HaPPy: Hyperthread-aware Power Profiling Dynamically

Yan Zhai, *University of Wisconsin;* Xiao Zhang and Stephane Eranian, *Google Inc.;* Lingjia Tang and Jason Mars, *University of Michigan*

https://www.usenix.org/conference/atc14/technical-sessions/presentation/zhai

This paper is included in the Proceedings of USENIX ATC '14: 2014 USENIX Annual Technical Conference.

June 19–20, 2014 • Philadelphia, PA

978-1-931971-10-2

HaPPy: Hyperthread-aware Power Profiling Dynamically

Yan Zhai
University of Wisconsin
yanzhai@cs.wisc.edu

Xiao Zhang, Stephane Eranian
Google Inc.
{xiaozhang,eranian}@google.com

Lingjia Tang, Jason Mars
University of Michigan
{lingjia,profmars}@eesc.umich.edu

Abstract

Quantifying the power consumption of individual applications co-running on a single server is a critical component for software-based power capping, scheduling, and provisioning techniques in modern datacenters. However, with the proliferation of hyperthreading in the last few generations of server-grade processor designs, the challenge of accurately and dynamically performing this power attribution to individual threads has been significantly exacerbated. Due to the sharing of core-level resources such as functional units, prior techniques are not suitable to attribute the power consumption between hyperthreads sharing a physical core.

In this paper, we present a runtime mechanism that quantifies and attributes power consumption to individual jobs at fine granularity. Specifically, we introduce a hyperthread-aware power model that differentiates between the states when both hardware threads of a core are in use, and when only one thread is in use. By capturing these two different states, we are able to accurately attribute power to each logical CPU in modern servers. We conducted experiments with several Google production workloads on an Intel Sandy Bridge server. Compared to prior hyperthread-oblivious model, HaPPy is substantially more accurate, reducing the prediction error from 20.5% to 7.5% on average and from 31.5% to 9.4% in the worst case.

1 Introduction

As more of the world's computation moves into large-scale datacenter infrastructures, power management and provisioning becomes increasingly important. In fact, prior work [4] shows that the cost of powering the servers housed in these infrastructures comprises about 30% of the total cost of ownership (TCO) of modern datacenter infrastructures. As we are reaching the limits of current power delivery systems, many datacenter infrastructures house more machines than can be powered by the supply infrastructure [17]. In tandem with these trends, datacenter designers and operators have been investigating techniques to manage the available power resources via software techniques such as power-capping [13, 11], scheduling [12], and energy accounting/pricing [28], among others. Software power capping and provisioning techniques ensure that servers do not use more than a specified power threshold by suspending a subset of jobs. Scheduling can also be used to limit processor utilization to reach energy consumption goals. Beyond power budgeting, pricing the power consumed by jobs in datacenters is also important in multi-tenant environments.

One capability that proves critical in enabling software to monitor and manage power resources in large-scale datacenter infrastructures is the attribution of power consumption to the individual applications co-running on a single server. This ability allows software to control power consumption at the level of individual applications on a single machine, as well as across entire clusters. However, accurate attribution on real-world commodity hardware has proven challenging for modern server designs, particularly due to the fact that *simultaneous multithreading*, (or *hyperthreading* [14]) is now commonplace in current server designs.

Processors that are *hyperthreaded* allow two or more *hardware thread contexts* to share a single physical core. Although the OS views each hardware thread context as a logical CPU, a number of core-level resources are shared across contexts such as functional units, alias register, and cache resources, among others. Modern processors do not provide specific power monitors for each hardware thread context and thus attributing the power consumption of individual processes and threads across logical CPUs has proven particularly challenging.

In this work, we present **HaPPy**, a **H**yperthread-**a**ware **P**ower **P**rofiling D**y**namically. HaPPy is able to dynamically and near instantaneously attribute the power consumed (in watts) to individual processes or threads. To the best of our knowledge, this is the first such hyperthread-aware power estimation approach. Central to HaPPy is an estimation model that uses Intel *Running Average Power Limit* (RAPL) power/performance monitoring interface [14] that is widely available on current commodity servers (Sandy Bridge/Ivy Bridge/etc). Although RAPL provides no power monitoring information of individual cores nor hardware thread contexts, HaPPy uses a novel execution isolation technique implemented on top of existing performance counter tool to predict the power consumed by individual threads.

