Traffic-Aware Power Optimization for Network Applications on Multicore Servers [*]

Jilong Kuang, Laxmi Bhuyan and Raymond Klefstad
Computer Science & Engineering Department
Unviersity of California, Riverside
900 University Ave, Riverside, CA 92521, USA
{jkuang, bhuyan, klefstad}@cs.ucr.edu

ABSTRACT
In this paper, we design, implement, and evaluate a traffic-aware and power-efficient multicore server system by translating incoming traffic rate to appropriate system operating level, which is then translated to optimal per-core frequency configuration. According to the varying traffic rate, the system can adjust the number of active cores and per-core frequency "on-the-fly" via the use of per-core DVFS, power gating, and power migration techniques based on our new power model which considers both dynamic and static power consumption of all cores. Results on an AMD machine with two Quad-Core Opteron 2350 processors for six real network applications chosen from NetBench [19] show that our scheme reduces power consumption by an average of 41.0% compared to running with full capacity without any reduction in throughput. It also consumes less power than three other approaches, chip-wide DVFS [22], power gating [17], and chip-wide DVFS + power gating [15], by 35.2%, 24.3%, and 10.5% respectively.

Categories and Subject Descriptors
C.4 [**Computer Systems Organization**]: Performance of Systems—*Performance attributes, Design studies*

General Terms
Design, Performance

Keywords
Packet processing, power efficiency, multicore architecture

1. INTRODUCTION
Explosive growth of Internet high-traffic applications, such as video streaming, cloud computing and file sharing, requires orders-of-magnitude increase in system throughput. Affordable multicore servers, such as Cavium's OCTEON [2], Cisco's AON [3], and IBM's BladeCenter [6], can now meet this throughput demand. Along with increased throughput, however, comes significantly increased power consumption [8]. Collectively, millions of servers in the global network consume a great deal of power [12]. And chip manufacturers continue to increase both the number of cores and their frequencies, substantially increasing both dynamic and static power consumption.

At the hardware level, there are two main techniques to reduce power consumption. The first technique, Dynamic Voltage and Frequency Scaling (DVFS), which is widely used, reduces or increases processor voltage/frequency just enough to meet performance requirements. DVFS can be either chip-wide, where the entire chip is scaled as one unit (e.g., Intel's Foxton technology [18]), or per-core, where individual cores on the chip can be scaled at different rates (e.g., AMD's Opteron processor [1]). The second hardware-level technique, called power gating, minimizes leakage current when a core is inactive by powering it down almost completely. Power gating has been introduced only recently by major chip manufacturers (e.g., Intel's Nehalem [14]).

To determine when to reduce power to the core, various application run-time characteristics are exploited, such as program phase analysis [10], degree of parallelism [15], and time slack detection [20]. We see great potential power-saving opportunity, however, in an additional aspect: *network traffic*. Computing power needs fluctuate dramatically with the large fluctuations in network traffic. For example, Figure 1 shows real-time network traffic in a typical day monitored by Equinix data center [4] at San Jose, CA. The traffic rate varied from 320K packets/s to 720K packets/s. Power consumption could be greatly reduced when traffic is low.

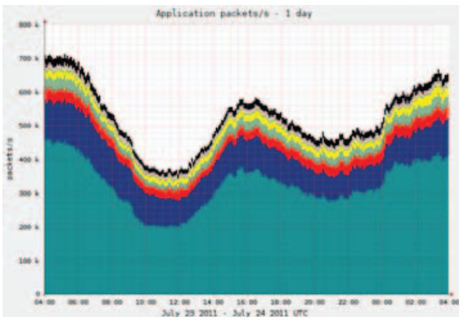


Figure 1: Traffic variation versus time in 24 hours. Different colors represent the breakdown of different packet types.

