# CPT: An Energy-Efficiency Model for Multi-core Computer Systems

Weisong Shi, Shinan Wang and Bing Luo
Department of Computer Science
Wayne State University
{shinan, weisong, bing.luo}@wayne.edu

***Abstract*—Resolving the excessive energy consumption of modern computer systems has become a substantial challenge. Therefore, various techniques have been proposed to reduce power dissipation and improve energy efficiency of computer systems. These techniques affect the energy efficiency across different layers in a system. In order to better understand and analyze those techniques, it is necessary to obtain a general metric that represents the energy efficiency of a computer system, for a specific configuration, given a certain amount of workload.**

**In this paper, we take the initial step and define a general energy-efficiency model, the CPT model, for multi-core computer systems. CPT is a unified model that helps to decide the best configuration of a system in terms of energy efficiency to execute a given workload. In addition, we expect the model can be utilized to analyze possible knobs that are used to improve energy efficiency. Three case studies are employed to illustrate the usage of the proposed CPT model.**

## I. Introduction

The conventional computing area is dominated by the pursuit of performance. The community has not realized the importance of energy efficiency until recently [1]. As a result, energy-efficient techniques have been used across different layers in almost every single system, ranging from a single chip to a large data center. These techniques include low-power circuit designs, tunable device states (Dynamic Voltage and Frequency Scaling), dynamic power management from operating systems, and energy-efficient software. Although current computer systems already achieve much higher efficiency ratings compared to that of previous generations, there is still potential headroom for improvement.

With the development in energy-efficient computing, one of the fundamental questions is how to define a model to quantify energy efficiency. An appropriate energy-efficiency metric not only can be used to evaluate and analyze existing techniques, but also helps to explore new techniques in this field. However, defining a energy-efficiency model is challenging. For one thing, the model should be sufficiently general in describing various techniques. Commonly used knobs such as multi-threading and DVFS have to be meaningful according to the model. Optimizations from different layers have to be expressed at some level from the model as well. For instance, the model is better to convey the idea of both clocking gating [2] and workload consolidation [3].

This work is in part supported by NSF grant CNS-1205338 and the Introduction of Innovative R&D team program of Guangdong Province (NO.201001D0104726115).

Moreover, energy efficiency has to be associated with workload characteristics. For example, requests per second or transactions per day is the metric that typically is used to measure the throughput of web service applications, while million instructions per second (MIPS) is of the most interested performance indicators in scientific computing field.

In this paper, we propose a general energy-efficiency model, CPT, which enables efficiency analysis of a system, given a running workload. The rest of the paper is organized as follows: we start the paper by presenting the CPT model and analyze each component in the model in Section II, followed by case studies in Section III. Related work is discussed in Section IV. Finally, we summarize the paper in Section V.

## II. The CPT Model

In the CPT model, given a fixed amount of workload, we ask the question how much energy is consumed in order to complete the task. Specifically, the model is represented as follows

$$\begin{aligned} \mathscr{E} = Workload/Energy &= \frac{W}{(P_{AI} + C \times P_t)T} \\ &= \frac{W}{P_{AI} \times T + C \times P_t \times T} \end{aligned} \tag{1}$$

where $\mathscr{E}$ stands for energy efficiency. $W$ represents the total workload size that is assigned to a system. $P_{AI}$ and $P_t$ denote active idle power of the system and average power dissipation of each thread, respectively. Specifically, $P_{AI}$ is the power dissipation while a system is idle in $C0$ state specified in ACPI standard [4]. $C$ indicates the concurrency level of the workload. Intuitively, the more concurrency threads that are used, the quicker a job can be completed. System power dissipation, however, rises with more system resources being used. The last factor, $T$, is the total time taken to complete the workload. The name of CPT was conceived using the three most important parameters, concurrency, power and executive Time.