We evaluate HaPPy on six data-intensive Google production workloads using real commodity server configurations found in datacenters. Compared to prior work, HaPPy is substantially more accurate and reduces the prediction error from 20.5% to 7.5% on average and from 31.5% to 9.4% in worse cases.

2 Background

The primary goal of this work is provide a technique to enable the attribution of power consumption to individual threads. In this section, we first describe the need for power estimation and the most related works. Then, we describe the underlying hardware monitoring interface that underpins our HaPPy approach.

2.1 Need for Power Estimation

Power estimation for individual jobs is critical for power management systems in datacenters [13, 11, 17, 12, 28]. To lower the total cost of ownership, particularly the cost of power infrastructures, modern datacenters are often designed to house more servers than can be powered by the underlying power supply infrastructure. At peak time, the power demand of the datacenter may surpass the supply of the power infrastructure, in which case power capping techniques are applied to lower the demand to under the provisioning threshold. There are various types of power capping techniques, including suspending or limiting the processor utilization of certain jobs. These approaches require accurate power estimation for individual jobs. Power estimation allows us to accurately identify the minimum amount of power-hungry jobs the system needs to suspend given the target power demand threshold. A more conservative power capping system without the power estimation might need to suspend all low-priority jobs, which is much less cost-effective. In addition to power capping, power estimation is also critical for facilitating accurate pricing and accounting in multi-tenant cloud infrastructures. Accurate power usage estimation for individual applications on a shared server allows us to design power-based billing and pricing for cloud infrastructure users.

2.2 State of Power Estimation in Datacenters

Power constraints are well recognized as one of the primary limiting factors for datacenter design, and there is a significant body of work [19, 13, 11, 26] targeting advanced power estimation and management in datacenters. Two works emerge as most related. The work by Fan *et al.* presents power provisioning designs for datacenters [11]. The model presented in this paper focuses on coarse-granularity prediction, which is suitable for its goal. However, it is hyperthread-oblivious and incurs high inaccuracy when directly applied to attributing total server power to individual co-running tasks running on hyperthreaded processors. The work by Shen *et al.* models CPU power at a fine-grained server requests level on hyperthreading disabled servers [26]. Our work is complementary to both of these important contributions as our hyperthread aware CPU power modeling is applicable to tasks concurrently running on a hyperthreaded machines.

2.3 The RAPL Interface

Recently Intel released the RAPL *model specific registers* (MSRs). These performance counters enable software to read processor energy consumption at run time on newer Intel processors such as Sandy Bridge and Ivy Bridge. RAPL MSRs separate processor energy consumption into three parts: `pp0`, `package`, and `dram`. `pp0` counts total energy consumed by all cores of a processor (note that RAPL does not provide per-core measurements on Sandy Bridge or Ivy Bridge); `package` includes both cores and uncore (e.g. last-level-cache) energy consumption; `dram` here means on-chip dram channels, not the commonly referred off-chip memory DIMM. Total processor energy consumption can be calculated by aggregating `package` and `dram` readings. The reported energy during a given time window can then be converted to the average power.

The Linux kernel provides a powerful open-source tool, called *perf* [1], to configure and monitor hardware performance counters. We have extended this interface to enable access to Intel's RAPL counters. The extension is implemented as a separate socket-level performance monitoring unit (PMU). To monitor energy consumption of a multi-socket system, it is only necessary to monitor the RAPL events from one CPU on each socket. Our *perf* patch has been open-sourced [2] and will appear in upstream kernels (Linux 3.14 and newer).

3 Power Modeling

In this section, we first present a hyperthread-oblivious model commonly used in prior work. We then discuss why it is insufficient and inaccurate on modern servers with hyperthreading. Finally, we present our hyperthread-aware model that can accurately attribute power consumption across co-running tasks.