Existing studies that consider traffic variation, however, are limited in the following two ways:

1. **Dynamic Power Only**. They assume that dynamic power dominates total power consumption, and that static power can be ignored [11, 17, 22]. However, static power has increased dramatically with increases in device speed and chip density. According to a projection by the International Technology Roadmap for Semiconductors, leakage power increases its dominance of total power consumption as semiconductors progress toward 32nm [7].
2. **Single Dimensional**. Traffic-aware studies focus either on single-core platforms and chip-wide DVFS [22], or adopt power gating only [11, 17]. Thus, these approaches cannot be applied to multicore systems that support both per-core DVFS and power gating.

Using a combination of per-core DVFS and power gating can potentially minimize power consumption when network traffic is low. However, with this approach, cores perform different amounts of work. Not all cores actively run all the time, and each core may run at a different frequency. Some cores may then be stressed more than others, and overworked cores will generate excess heat, increasing static power consumption exponentially with temperature [23]. It is therefore advisable to migrate active cores periodically for lower peak core temperature and less static power consumption. A software approach called *power migration* can be used to achieve thermal load balancing across the cores. Locations of more- and less-active cores can be dynamically changed according to some policy while keeping the same system operating level. Given the same amount of heat generation depending on the number of active cores and core frequency, power migration can

[*]This work was supported by National Science Foundation grants CNS-0832108 and CSR-0912850.

redistribute the generated heat in space and time to reduce peak core temperature and improve thermal uniformity.

This paper describes a power-efficient multicore server system for network applications which dynamically adjusts system operating level and per-core frequency configuration based on incoming traffic rate. Our on-line algorithm optimizes a novel power model that considers both dynamic and static power. The dynamic per-core frequency configuration is achieved through a combination of per-core DVFS, power gating, and power migration.

We first derive a formula to translate traffic arrival rate to required cumulative core frequency. Then, based on our power model, we derive the optimal system operating level to maintain sufficient system throughput for the current traffic while using minimal dynamic power. Lastly, because each core may be configured at a different operating frequency, we migrate active cores in the system periodically to achieve thermal balancing and reduce peak core temperature. To the best of our knowledge, we are the first to target power optimization considering both dynamic and static power for network applications running on multicore servers.

To verify our design, we implement our approach on a multicore server system with varying traffic loads, running six real network applications from NetBench [19]. Our approach reduces power consumption by an average of 41.0% compared to running with full capacity without any reduction in throughput. Our approach also outperformed three other approaches with negligible overhead: chip-wide DVFS [22], power gating [17], and a hybrid combination of chip-wide DVFS and power gating [15].

The rest of this paper is organized as follows: Section 2 presents our system design which includes the traffic-aware power optimization scheme in a three-step approach. Section 3 presents our implementation and performance evaluation. Finally, Section 4 concludes this paper.

2. TRAFFIC-AWARE POWER OPTIMIZATION

2.1 System Design

The typical application supported by this work runs on a multicore server and processes a stream of network requests. Figure 2 shows the system overview, where incoming packets from the network are first stored in a global FIFO queue and then scheduled to proper cores for packet processing. The core component is system manager, which consists of four functional modules: traffic monitoring, power managing, core configuring, and task scheduling.

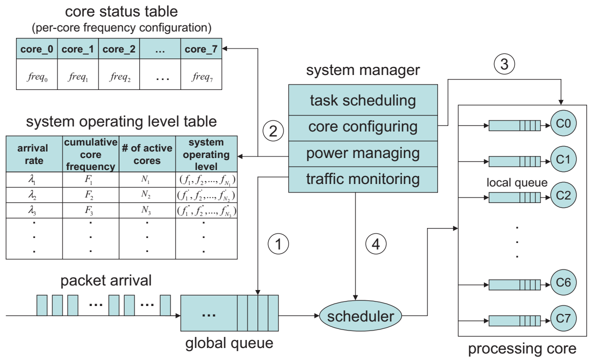


Figure 2: Overview of the traffic-aware system.