In order to improve the overall $\mathscr{E}$, each part in Equation 1 should be considered. In reality, changing each item usually subsequently alters other factors in Equation 1. For example, improving performance reduces $T$; however, in order to improve performance, usually, active power increases. To be clear, we argue that it is more important to compare the energy efficiency of the same workload using different designs and/or implementations.

### A. Workload (W)

Given the other factors fixed, in order to improve $\mathscr{E}$, intuitively, we can assign more workload to a system as much as possible. These scenarios can be found in data centers, where facility power dissipation is limited. At this level, the concurrency can be roughly estimated as how many nodes have been deployed in a data center. Hence, it is better to operate a facility to its upper capacity limit so that energy efficiency can be maximally guaranteed.

Fan *et al.* propose several ideas to improve $\mathscr{E}$ in [5]. Through in-depth analysis, the authors discuss possible capacity that can be safely incorporated into different layers, which include racks, PDUs, and clusters. Although in this process, there are more nodes added into the system, $P_t$ can be reduced via DVFS and idle state so that the denominator in Equation 1 remains within the total power limit of the facility. The basic idea of techniques in this category is to accomplish more jobs while keeping the product of $C$ and $P_t$ unchanged.

### B. Concurrency (C)

As multi-core platforms become common on servers and even smart phones, implementing concurrency applications generally will help to improve system performance. By assigning each piece of a job to different cores or processors, system resources can be efficiently utilized. Most likely, if a job can be finished earlier, its $\mathscr{E}$ can be improved as well because $T$ is reduced. However, it is not always the case. The well-known concurrency hazards are a collection of problems that could possibly occur if concurrency is not implemented properly. The hazards include false sharing, memory contention, incorrect granularity, and so on. One the one hand, the execution time, $T$, could be increasing. On the other hand, the total $P_t$ might rise which in turn sabotages $\mathscr{E}$. False sharing is a typical problem that can be found in multi-threading applications with shared memory. In the case of false sharing, a certain amount of cache lines are being swapped in and out from a cache frequently, which invokes additional power dissipation and extends the execution time. In this case, $P_t$ and $T$ are both affected.

Speedup models are supposed to estimate the benefits in terms of execution time by using more threads to work on a job [6]. Normally, allocating more cores to a job has dramatic benefit on execution time if it is used properly. For example, embarrassingly parallel applications benefit the most from multi-core architecture theoretically because there is no dependency between paralleled tasks. A typical example is that a GPU has a much larger number of cores (from 500 - 900) compared to that of a general purpose CPU. However, most other parallel applications do not hold the assumption that there is no dependency between tasks. Communication between tasks becomes the primary concern when the number of cores increases. Consequently, the bottleneck of finishing a job shifts from computation needs to communication demands. The speedup effect diminishes while the concurrency level still ascends, eventually decreasing $\mathscr{E}$ [6].

The concurrency level, $C$, influences the overall $\mathscr{E}$ in various ways. Its effect is a complex combination of system architecture and workload characteristics. Optimal concurrency level from a performance perspective does not necessarily indicate the maximum $\mathscr{E}$. Selecting an appropriate $C$ becomes a more complicated problem in power-aware computing setting (DVFS-enabled).

### C. Active idle power ($P_{AI}$)

The active idle state is the normal operating state of a system when it is idle, or according to ACPI standard, the $C0$ idle state. No wake up is required before a system executes jobs. In active idle state, power has been utilized to maintain the operation state of a system.

Most techniques that used to improve $\mathscr{E}$ by reducing $P_{AI}$ are at circuit level. The idea is to reduce leakage power. Usually low power design devices can be used to achieve this goal. As the density of transistors on a die increases tremendously, the static power dissipation of a processor occupies a large portion of the total power. The reason is because leakage power rises as more transistors are put on a chip. Low power devices usually sacrifice some performance to achieve less power dissipation.

One well-known strategy,"race to idle" [7], states that a system should finish its job as quickly as possible and rush to an idle state. This is partially because $\mathscr{E}$ can be improved by reducing $T$. In addition, a system could enter deeper $C$ states.