3.1 Hyperthread-oblivious Model

We first present a hyperthread-oblivious (HT-oblivious) model, which is used in prior work for event-driven power accounting [6, 26]. The model is based on the hypothesis that the power consumption of a task is proportional to the amount of computation CPUs perform for that task, and one can estimate the amount of CPU computation using hardware events, such as CPU cycles and instructions.

Figure 1 presents the correlation between applications' power consumption and their aggregated non-halted CPU cycle[1] counts (total cycle counts for all

[1]Non-halted means CPU is not executing the "halt" instruction in x86 instruction set.

workload	description	characteristics
bigtable (BT) [8]	Distributed storage system for managing structured data	Memory-intensive
web-index (IDX) [5]	Web indexing	CPU-intensive
youtube-encoding(YTB)	Youtube video encoding	Floating point-intensive
warp-correction (IMG)	Corrects warped images in scanned material	CPU-intensive
mapreduce (MR) [10]	Map-reduce benchmark written in Sawzall [24] script	Memory-intensive
rpc-bench (RPC)	Google rpc call benchmark	CPU-intensive

Table 1: *Brief description of Google's internal applications used in this study. All applications are memory resident and fully utilize server memory and CPUs.*

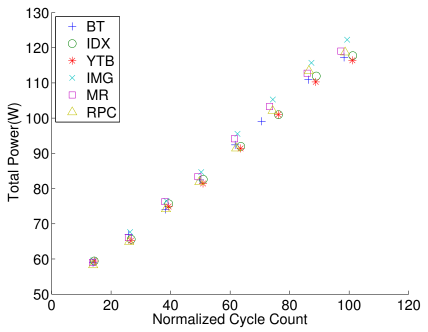

Figure 1: *Correlation of power with non-halted CPU cycle.*

threads of an application). In these experiments, we use several diverse real Google workloads (see description in table 1) and an Intel Sandy Bridge server with the same configuration found in production. We run N instances of each application on N physical cores on a server. We do not use hyperthreading in this experiment, so only 1 hyperthread of a physical core is used. During the experiment, we collect the total processor power consumption and aggregated non-halted CPU cycles using *perf* [1]. As demonstrated in Figure 1, the aggregated CPU cycles are strongly correlated with the power consumption (linear correlation coefficient 0.99).

Besides non-halted CPU cycles, we also examined other metrics including instruction count, last-level-cache reference and miss, through a wide range of microbenchmarks, including a busy-loop benchmark (high instruction issue rate), a pointer chasing benchmark (high cache miss rate), a CPU and memory intensive benchmark (to mimic power virus behavior), and a set of bubble-up benchmarks that incur adjustable amounts of pressure on the memory systems [27]. Our conclusion is that non-halted cycle is the best to correlate power (linear correlation coefficient above 0.95). This finding is consistent with prior work [11] which suggests a strong correlation between the machine-level power consumption and CPU utilization.

Based on the correlation between the power consumption and the cycle count when hyperthreading is not in use, the power consumption across all currently running

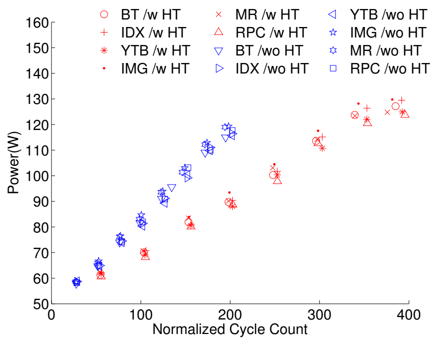

Figure 2: *Correlation between power and cycle when hyperthreads are enabled.*

tasks can be attributed simply based on each task's cycle count. This HT-oblivious model for attributing power is as follows:

$$Power(Task_i) = Total\_Active\_Power \times \frac{Cycle(Task_i)}{\sum_{j=1}^{m} Cycle(Task_j)} \quad (1)$$

3.2 Why is accounting hyperthreading important?

The HT-oblivious model assumes that the power consumption of a task is strongly correlated with the aggregated CPU cycles of the task. However, as we will demonstrate in this section, this is no longer the case when hyperthreads are used. Figure 2 presents the correlation between the measured power consumption and the aggregated cycle counts when tasks use hyperthreads (2 tasks pinned to the two hyperthreads of each physical core) versus when tasks do not use hyperthreads (only 1 task pinned to each physical core). As presented in Figure 2, *the aggregated CPU cycles of a task are not strongly correlated with its power consumption when hyperthreading may be in use*. For example, as shown in the figure, when the normalized cycle count is around 200, the power consumption can be wildly different ranging between 85w and 120w, almost 40% difference.