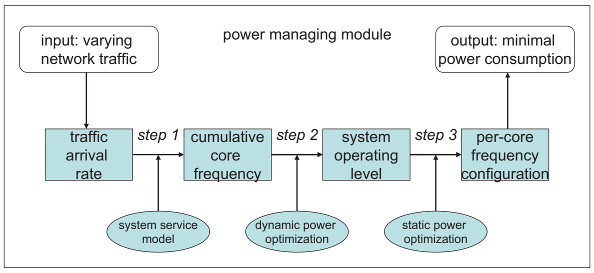
1. **Traffic monitoring module** tracks packet inter-arrival times to obtain the packet arrival rate and detect the rate change point whenever the traffic rate varies. We monitor the traffic and detect the rate change point by using the sampling technique based on maximum likelihood ratio [21, 22], which is particularly useful for server/router deployment where traffic changes can be predicted only based on prior information.

2. **Power managing module** manages two runtime tables, a system operating level table, which caches optimal system operating level for a given traffic arrival rate, and a core status table, which tracks the actual per-core frequency configuration. The system operating level is represented by tuple $(f_1, f_2, ..., f_N)$ throughout this paper, where $f_1 \geq f_2 \geq ... \geq f_N$. This tuple indicates that N cores are active running and the ith core has the frequency f_i $(1 \leq i \leq N)$. Given a certain arrival rate, the power managing module appropriately derives the optimal system operating level based on our dynamic power optimization scheme. In addition, it also initializes the core status table at each traffic rate change point, and periodically updates the core status table to enable power migration for active cores between two consecutive traffic rate change points based on our static power optimization scheme.

3. **Core configuring module** adjusts the frequency level of each core based on the information from the core status table managed by the power managing module. At each configuration point, it applies power gating to cores labeled as inactive as soon as their local queues become empty, and applies per-core DVFS for cores labeled as active and adjusts their frequencies according to their respective configurations chosen from one of the five frequency levels, namely 1GHz, 1.2GHz, 1.4GHz, 1.7GHz and 2GHz. The core configuring module is critical in the system because it is where the three applied power techniques are actually enabled.

4. **Task scheduling module** appropriately schedules packets in the global queue to active cores in our system. When the per-core frequency configuration updates, the scheduler stops sending packets to power-gated cores. For active cores with different core frequencies, the scheduler distributes the workload, which is determined by the number of packets, per-packet size, and application type, in proportion to the core frequency. This approach achieves weighted load balancing across cores under varying traffic rate and avoids loss of system throughput due to the change of system operating level and update of per-core frequency configuration.

As we focus on the power managing module in this paper, we propose a three-step approach as shown in Figure 3 to solve the power optimization problem.



input: varying network traffic
power managing module
output: minimal power consumption
traffic arrival rate
step 1
cumulative core frequency
step 2
system operating level
step 3
per-core frequency configuration
system service model
dynamic power optimization
static power optimization

Figure 3: A three-step power optimization scheme.

2.2 Step 1: System Service Model

The system service model translates the traffic arrival rate to required cumulative core frequency in the multicore system. As traffic rate varies, an ideal cumulative core frequency should be just sufficient to satisfy the demand without over-provisioning.

First, we let service rate equal to arrival rate, because 1) to guarantee a stabilized system without packet overflow, service rate should be no less than arrival rate, and 2) to avoid over-provisioning and achieve power efficiency, service rate should be no greater than arrival rate.

Second, [9, 22] have shown that the service rate is linearly proportional to the CPU frequency for a single-core system. However, because we target multicore architectures where different cores may run at different frequencies, it is necessary to re-think and justify the relationship between the service rate and cumulative core frequency. We, therefore, conduct two empirical trace-driven studies with 6 chosen network applications from NetBench on our multicore machine to help establish the relationship.

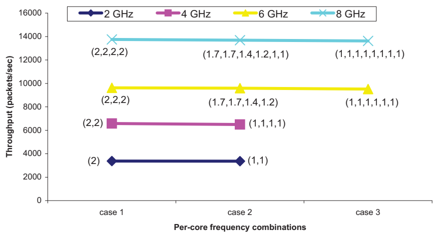


Figure 4: Throughput versus frequency combinations.