One of the most beneficial result that comes from reducing active idle power is that it almost has no effects to other components in Equation 1. Therefore, it can be safely used together with other proposed techniques targeting other components. Moreover, $P_{AI}$ does not depend on a particular architecture or workload type.

### D. Power dissipation per thread ($P_t$)

Power dissipation per thread represents the dynamic power dissipation in some sense. Decided by how efficiently system resources are being used, dynamic power dissipation associates with run-time system management, system architecture, workload characteristics, and so on. Consequently, various factors affect power dissipation per thread. For instance, database applications exhibiting high memory and I/O utilization have distinct features from computation intensive applications in terms of dynamic power. Another example is low-power electronic devices execute specific types of tasks more efficiently compared to their counterparts. Measuring per thread power on a multi-core processor is challenging; therefore, power models are used instead of hardware measuring. We developed SPAN [8] to model power dissipation at per function per thread level.

Clock gating is one of the most widely used techniques [2]. By disabling part of the circuits so that they do not have to switch states, clock gating saves dynamic power. Workload characteristics mainly determine the effects of clock-gating. General purpose processors, although having more computation power, usually cannot satisfy low power features.

Application-specific integrated circuit (ASIC) is much more energy efficient, for certain types of tasks [9]. For example, most of the latest smart phone platforms have GPU units, which perform graphic jobs more efficiently. In this sense, it reduces $P_t$ required to finish certain tasks.

Another principle to reduce dynamic power is to put devices or components into lower power modes if they are not in use. DVFS allows run-time adjustment of power dissipation of a CPU. According to the equation $P = CV^2F$, reducing voltage and frequency has a cubic effect on power dissipation. However, one of the disadvantages of using DVFS is that it will extend execution time $T$. In this sense, the overall effects of DVFS on $\mathscr{E}$ is unclear. Normally, because of the existence of static power, extending workload execution time reduces $\mathscr{E}$ even though the average power decreases. The key point is to apply DVFS on applications properly. Isci *et al.* propose a run-time prediction model to identify program phases that are less CPU-bounded [10]. Afterwards, DVFS can be applied to these phases with limited performance loss. Hence, the extended $T$ value can be controlled during this process. The same idea can be applied to similar scenarios. For example, as far as I/O bounded applications are concerned, DVFS effectively improves $\mathscr{E}$. If CPU-bounded applications are the major targets, I/O devices can be safely put into deeper $D$ states.

Other than frequency, workload characteristics also contribute to the per thread power to a certain extend. Activities on different components of a system determines the total amount of power dissipation at that moment. For example, IPC values are highly related to CPU power [8]. Some applications suffers from high last level cache (LLC) misses, which leads to high memory power dissipation and low $\mathscr{E}$. Either by altering the implementation or algorithms, $P_t$ can be controlled. However, optimization techniques used at this level sometimes fall into the same category of performance optimization.

## III. Case Study

In this section, we use three case studies to illustrate the usefulness and effectiveness of CPT: 1) the effect of concurrency ($C$); 2) the impact of thread mapping; and 3) the influence of DVFS.

**Experiment setup** We conduct the experiments on a Intel Xeon E5620 server. The specifications are listed in Table I. There is a total of eight frequencies available, with a maximum of 2400MHz and minimum of 1600MHz. We use the NPB benchmark suite with OMP implementation to demonstrate the idea of CPT. In order to measure the energy consumption of the workload on the system, we connect a power measurement device, Watt's Up Pro [11], between the power outlet and the server. Watt's Up Pro is able to record power dissipation of the entire system at a frequency of 1Hz. It is connected to the system with a serial port. Watt's Up averages power measurement inside one second intervals, so that it is safe to use power readings and execution time to calculate total energy consumption.

| System component | Configuration |
|---|---|
| CPU | Intel Xeon E5620 |
| Microarchitecture | Nehalem |
| Processor core | Westmere-EP |
| L1 cache | 4 × 32KB I cache<br>4 × 32KB D cache |
| L2 cache | 4 × 256KB |
| L3 cache | 12MB |
| Frequency | 2400MHz |
| Number of sockets | 2 |
| Num of cores per chip | 4 |
| Num of threads per chip | 8 |
| Total num of threads | 16 |
| Kernel version | Linux 2.6.31 |

TABLE I
System specification.