To illustrate the reason behind this 40% discrepancy, imagine when both hyperthreads (HT) of a physical core are in use, the aggregated CPU cycles may double comparing to the scenario when only 1 HT per core is in use (each HT is a logical CPU). However, the power consumption is not doubled. Actually, there is only a slight power increase over the scenario when 1 HT/core is used. This is because that two hyperthreads of a physical core share many low-level hardware resources (such as functional units), thus only incur slight power increase when both are active. *A basic hyperthreading-oblivious model does not distinguish these two scenarios (with and without hyperthreading), and therefore is inaccurate*.

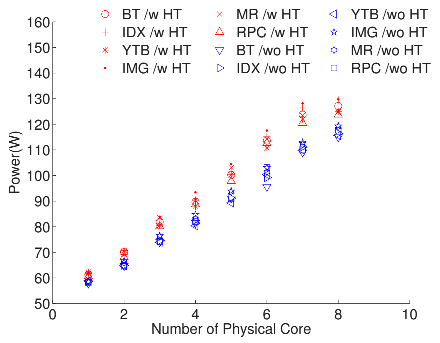

Figure 3: *Power comparison when using only one or both hyperthreads of a physical core.*

Figure 3 further demonstrates the power consumption difference between using no hyperthreading (1 task pinned to only one logic CPU of a physical core) and using hyperthreading (2 tasks pinned to the two hyperthreads, logical CPUs, of a physical core). The x-axis shows the number of physical cores used in each experiment. For n physical cores, we execute n replica of a task for w/o hyperthreading scenario and $2n$ replica of the task for w/ hyperthreading scenario. The y-axis shows the measured total CPU power. Again, as presented in Figure 3, *accounting hyperthreading is critical for the accuracy of a power model*. For example, when attributing power for 8 (single-threaded) tasks, it is important to differentiate whether 8 tasks are running on eight cores (~115w total and 14 w/task) or on four physical cores with 2 hyperthreads each core (~90w total and 11 w/task, 30% less than 14 w/task) or the mix of both scenarios. The Evaluation section (Section 4) will further demonstrate the inaccuracy when one fails to acknowledge the hyperthread configurations. Also note that, as shown in Figure 3, the ratio between power consumption when both hyperthreads are in use and that when only one hyperthread is in use is about 1.1. We refer this ratio as R_{ht} for the rest of this paper.

3.3 Hyperthread-aware Model

In this section, we present a novel hyperthread-aware (HT-aware) model that addresses the challenge of accounting per task power consumption when tasks may use hyperthreads. We first break down the total CPU power consumption into *static power* and *active power* , and then focus on modeling the active power. To attribute the active power across all co-running tasks, our model first attributes the power consumption for each physical core using a novel technique to account for how tasks are taking advantage of hyperthreading of the core. We then attribute the power consumption of each task based on the cycles each task executes on each core.

3.3.1 Attributing active power among physical cores

Static Power - Processors often consume a small amount of power just to be active, even when there is not much computation activity. For example, linearly extrapolating the data points in Figure 3 shows that when 0 core is in use, the power consumption is around 50 watt. This means that there is around 50 watt of power consumption even when there is minimum core activity. We refer to this power consumption as the *static power* and use linear extrapolation to estimate it. The static power on our test machine is estimated to be 52.5 watts.

