when a whole chip is idle, it would certainly be possible to consolidate several processes into one chip in order to reduce total energy consumption.

Memory-bounded workload: For memory-intensive workloads the situation clearly changes. As they are not bounded by the pipeline resources, executing 2 threads on the same core in SMT mode does not significantly hurt the performance. Comparing the IPC for configurations 1 and 2 in Table 8b, we observe that the IPC reduces only by 6.8% (from 0.44 to 0.41). The same behavior is observed for configurations 5 and 6, where four threads are run and the IPC decreases by 5.7%.

lbm is a memory-intensive application and it saturates the memory bandwidth of the first chip, as we saw before in the incremental execution of ld_mem (Section 4.2.2). As each chip has two memory channels (one per core), distributing the processes across both chips will better use the available bandwidth to memory, compared to consolidating them into one chip. In Table 8b we can observe that the performance nearly doubles when we go from single chip configurations (1, 2 and 4) to double chip ones (3, 5 and 6).

Effects on scheduling: Recent versions of Linux use scheduling domains for representing the CPUs hierarchy with a tree-based shape. In our system, at the first level there are different chips or packages in the system. The second level has the cores belonging to the chips from the previous level. Finally, the third level contains the HW threads or contexts for every core.

When using the default behavior, the Linux scheduler tries to distribute the threads throughout all the cores in the sys-

	1	2	3	4	5	6
Pattern	H0L0 0000	H000 L000	L0L0 H000	L0H0 L000	H0H0 L0L0	H0L0 H0L0
HW threads	2	2	3	3	4	4
Cores	2	2	3	3	4	4
P_{avg} (%)	21.1	19.7	26.2	26.6	34.9	32.6
IPC	1.54	1.56	1.60	1.93	2.78	3.06
EDP	1.05	1	1.45	1	1.23	1
ED^2P	1.07	1	1.74	1	1.36	1

Table 9: Effect of core configurations for a mixed configuration (h264ref and lbm). Power is normalized to the idle power. EDP and ED^2P values are normalized to the best configuration within each group (2, 3 or 4 threads).

tem, avoiding to run two threads on the same core unless it is not possible (i.e., there are more running threads than cores in the system). As we have seen, running two threads in SMT mode is not very efficient mainly when the threads are CPU-bound. Linux prevents putting threads into the same core as long as there are free ones available. However, if the sched_smt_power_savings flag is active, Linux will group processes degrading overall performance and energy efficiency. We have also seen that when using processors with power-saving techniques at the core level, grouping the threads in the same chip, leaving other idle, introduces no benefit. This is due to the fact that what really matters is the number of active cores. In this case, sched_mc_power_savings would not lead to a power reduction.

In terms of performance and energy efficiency, we analyze the effect of grouping the threads into a single core/chip. In general, the major source of slowdown between threads is sharing the caches. In our setup, the L2 is private so threads do not suffer any slowdown, due to the cache, whether they are placed on different chips or on the same chip. However, we have seen that there are other resources shared at the chip level that have to be taken into account for memory-bound threads. In this scenario, multi-chip configurations are much more efficient in terms of energy-delay product with reductions up to 2.9X (configuration 3 vs. 2) and 3.7X (configuration 4 vs. 5) as shown in Table 8b. Thus, the decision on whether to consolidate tasks into the same core/chip cannot be static. It depends on the low-power capabilities of the underlying architecture and the characteristics of the application.

Mixed workload: A scheduler that is aware of the workload characteristics can use this information to increase the system performance and/or reduce the power consumption. Table 9 shows the results of executing a mixed workload consisting of several h264ref and lbm processes [4]. Comparing configurations 5 and 6 we observe that the latter is a heterogeneous workload mix at the chip level (each chip executes a CPU-bound and a memory-bound workload), whereas the former is a homogeneous mix at the chip level. This will affect both performance and power consumption. The performance of configuration 6 is 10% better and the power consumption is 2.3% less. This leads to a 18.7% improvement in EDP and 26.3% in ED^2P.

An even more noticeable situation is seen in configurations 3 and 4. As in the previous case, placing both memory-bound workloads on the same chip limits their performance, without decreasing the total system power consumption. Thus, by co-scheduling the high-IPC and the memory-intensive workloads on the same chip we can reduce the interference between them, boosting the performance and reducing the energy consumption (1.7X improvement in the ED^2P).