The first study examines the effect of various per-core frequency combinations versus throughput given the same cumulative core frequency. Figure 4 shows the results of the URL application when we vary the cumulative core frequency from 2GHz to 8GHz. For each cumulative core frequency, we change the per-core frequency combinations. From this figure, we observe that the throughput (service rate) only depends on cumulative core frequency, regardless of per-core frequency combinations. This is because when incoming packets are processed on multiple cores with different service rates, we can equivalently treat this multicore server as a single-core system with the aggregated service rate equal to the sum of per-core service rate.

The second study builds the relationship between the service rate and cumulative core frequency in our system. We vary the cumulative core frequency from the minimum (1GHz) to the maximum (16GHz) and record the system throughput. The results show that for our multicore server, the throughput (service rate) is also linearly proportional to the cumulative core frequency. Figure 5 illustrates both the experiment result and the fitted line for the URL application (Other applications have the similar results with different parameters and coefficients). Therefore, our system service model for the URL application is given by the linear function in Equation 1, where X represents cumulative core frequency and Y represents the service rate, or the arrival rate in our case.

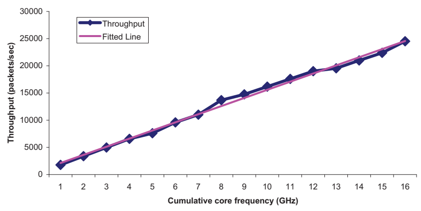


Figure 5: Throughput versus cumulative core frequency.

$$Y = 1496 \cdot X + 628 \tag{1}$$

2.3 Step 2: Dynamic Power Optimization

The dynamic power optimization scheme takes cumulative core frequency as input and produces the optimal system operating level as output. More specifically, it answers the following two questions: Q1) what is the theoretically optimal number of active cores from our power model? Q2) what is the frequency assignment for active cores considering the discrete frequency levels?

2.3.1 Power Model

Consider a network application running on a core at voltage v and frequency f. The dynamic power consumption is given by: $P_{dynamic} = K_a \cdot f \cdot v^2$, where K_a is a task/core dependent factor determined by the switched capacitance. Besides, the frequency f is almost linearly related to the voltage $v : f = K_b \cdot (v - v_t)^2/v$, where v_t is the threshold voltage and K_b is a constant. For a sufficiently small threshold voltage, the frequency is approximated to $K_b \cdot v$. Therefore, we assume the dynamic power consumption is cubic to the frequency as shown in Equation 2, where $K = K_a/K_b^2$.

$$P_{dynamic} = K \cdot f^3 \tag{2}$$

With respect to static power, we assume that for power-gated cores, they consume zero static power. For active cores, static power consumption is exponential to core temperature [23]. However, as this step focuses on dynamic power optimization, we ignore the temperature effect and assume the static power of each active core is constant, P_s. Detailed discussion for static power consumption is given later because it is related to thermal balancing and power migration. Thus, the total power consumption of an active core is given by Equation 3:

$$P_{core} = P_{static} + P_{dynamic} = P_s + K \cdot f^3 \tag{3}$$

In a multicore system with N active cores, suppose f_i is the frequency on core i and $P(f_1, f_2, ..., f_N)$ is the total system power consumption as a function of system operating level denoted as $(f_1, f_2, ..., f_N)$. We have the following:

$$P(f_1, f_2, ..., f_N) = N \cdot P_s + K \cdot (f_1^3 + f_2^3 + ... + f_N^3) \tag{4}$$

In the following, we focus on answering Q1 in a quantitative approach under the assumption that the core frequency is continuous. Later on, to answer Q2, we will relax this constraint in practical scenario with discrete frequency levels.