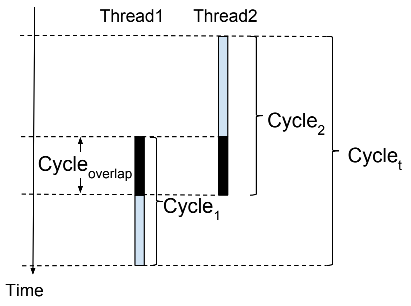

Figure 4: *Illustration of how we capture the detailed utilization of a physical core using three counters. Thread 1 and 2 are sibling hyperthreads of a physical core, and their non-halted CPU cycles are respectively $Cycle_1$ and $Cycle_2$. $Cycle_t$ represents non-halted cycles when* at least *one of the two hyperthreads of a physical core is active.*

Attributing the active power to each physical core - We calculate the active power using the total CPU power consumption minus the static power estimated as discussed above. To attribute the active power across physical cores, our model takes advantage of three hardware counters. As illustrated in Figure 4, $Cycle_1$and $Cycle_2$ are non-halt CPU cycles respectively for thread 1 and 2. $Cycle_t$ is CPU cycles when *at least* one of the two hyperthreads of a physical core is running [2].

From these three counters, we can infer:

$$Cycle_{overlap} = Cycle_1 + Cycle_2 - Cycle_t \tag{2}$$

$$Cycle_{nonoverlap} = Cycle_t - Cycle_{overlap} \tag{3}$$

$Cycle_{overlap}$ is the portion of time when both hyperthreads of a physical core are running, while

[2]In *perf* tool, $Cycle_1$ and $Cycle_2$ are obtained by cpu event `0x3c` with `umask=0x00`, while $Cycle_t$ can be obtained by same event with `umask=0x00` and `any=1`.

$Cycle_{nonoverlap}$ is that when only one thread is running. Using $Cycle_{overlap}$ and $Cycle_{nonoverlap}$, we define the weighted cycle of a physical core as follows:

$$Cycle_{weighted}(Core) = R_{ht} \times Cycle_{overlap} + Cycle_{nonoverlap} \quad (4)$$

The **intuition** here is that *when two hyperthreads are both executing on a physical core, the power consumption of the physical core is R_{ht} times its power consumption when only one hyperthread is executing on the core.* R_{ht} is computed from the data in Figure 3 to be 1.1. So by calculating the time period when two hyperthreads are executing ($Cycle_{overlap}$) and that when only one is executing ($Cycle_{nonoverlap}$), we can calculate a weighted core utilization ($Cycle_{weighted}$) and use it to estimate each core's power consumption.

Now we can attribute the total active power of a processor among individual cores proportional to each core's hyperthread-weighted $Cycle_{weighted}$:

$$Active\_Power(Core_i) = Total\_Active\_Power \times \frac{Cycle_{weighted}(Core_i)}{\sum_{j=1}^{n} Cycle_{weighted}(Core_j)} \quad (5)$$

3.3.2 Attributing active core power to hyperthreads

Following the same principle of Equation 4, we can calculate the weighted cycles for individual hyperthreads on each core:

$$Cycle_{weighted}(HT_i) = R_{ht} \times \frac{Cycle_{overlap}}{2} + (Cycle_i - Cycle_{overlap}) \quad (6)$$

HT_i is one of two hyperthreads of a physical core. Recall that $Cycle_{overlap}$ indicates the time when both threads are running (Equation 2), in which case we attribute the power to each individual hyperthread evenly. $Cycle_i - Cycle_{overlap}$ represents the time thread i runs alone, in which case the thread is attributed the total power consumed by the core. Using $Cycle_{weighted}(HT_i)$ calculated by Equation 6 and $Active\_Power(Core_i)$ calculated by Equation 5, we can proportionally attribute the total active power of a core to each hyperthread on that core using the following equation:

$$Active\_Power(HT_i) = Active\_Power(Core) \times \frac{Cycle_{weighted}(HT_i)}{Cycle_{weighted}(Core)} \quad (7)$$

3.4 Mapping from hardware to applications

Reserving logical CPUs is a common practice in datacenters to achieve better performance isolation [22]. With this technique, the process threads associated with a job run on dedicated CPUs using containers and the `set_affinity` API [3]. Our approach attributes active power to such jobs by calculating the power consumption of the hyperthread contexts associated with each hosted process as shown in Equation 7. Reserving CPUs is often used for minimizing performance interference to latency-critical jobs as well as in Infrastructure as a Service (IaaS) type of multi-tenant cloud services. We use this execution approach as the basis of our evaluation.