Effects on scheduling: Current implementation of the Linux scheduler does not take into account workload characteristics. This means that the scheduler may fail to achieve the optimal performance and/or the minimum energy consumption. For instance in Table 8b the scheduler may choose either configuration 2 or 3, as none of them uses SMT. If the former configuration is chosen, a 5X ED^2P deterioration will be experienced. In Table 9 the scheduler may choose either configuration 3 or 4, leading to a 1.7X ED^2P worsening. These results show the importance of considering the workload characteristics and interaction in order to take more efficient scheduling decisions.

6. RELATED WORK

Several papers focus on the energy/thermal power behavior of CMP/SMT processors. However, either they do not consider a hybrid CMP/SMT processor, like the POWER6 [3, 21, 13] or they use a simulation framework [25, 19] or they only consider temperature but not power consumption [7]. To the best of our knowledge, we show the first characterization of a real hybrid CMP/SMT implementation.

There are different studies that characterize and propose techniques to improve power consumption and to reduce the temperature on real machines. Choi et al. [7] propose a thermal-aware task scheduler for the IBM POWER5. They use heat slack in order to reduce the temperature by using thread migration. Hanson et al. [14, 13] conduct a temperature and power characterization on an Intel Pentium M processor. They also create a runtime system which monitor the temperature on the system and tries to maximize the performance while ensuring the system is working under safe power and thermal constraints. Kursun et al. [17] characterize how process variation affects the temperature on the processor. They also develop a scheduling technique that aims to reduce the possible hotspots created by process variation.

Using performance counters to estimate power consumption on a system is an active topic where different studies exist. Pusukuri et. al [24] propose a simple model that uses cycle count and L3 misses to estimate power consumption on AMD processors. Powell et al. [23] develop a technique that estimates power consumption for the different parts of a processor. They do so by correlating activity in these parts with the power consumption using a few performance counters. Bircher and John [3] use performance counters to estimate power consumption for the whole system including memory, chipset, I/O, disk and the processor. The model for the IBM POWER6 processor presented in this work is similar to the last one mentioned.

Fan et al. [9] perform a study on power consumption in a Google's data center. They show that servers spend a considerable percentage of the time in idle mode between work requests. They also state that if the idle power of a system can be reduced down to 10% of its peak power consumption, the energy consumption can be reduced by 50%. Therefore, initiatives like Linux tickless are very useful in order to create systems that are more energy-efficient. In the same direction, Meisner et al. [20] try to bring the power consumption close to zero when the system is idle by switching off all the non-critical components in the system until a new work request arrives. In addition, they compare their approach to DVFS.

7. CONCLUSIONS

In this work we present a power and thermal characterization for a multichip POWER6-based system comprising two chips. We characterize the power consumption and the

[4] In this case, the patterns are composed of Hs and Ls, standing for h264ref and lbm, respectively.

thermal behavior both when the system is idle and when it is under load. Several levels are analyzed: hardware, operating system and application.

Our results show that 1) At the hardware level: workload-aware manipulation of thread priorities improves the system's energy-delay product by as much as 25%. The combined effect of employing the nap mode and hardware thread priorities show very significant benefits, reducing 24% both the core temperatures and the system power consumption. The increase in power consumption for one active core in the system is approximately 7.6%. When a core is under load its temperature increases which also affects the temperature of its sibling core, up to 8%. 2) At the OS level: The tickless mechanism does not significantly reduce the power consumption for a standard tick resolution (HZ=100) as the number of times the cores exit the nap mode is not enough to increase the power consumption. 3) At the application level: We show that the most relevant factor affecting core temperature is how much computing-intensive a workload is. Typically, a workload with high IPC will increase core temperature more than a low IPC one. A difference up to 22.7% can exist within a single core. However, the total system consumption is not dominated by the workload IPC. Memory intensive workloads consume more power than high-IPC ones. Workloads resulting from the combination of memory-intensive and computing-intensive benchmarks are the ones which lead to a higher power consumption and core temperature.

From the characterization study and using the information provided by the performance counters we build a power consumption model for the whole system. Moreover, by using microbenchmarks we create this model by combining the CPU and the memory subsystem power consumption. We test the accuracy of the model by running a set of SPEC CPU2006 benchmarks and comparing the estimated power consumption with the actual measurements. The results show that our model is considerably accurate with an error below 3% for the CMP case and below 5% for the CMP+SMT case.