Suppose we have x active cores to handle cumulative core frequency F. As the dynamic power is proportional to the cube of core frequency, we know that when every active core is running at the same frequency of F/x, the total dynamic power consumption reaches minimum. Thus, from Equation 4, we can derive the total power consumption as follows.

$$P = P(f_1, f_2, ..., f_x) = x \cdot P_s + K \cdot (f_1^3 + f_2^3 + ... + f_x^3) \tag{5}$$
$$\geq x \cdot P_s + x \cdot K \cdot (F/x)^3 = x \cdot P_s + \frac{K \cdot F^3}{x^2}$$

This function ($P = x \cdot P_s + \frac{K \cdot F^3}{x^2}$) is a unimodal function and has a global minimum as illustrated in an example in Figure 6. This curve is drawn for the URL application when we set $P_s = 5.8$, $K = 1.6$ and $F = 3$. More details about the parameters can be found in Section 3. It shows that starting from a single active core ($x = 1$), increasing the number of active cores will reduce the total power consumption while satisfying the cumulative core frequency requirement, until the number of active cores increases past a certain threshold value. We call this value x^*, which is the optimal number of active cores that strikes a good balance between static and dynamic power. In fact, from the classic algebra inequality as shown in Equation 6, we can easily solve the problem.

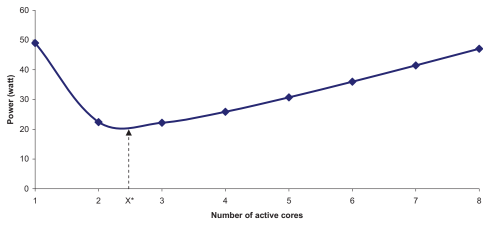


Figure 6: Power consumption as the number of active cores varies given the same cumulative core frequency.

$$\frac{a + b + c}{3} \geq \sqrt[3]{a \cdot b \cdot c} \tag{6}$$

when a=b=c, left side reaches minimum.

$$\begin{aligned} P &= x \cdot P_s + \frac{K \cdot F^3}{x^2} = \frac{x \cdot P_s}{2} + \frac{x \cdot P_s}{2} + \frac{K \cdot F^3}{x^2} \\ &\geq 3 \cdot \sqrt[3]{\frac{P_s^2 \cdot K \cdot F^3}{4}} \end{aligned} \tag{7}$$

In addition, the minimal power consumption is achieved if and only if Equation 8 is satisfied.

$$\frac{x \cdot P_s}{2} = \frac{K \cdot F^3}{x^2} \Rightarrow x^* = \sqrt[3]{\frac{2K \cdot F^3}{P_s}} \tag{8}$$

2.3.2 Frequency Assignment

After obtaining the optimal number of active cores, we address Q2, the frequency assignment problem to appropriately assign frequency to each active core considering the discrete frequency levels. We propose two rules to guide the frequency assignment [1].

- Rule 1: *Always provide the minimal cumulative core frequency that satisfies the traffic demand.*
- Rule 2: *For a given cumulative core frequency, the per-core frequency combination with the least standard deviation consumes the least power.*

To demonstrate the two rules, we carry out two empirical studies with the same settings as in Section 2.2. In the first study, we vary the cumulative core frequency from 2GHz to 8GHz. For a given cumulative core frequency, we vary the per-core frequency combinations and record the net power consumption (load power minus idle power). Figure 7 shows the results for the URL application. From this figure, we observe that power consumption varies substantially, as much as 114% when comparing $(2, 2)$ to $(1, 1, 1, 1)$, with different per-core frequency combinations, which indicates that a proper frequency assignment is very necessary for multicore servers supporting per-core DVFS and power gating.

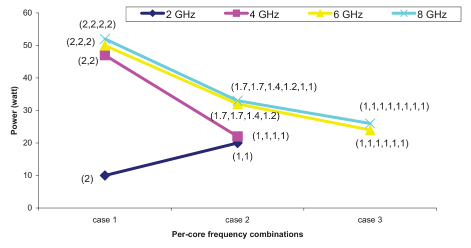


Figure 7: Power versus per-core frequency combinations.

In the second study, we take two cores and change the frequency in all possible combinations and record the power consumption.

[1]Algorithm pseudocode is omitted due to space limit. Interested readers can refer to [13] for details.