**Case Study 1: Concurrency:** We show the effects of concurrency on other factors and $\mathscr{E}$. Firgure 1 demonstrates the effects of concurrency on the average power dissipation, workload execution time, and total energy consumption. As discussed, increasing concurrency level properly can reduce execution time. The specific speedup factor varies among different workloads. EP, IS, LU, and UA benchmarks are shown to be most affected by the concurrency level. However, most of them suffer from the diminishing of speedup. Most benchmarks in this category show less speedup if the number of threads utilized is equal or greater than four. Although FT, VIPS, and SP benchmarks benefit from concurrency, execution time is no longer monotonically related to the number of threads. The execution time increases if all 16 logical cores are employed. For the Raytrace from Parsec benchmark, it reaches optimal energy efficiency when only four cores are used. This is a typical case to consider because using more threads does not improve energy efficiency for Raytrace. It is because resource contention and serial portion of the workload become dominant factors rather than computation needs.

Average power dissipation per thread, $P_t$, decreases generally if more cores are involved in the computation. The exceptions are EP, FT, and IS benchmarks when two cores are deployed. The reason is probably because more function unit on the chip are operational if two cores are used that techniques such as clock gating is no longer in use. It is worth noticing that the total average power dissipation, which is the sum of $P_{AI}$ and $C \times P_t$, increases monotonically as more cores are used even though the value of $P_t$ drops in most cases. The extra energy consumption due the difference in power dissipation does not sabotage the overall energy efficiency because of the speedup. However, as speedup diminishes, this part of energy consumption affects the overall energy efficiency.

In this set of experiment, the optimal configuration that minimizes execution time matches the configuration generates most energy efficiency. The reason is because $P_{AI}$ occupies a great portion in the total power dissipation even if all 16 logical cores are deployed; as a result, $P_{AI} \times T$ contributes a large portion to the total energy consumption. Because of the existence