Finally, we show that by placing threads in a workload and package-aware manner, we can achieve significant energy improvements, without incurring significant performance degradation, with a 3.7X reduction in energy-delay product. We expect that characterizations, like the one done in this paper, will help in the design of power and temperature-aware schedulers for fully exploiting the low-power and thermal capabilities of the underlying CMP/SMT processors. Our two case studies show examples of such applications.

Although the results obtained in this characterization are not directly portable to other systems or architectures, we believe that the methodology presented in this work, based on a combination of microbenchmarks stressing different bottlenecks in the system, and industry-standard benchmarks such as SPEC CPU2006, can indeed be used for characterizing other systems or architectures. It is left as future work envisioning ways to automatically characterize a system, provided a description of the architecture, following this methodology.

8. ACKNOWLEDGMENTS

This work was supported by a Collaboration Agreement between IBM and BSC with funds from IBM Research and IBM Deep Computing organizations. It has also been supported by the Ministry of Science and Technology of Spain under contract TIN-2007-60625 and grants AP-2005-3776 and AP-2005-3318, and by the HiPEAC Network of Excellence (IST-004408).

9. REFERENCES

[1] Power ISA Version 2.06, 2009. http://www.power.org/resources/downloads/PowerISA_V2.06_PUBLIC.pdf.

[2] B. Behle et al. IBM EnergyScale for POWER6 Processor-Based Systems. *IBM White Paper*, 2009.

[3] W. Bircher et al. Complete system power estimation: A trickle-down approach based on performance events. *ISPASS*, 2007.

[4] C. Boneti et al. Software-Controlled Priority Characterization of POWER5 Processor. *ISCA*, 2008.

[5] D. M. Brooks et al. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *MICRO*, 2000.

[6] J. Casazza. *Intel Core i7-800 Processor Series and the Intel core i5-700 Processor Series Based on Intel Microarchitecture (Nehalem)*, 2009.

[7] J. Choi et al. Thermal-aware task scheduling at the system software level. *ISLPED*, 2007.

[8] EPA. EPA Report to Congress on Server and Data Center Energy Efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[9] X. Fan et al. Power provisioning for a warehouse-sized computer. *ISCA*, 2007.

[10] M. S. Floyd et al. System power management support in the IBM POWER6 microprocessor. *IBM J. Res. Dev.*, 51(6), 2007.

[11] R. Gioiosa et al. Analysis of System Overhead on Parallel Computers. *ISSPIT*, 2004.

[12] R. Gioiosa et al. Transparent Incremental Checkpoint at Kernel level: A Foundation for Fault Tolerance for Parallel Computers. *SC*, 2005.

[13] H. Hanson et al. Power, Performance, and Thermal Management for High-Performance Systems. *HPPAC*, 2007.

[14] H. Hanson et al. Thermal response to DVFS: Analysis with an Intel Pentium M. *ISLPED*, 2007.

[15] F.E. Harrell, Jr. *Regression Modeling Strategies.* Springer-Verlag New York, Inc., 2006.

[16] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 2006.

[17] E. Kursun et al. Variation-aware thermal characterization and management of multi-core architectures. *ICCD*, 2008.

[18] H. Q. Le et al. IBM POWER6 microarchitecture. *IBM J. Res. Dev.*, 51(6), 2007.

[19] Y. Li et al. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. *HPCA*, 2005.

[20] D. Meisner et al. PowerNap: Eliminating server idle power. *ASPLOS*, 2009.

[21] A. Naveh et al. Power and thermal management in the Intel Core Duo processor. *Intel Technology Journal*, 10(2), 2006.

[22] V. Pallipadi. Cpuidle - Do nothing, efficiently... *Linux Symposium*, June 2007.

[23] M. D. Powell et al. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. *HPCA*, 2009.

[24] K. K. Pusukuri et al. A Methodology for Developing Simple and Robust Power Models Using Performance Monitoring Events. *WIOSCA*, 2009.

[25] R. Sasanka et al. The energy efficiency of CMP vs. SMT for multimedia workloads. *ICS*, 2004.

[26] S. Siddha et al. Getting maximum mileage out of tickless. *Linux Symposium*, June 2007.

[27] B. Sinharoy et al. POWER5 system microarchitecture. *IBM J. Res. Dev.*, 49(4/5), 2005.

[28] V. Srinivasan et al. Energy-Aware Task and Interrupt Management in Linux. *Linux Symposium*, 2, August 2008.