When CPUs are time-shared by multiple jobs, our models can be used to capture the power consumption change at each context switch using Equation 7. The cost of reading the performance counters is typically hundreds of nanoseconds while the remaining cost of a context switch is on the order of microseconds [20]. It is important to note that the OS scheduler only needs to save threads' co-run performance counter information at every context switch, more complex calculations can be deferred to a coarser scale (seconds) or on demand.

4 Evaluation

In this section, we evaluate the accuracy of our model using production Google applications listed in Table 1. We also duplicate our experiment using SPEC benchmarks for repeatability. Our Google configuration is a 2.6GHz Intel Sandy Bridge machine, equipped with 2 processor sockets. Each socket has 8 cores, each with two hyperthreads. Only one socket is used in our experiments. The testbed runs a customized Linux kernel with necessary RAPL support. We collect energy readings via *perf* tool every 10 seconds and report the average power during a 300-second application execution.

4.1 Methodology

In our experimental setup, we co-run two jobs on a processor socket and pin them to disjoint sets of cores. The first job spawns $2N$ processes on N cores, while the second job spawns N processes on another N cores. We scale up N from 1 to 4 in our experiments. With this configuration, the first job makes full usage of all hyperthreads ($2N$ hyperthreads on N cores), while the second job uses half of the available hyperthreads (N hyperthreads on N cores). We first calculate total active power consumed by both jobs as $Power_{total}$ (measured total processor power minus static power). We then estimate each job's power consumption, $Power_1$ and $Power_2$, using both HT-oblivious model (Equation 1) and our HT-aware model (Equations 2 - 7).

To evaluate the accuracy for estimating per job power consumption, we remove one job from the server and measure the power consumption of the remaining job as $Power'_{total}$. The delta between $Power_{total}$ and $Power'_{total}$ is the actual active power of the removed job. We refer to this measured per task power as "Oracle", and use it as an evaluation baseline in the following section.

Workload	HT-oblivious		HT-aware	
	Avg Error	Max Error	Avg Error	Max Error
{bigtable (BT), warp-correction (IMG)}	5.8w(23.1%)	11.9w(28.5%)	1.8w(7.0%)	3.7w(8.6%)
{web-index (IDX), mapreduce (MR)}	5.1w(19.4%)	13.4w(31.0%)	2.4w(9.3%)	4.4w(10.2%)
{youtube-encoding(YTB), rpc-bench (RPC)}	4.8w(19.1%)	13.9w(34.5%)	1.5w(6.2%)	3.9w(9.6%)
Average	5.2w(20.5%)	13.1w(31.3%)	1.9w(7.5%)	4.0w(9.4%)

Table 2: *Average and maximal errors for Google benchmarks, both in absolute watt and relative percentage, of two models when using Oracle as baseline. Error in percent is calculated as* $\frac{|Oracle_{avg}-Model|}{Oracle_{avg}}$ *and* $\frac{|Oracle_{max}-Model|}{Oracle_{max}}$.