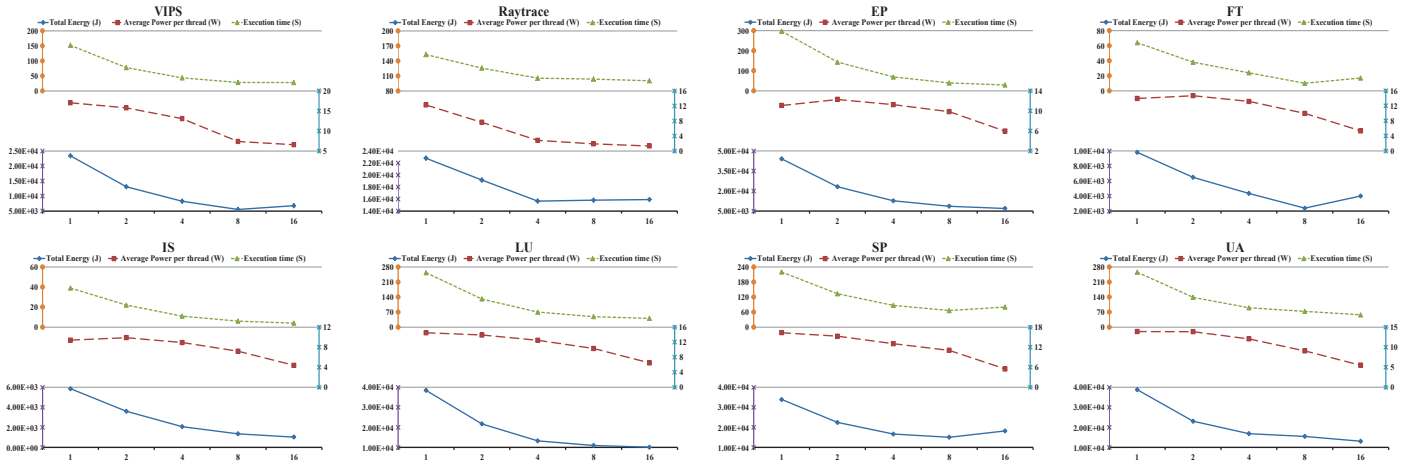


Fig. 1. Execution time, average power dissipation per thread, and total energy consumption of NPB and PARSEC benchmark; The X-axis represents total number of threads ($C$); (from bottom up)The first Y-axis depicts energy consumption in Joules; The second Y-axis is for average total power dissipation per thread ($P_t$); The third Y-axis shows total execution time ($T$); $P_{AI}$ stays around 137W. CPU frequency is set to 2.4GHz.

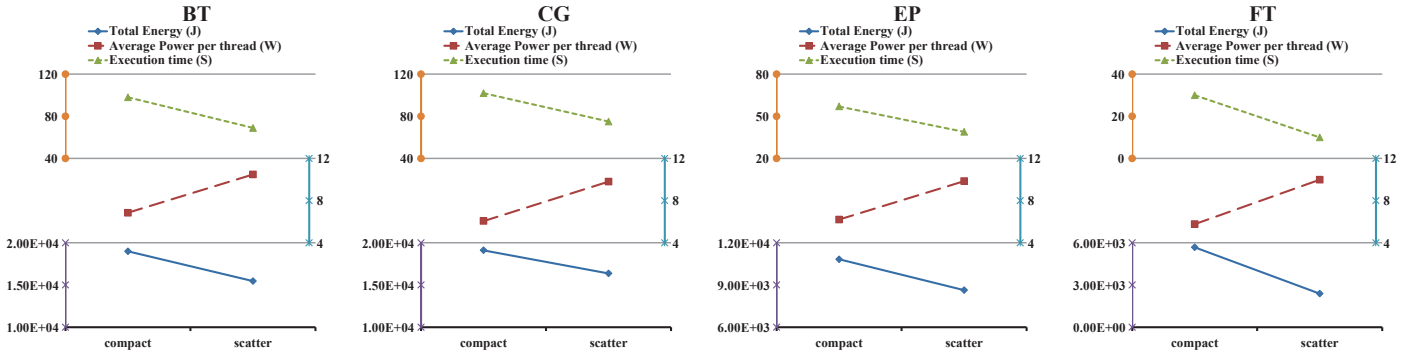


Fig. 2. Execution time, average power dissipation per thread, and total energy consumption of 8 thread version of NPB-OMP benchmark; The X-axis represents different thread mapping strategy; (from bottom up)The first Y-axis depicts energy consumption in Joules; The second Y-axis is for average total power dissipation per thread ($P_t$); The third Y-axis shows total execution time ($T$); $P_{AI}$ is relative stable.

of a large amount of static power due to the smaller transistor size ,"race to idle" plays a vital role to achieve most efficiency configuration. In addition, it is worth noting that increasing concurrency level always generates positive speedup results in the tested benchmarks if no simultaneous multi-threading (SMT) is considered, which indicates the improved balance between multic-ore CPU and memory subsystem in terms of speed. The improved memory performance mainly can be attributed to the NUMA architecture. Moreover, although SMT can be effective in most of cases, its usage depend on data demands of the workload.