Figure 8 shows the 3D plot for the results with the URL application, where each black point represents a per-core frequency combination and its corresponding power consumption. In addition, based on Figure 8, we also plot Figure 9 illustrating the power consumption versus cumulative core frequency. When a certain cumulative core frequency corresponds to multiple power consumptions, we take the minimal one. From these two figures, we notice: 1) If we only consider the minimal power consumption for a given cumulative core frequency as shown in Figure 9, we see higher cumulative core frequency corresponds to higher power consumption. 2) For the same cumulative core frequency, the more evenly-distributed per-core frequency combination results in less power consumption as shown in Figure 8. For example, point $(1.7, 1.7)$ is lower than point $(2, 1.4)$ and point $(1.2, 1.2)$ is lower than point $(1.4, 1)$, although they have the same cumulative core frequency in both cases. In summary, this exhaustive study empirically validates our two rules.

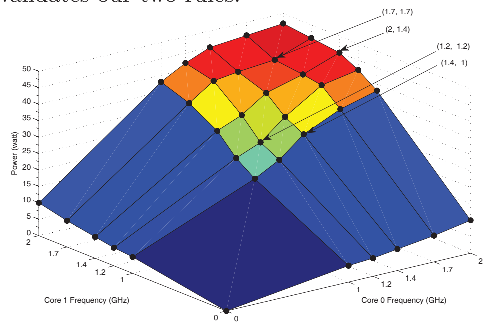


Figure 8: Power versus two-core frequency combinations.

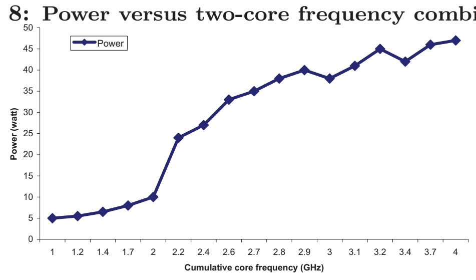


Figure 9: Power versus cumulative core frequency (two cores).

2.4 Step 3: Static Power Optimization

The static power optimization scheme takes system operating level as input, which virtually contains an array of core frequencies optimized for a given traffic rate, and produces as output the actual per-core frequency configuration that is dynamically updated for power migration. This step focuses on the power migration design for active cores to achieve thermal balancing and reduce peak core temperature that effectively reduces static power consumption.

2.4.1 *Design Overview*

While keeping the system operating level constant, we appropriately vary the physical location for active cores so that thermal balancing is achieved across all cores. If the time interval between two migration points is small enough, we can minimize peak core temperature and effectively reduce static power consumption. Figure 10 illustrates the overview of our power migration scheme with varying network traffic rate. At each traffic change point (T_i), we apply the dynamic power optimization scheme to obtain the optimal system operating level. Because the number of active cores may be less than the total number of cores, and per-core frequency is heterogeneous, we periodically redistribute the power dissipation at each migration point (t_i) among all cores. In Figure 10, color squares represent active cores with different frequency levels (the darker the color, the higher the frequency), whereas white squares represent power-gated cores. In this example, the migration process happens among core pairs (C1, C2), (C6, C4), and (C5, C3), where the highest frequency cores C1, C6 and C5 are swapped with the lowest frequency cores C2, C4 and C3.

2.4.2 *Migration Policy*

The policy of our power migration consists of both long-term update and short-term update of core status table. The long-term update refers to the initialization of core status table at each traffic change point, which is in the order of minutes based on network traffic studies in [11, 17, 22]. The short-term update refers to the periodic update of core status table at each migration point between two consecutive traffic change points. Considering core thermal behavior, packet processing time and system reconfiguration overhead, we find an update frequency of 1 second to be a good value for short-term update in our system.

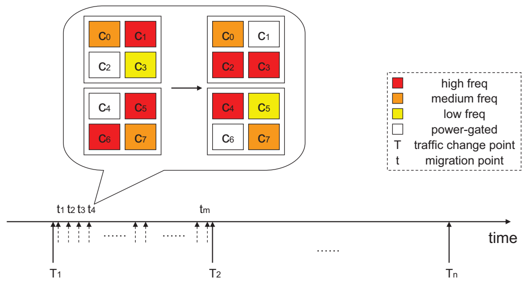


Figure 10: Illustration of power migration for active cores.