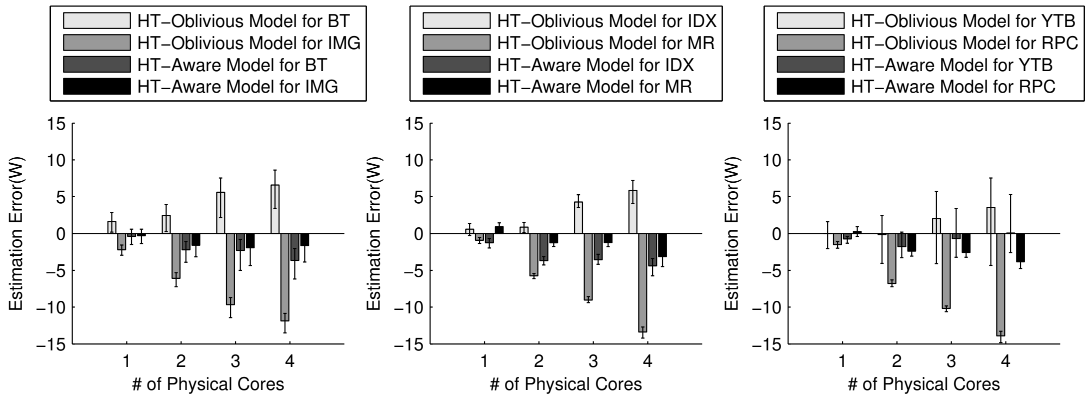

Figure 5: *Results for three sets of workload {bigtable (BT), warp-correction (IMG)}, {web-index (IDX), mapreduce (MR)}, and {youtube-encoding(YTB), rpc-bench (RPC)}*

4.2 Results

We conduct the evaluation using three pairs of co-running applications, chosen arbitrarily: {bigtable (BT), warp-correction (IMG)}, {web-index (IDX), mapreduce (MR)}, {youtube-encoding(YTB), rpc-bench (RPC)} (applications are described in Table 1). As discussed in Section 4.1, for each pair, we conduct four experiments varying the number of physical cores N from 1 to 4. The first job in a pair spawns $2N$ processes running on N cores, while the second only spawns N processes on another N cores. Each experiment runs three times. Both the average values and standard deviations are reported.

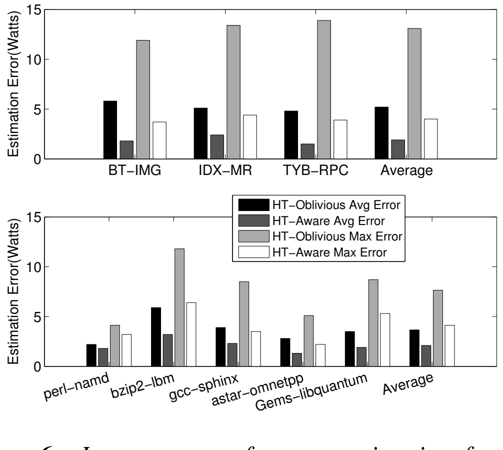

Figure 6: *Improvement of power estimation for both Google and SPECPU benchmarks.*

Figure 5 presents our experimental results. It shows the power attributing results for each co-run pair. As shown in the figure, the prediction accuracy of our HT-aware model outperforms the HT-oblivious model. Figure 5 shows that the HT-oblivious model tends to overestimate jobs with both hyperthreads running (*i.e.*, bigtable (BT), web-index (IDX), and youtube-encoding(YTB)) and underestimate jobs with only one hyperthread running (*i.e.*, warp-correction (IMG), mapreduce (MR), and rpc-bench (RPC)). These prediction errors are expected since the HT-oblivious model solely depends on CPU cycles. In contrast, our HT-aware model takes architecture details into consideration and is more accurate in all cases. As summarized in Table 2, it on average reduces error from 20.5% to 7.5% when compared to HT-oblivious. The maximal error of HT-aware prediction is significantly less than the HT-oblivious model, reducing from ∼13 watts error (or 31.3%) for the HT-oblivious model to ∼4 watts (or 9.4%) for our HT-aware model.

4.3 SPEC results

To demonstrate the repeatability of our experiments beyond Google applications, we duplicated the experiments using SPEC CPU2006 benchmarks on another Sandy Bridge machine. On this machine, each CPU socket has six 1.9GHz physical cores. In these experiments, we used 10 SPEC benchmarks and randomly group them in 5 pairs. Figure 6 presents the power prediction error achieved by HaPPy versus the hyperthread-oblivious model. As shown in the figure, there is a significant improvement in prediction accuracy for both Google and SPEC workloads.