**Case Study 2: Thread Mapping:** Maintaining a constant CPU frequency and workload concurrency level, the organization of threads on a system also affects total energy efficiency. This technique is known as thread mapping. Compact thread mapping means that the threads are allocated to one processor as much as possible. This approach reduces data access latency since sibling threads are sharing the same cache at some level. Scatter scheme assigns thread evenly to each processor, which reduces off-chip resource contention, such as LLC. Figure 2 shows the effects of thread mapping on energy consumption when eight threads are used. All benchmarks exhibit the same behavior. A system consumes less power by using only one processor. However, execution time is reduced considerably if two processors are deployed together. On average, there is 33% execution time reduction. The combined result is that a total of 22% energy saving can be achieved if scatter thread mapping is used. Speedup probably comes from fully utilize front bus with two sockets. The results do not necessarily show that scatter mapping scheme outperforms a compact mapping scheme in terms of energy efficiency in all cases. The major advantage of using scatter policy is that an additional LLC can be involved in the computing. In other words, compact policy can be efficient if the working set size is small enough., in which case, the compact policy will consume less power (small $P_t$) with a limited amount of performance loss (larger $T$). Because of this characteristics, compact policy can be used for power capping as well. We discuss it in more details in the next case study.

**Case Study 3: DVFS** As we discussed in Section II-B,

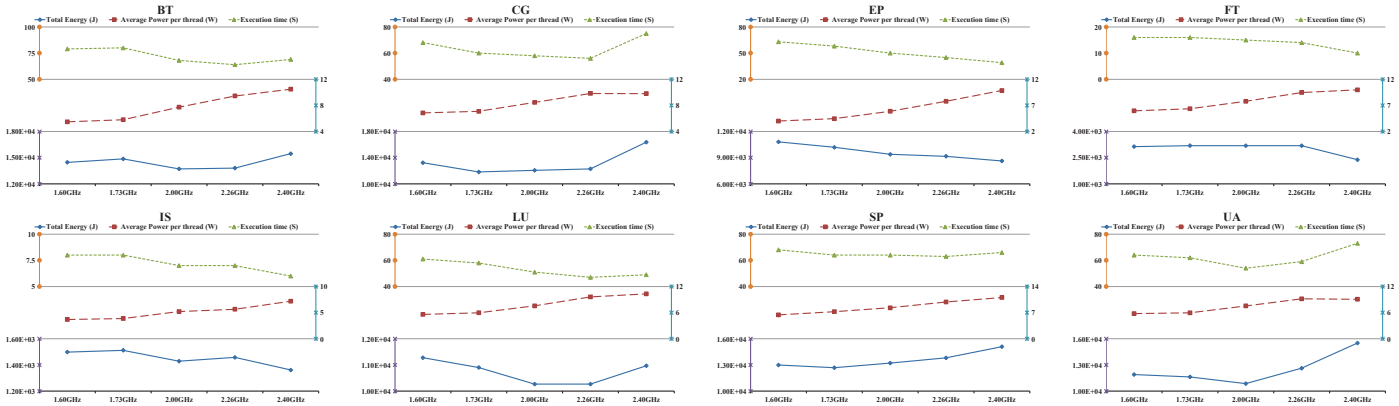


Fig. 3. Execution time, average power dissipation per thread, and total energy consumption of 8 thread version of NPB-OMP benchmark; The X-axis represents different CPU frequencies; (from bottom up)The first Y-axis depicts energy consumption in Joules; The second Y-axis is for average total power dissipation per thread ($P_t$); The third Y-axis shows total execution time ($T$); $P_{AI}$ is relative stable and stays around 137W for all the cases. Scatter is used as the thread mapping scheme.