5 Related Work

Several techniques have been proposed to predict the server power [6, 9, 18, 15, 21, 23]. For example, Bellosa proposed an event driven approach for power modeling. Choi *et al.* discussed power prediction and capping in consolidated environment [9]. Lee *et al.* designed a regression model for power prediction in hardware simulators [18], whereas our model is applicable for real machines. Isci *et al.* presented a framework to collect and analyze power phases [15]. These work either do not explicitly address hyperthreaded servers or simply disable hyperthreading. Fine-grained power profiling tools are also proposed [25]. Shen *et al.* designed a power container to profile server request level power [26]. Kansal *et al.* and Bertran *et al.* used system events to model application level power [16, 7]. Again these models are not aware of hyperthreads. Power management in data center has attracted much research attention recently [19, 13, 11]. These power management techniques require accurate power estimation.

6 Conclusion

In this paper, we present a simple and accurate hyperthread-aware power model to attribute power consumption of a server to individual co-running applications. By leveraging on-chip energy counters and *perf* tool, we prototype our model as a lightweight runtime task power profiler. Our evaluation using Google commercial benchmarks shows that the prediction accuracy of our model is significantly better than the state-of-the-art hyperthread-oblivious model.

Acknowledgement

We would like to thank Keith Smith for his valuable feedback. This work was partially supported by NSF Awards CCF-SHF-1302682, CNS-CSR-1321047 and a Google Research Award.

References

[1] `https://perf.wiki.kernel.org/`.

[2] `https://lkml.org/lkml/2013/10/7/359`.

[3] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proc. of the 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 1999.

[4] L. A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2nd edition, 2013.

[5] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, Mar. 2003.

[6] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proc. of the SIGOPS European Workshop*, Kolding, Denmark, Sept 2000.

[7] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proc. of the 24th ACM International Conference on Supercomputing (SC)*, pages 147–158, 2010.

[8] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. In *Proc. of the 7th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 205–218, 2006.

[9] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam. Profiling, prediction, and capping of power consumption in consolidated environments. In *Proc. of Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS)*, pages 1–10, 2008.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. of the 6th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2004.

[11] X. Fan, W. D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proc. of the 34th annual International Symposium on Computer Architecture (ISCA)*, pages 13–23, 2007.

[12] I. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. In *Proc. of 18th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 51–64, 2013.

[13] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *Proc. of the 4th ACM European Conference on Computer systems (EuroSys)*, pages 317–330, 2009.

[14] Intel Corporation. Intel 64 and IA-32 architectures software developer's manual, volume 3: System programming guide, 2013.

[15] C. Isci and M. Martonosi. Phase characterization for power: evaluating control-flow-based and event-counter-based techniques. In *Proc. of 12th Int'l Symp. on High Performance Computer Architecture (HPCA)*, pages 121–132, 2006.

[16] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):26–31, 2008.

[17] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing distributed UPS energy for effective power capping in data centers. In *Proc. of the 39th annual International Symposium on Computer Architecture (ISCA)*, pages 488–499. IEEE, 2012.

[18] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 185–194. ACM, 2006.

[19] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proc. of the 4th International Conference on Autonomic Computing (ICAC)*, 2007.

[20] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *Proc. of the Workshop on Experimental Computer Science (ExpCS)*, 2007.

[21] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. In *Proc. of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 160–171, 2003.

[22] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proc. of the 44th annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, New York, NY, USA, 2011. ACM.

[23] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. Evaluating the effectiveness of model-based power characterization. In *Proc. of the USENIX Annual Technical Conference (USENIX ATC)*, 2011.

[24] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming, Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure*, 13(4):277–298, Oct. 2005.

[25] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the Cinder operating system. In *Proc. of the 6th ACM European Conference on Computer systems (EuroSys)*, pages 139–152, 2011.

[26] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and C. Zhuan. Power containers: An OS facility for fine-grained power and energy management on multicore servers. In *Proc. of 18th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Houston, Texas, Mar. 2013.

[27] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-Flux: Precise online QoS management for increased utilization in warehouse scale computers. In *Proc. of the 40th annual International Symposium on Computer Architecture (ISCA)*, 2013.

[28] Q. Zheng and B. Veeravalli. Utilization-based pricing for power management and profit optimization in data centers. *Journal of Parallel and Distributed Computing*, 72(1):27–34, 2012.