DVFS is an effective way of reducing $P_t$. However, execution time increases because of the compromised computation capability. Figure 3 shows the effects of tuning the CPU frequencies for different benchmarks. It is obvious that the most energy efficient frequency is depend on the particular workload. For example, at 2.00 GHz, BT, LU, and UA benchmark achieve most energy efficiency among all the different among all the different frequencies. While, for SP and CG, it is 1.73GHz. In addition, a finer tuning can be made for each benchmark to achieve better energy efficiency. We pick IS benchmark as an example to show the effects. We use Intel Core 2 Quad 8200 as a experiment platform in this study to measurement CPU power dissipation directly from the power supply. We use SPAN [8] to record different functions activities in IS benchmark. There are three major steps in IS.A: `create_seq()`, `rank()`, and `full_verify()`. In this example, the `rank()` function, which produces approximate 0.15 Instructions Per Cycle (IPC), is less CPU-bound during its execution (IPC values are used broadly as CPU power model input [8]). For example, we are able to use DVFS to scale down CPU frequency from 2.34GHz to 2.00GHz during the execution of `rank()`. As Figure 4 shows, we achieved 24% energy saving with 3% performance loss. On the contrary, 10% energy saving is achieved with 10% performance loss if we scale down CPU frequency during the execution of `create_seq()`, which has higher IPC value during its execution (around 0.6). In addition, we observe that multiplication operations are intensively used in the source code of `create_seq()`, while `rank()` function mainly contains branch-prediction and data movement operations. Hence, tuning $P_t$ using DVFS will effect both execution time and power dissipation. In order to improve $\mathscr{E}$, such a technique needs to be carefully applied.

Given a cap power dissipation [12], using thread mapping and/or DVFS can control the power dissipation of a system. The CPT model also can be used to demonstrate this situation. Since $P_{AI} + C \times P_t$ is fixed (due to the cap), the system configuration that generates highest performance provides maximum energy efficiency. The specific configuration, however, depends on the workload and the system. Figure 5 illustrates this scenario. Although either applying compact mapping or slower CPU speed reduces power dissipation, the performance loss introduced by DVFS is less for SP benchmark; while it is the opposite situation for EP benchmark. The advantage of using compact threading mapping includes allocating the data to a relative closer cache, other than the remote cache. A recent study [13] shows that memory performance can be affected by DVFS, which should be analyzed based on various computer systems. In other words, either using a different thread mapping strategy or DVFS, the off-chip data access can be affected. However, compact thread mapping produces a more energy efficient result if the workload phases tend to be computation intensive. EP benchmark can be considered as an extreme case. Although the DVFS strategy reduces off-chip data access bandwidth as well, it utilizes all the LLC from both processors, which results in a higher performance gain for a certain set of benchmarks.

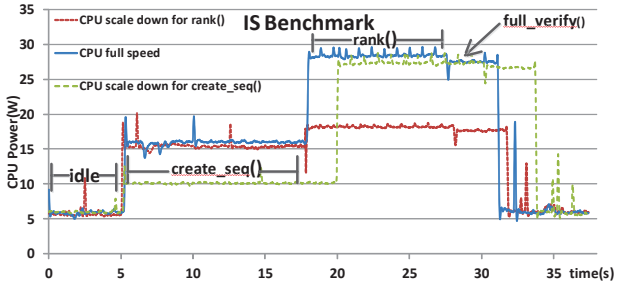


Fig. 4. Power dissipation of IS on a Intel Core2 Quad 8200 processor with different DVFS setting.

## IV. Related Work

In the data centers, regarding the effectiveness of the power usage, Green Grid first officially introduces the equation:

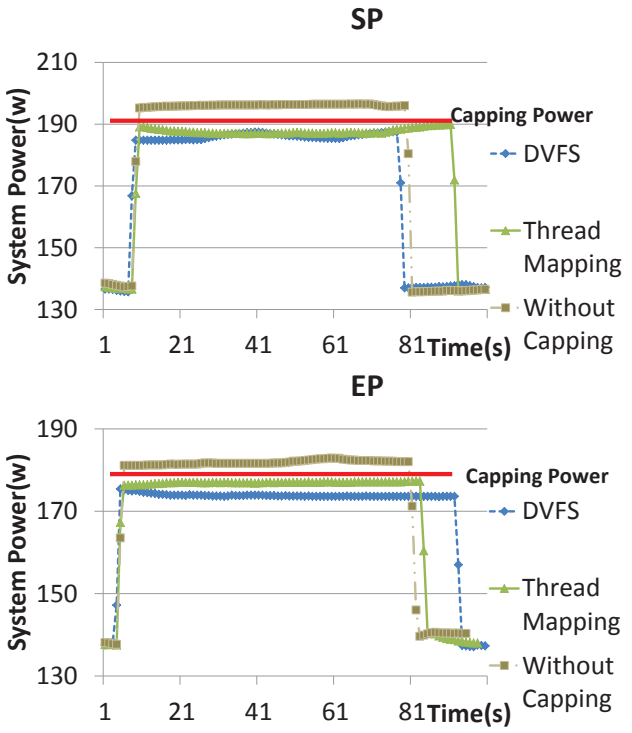


Fig. 5. Power dissipation and execution time using different power capping techniques.

$PUE = Total\ Facility\ Power / IT\ Equipment\ Power$ [14]. The formula is widely used to evaluate the efficiency of a whole data center design. The proposed CPT model is complementary with PUE in the sense that CPT emphasizes useful work produced by a system.

In-depth understanding of software power dissipation becomes one of the major considerations when designing power-aware systems. Ge *et al.*, propose PowerPack [6] to generate component level power profiles. This approach targets on cluster level. PowerPack provides APIs to synchronize external power measurement and function execution of the target application. However, manual instrumentation is inconvenient for large scale applications. Hänig *et. al*, propose SEEP [15], which uses symbolic execution to explore possible code paths and entities in a program and to generate energy profiles for specific target platforms. Instruction level energy profiles are needed for each platform in advance in order to generate energy profiles for a program.

## V. Summary

In this paper, we propose a general CPT model to analyze the system energy efficiency for a given workload. Most techniques on the market can be categorized as altering one or two parameters in the proposed model. We show three case studies to illustrate how to use CPT model to analyze different techniques. In practice, each technique proposed has to be examined from aspects mentioned in Section II. Energy efficiency is closely related to the system architecture, workload characteristics, and system configurations. We expect the CPT model helps to identify the bottleneck of existing systems and serves as guidance for future energy-efficient system designs.

## References


[1] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Commun. ACM*, vol. 53, pp. 50–58, March 2010. [Online]. Available: http://doi.acm.org/10.1145/1666420.1666438

[2] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," in *Custom Integrated Circuits Conference, 1997., Proceedings of the IEEE 1997*, may 1997, pp. 479 –482.

[3] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson, S. Bennett, A. Kagi, F. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48 – 56, may 2005.

[4] ACPI, "Advanced Configuration Power Interface," http://www.acpi.info/.

[5] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 13–23. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250665

[6] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, 2010.

[7] B. Steigerwald, C. D. Lucero, C. Akella, and A. Agrawal, *Energy Aware Computing Powerful Approaches for Green System Design*. Intel Press, March 2012.

[8] S. Wang, H. Chen, and W. Shi, "SPAN: A software power analyzer for multicore computer systems," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 1, pp. 23 – 34, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221053791000003X

[9] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 37–47. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815968

[10] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, dec. 2006, pp. 359 –370.

[11] W. Meters, "Watts up? .net," https://www.wattsupmeters.com.

[12] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, pp. 183–195, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10586-007-0045-4

[13] R. Schöne, D. Hackenberg, and D. Molka, "Memory performance at reduced cpu clock speeds: An analysis of current x86_64 processors," in *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, ser. HotPower'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387869.2387878

[14] A. Rawson, J. Pflueger, and T. Cader, "The green grid data center power efficiency metrics: Pue and dcie," The Green Grid, Tech. Rep., October 2007.

[15] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat, "Seep: exploiting symbolic execution for energy-aware programming," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:5. [Online]. Available: http://doi.acm.org/10.1145/2039252.2039256