Long-term update: The long-term update should be based on previous history in the core status table for thermal balancing. At the traffic change point, because the system operating level will change in terms of the number of active cores and core frequency, we want every core to have even power dissipation over a period of time. Therefore, our long-term update can be described as follows:

General policy: Given the current system operating level (f_1, f_2, ..., f_N)*, we first sort the per-core frequency configuration in the previous core status table according to the frequency level from the lowest to highest. Then, we assign frequency* f_1 *to the first core in that list and frequency* f_2 *to the second core in that list and so on. For cores that are not assigned a frequency level, we leave them to be power-gated.*

We have one exception for the above-mentioned long-term update. As we target servers with multicore processors, we should use as few processors as possible while satisfying traffic demand. Thus, when all active cores can fit into one processor, we should always use only one processor. Considering the general policy, we add the following exception rule:

Exception: If all active cores can fit into one processor, we choose the processor which contains the core that is assigned the frequency f_1.

Short-term update: The short-term update aims to achieve thermal balancing across all cores in the system through power migration. We argue that the migration policy has to be temperature-aware to guarantee thermal balancing during two consecutive short-term updates. Because core frequency is directly related to core temperature and per-core frequency is easy to obtain, we propose a frequency-aware migration policy to guide the short-term update as follows:

General policy: We sort the per-core frequency configuration in the current core status table according to the frequency level from the lowest to highest. Then, we swap the frequency between the first core in the list and the last core, and between the second core and the last but one core, and so on.

This strategy lets the power dissipation be evenly distributed across all cores during two consecutive short-term updates; thus overall thermal balancing will be achieved as expected. In the exceptional case where only one processor is used, we apply the following rule:

Exception: If all active cores can fit into one processor, we switch the active processor at every migration point and copy the same per-core frequency configuration within a processor from one to the other. However, at every other migration point, we update the per-core frequency configuration within a processor following the short-term update general policy.

This exception rule ensures we will keep the frequency assignment to one processor when it is possible. Using the regular short-term policy without this exception will likely split the frequency assignment across multiple processors.

3. EXPERIMENTAL EVALUATION

3.1 Experiment Setup

We implement our scheme along with three other schemes on an AMD server with two Quad-Core Opteron 2350 processors. For power measurement, we use a power analyzer (model EXTECH 380801 [5]) to obtain the real-time whole system power. We use the net power consumed exclusively by network applications as the metric for fair comparison. Net power is obtained by subtracting idle power from load power.

In our experiment, per-core DVFS is achieved by setting the core frequency to one of the five predefined frequency levels: 1GHz, 1.2GHz, 1.4GHz, 1.7GHz and 2GHz. We rely on the Linux kernel CPUfreq subsystem to implement the frequency scaling. Power gating is achieved by removing cores from active working set based on kernel's built-in CPU "hotplug" support, which mimics precisely the behavior of power gating [16]. Task scheduling module achieves power migration by dynamically scheduling incoming packets to active cores.

We parallelize six network applications from NetBench [19] (as listed in Table 1) and execute them in a multi-threaded fashion with packet-level parallelism. To guarantee each active core is running a thread, we enforce thread-to-core binding by setting thread affinity. We select two applications from each category (i.e., Micro-level, IP-level and Application-level). The packet trace is from NetBench with $10,000$ packets, which are repeatedly processed in our experiment. The packet size ranges from 40 bytes to 1500 bytes with an average of 723 bytes. The routing table size for TL, Route and DRR is 128, and we use the small_input file for URL.

Table 1: Six network applications from NetBench.

Name	Functionality	Category
CRC	CRC-32 checksum calculation	Micro level
TL	Radix-tree table lookup routine	Micro level
Route	IPv4 routing based on radix	IP level
DRR	Deficit-round robin scheduling	IP level
URL	URL-based switching	Application level
MD5	Message digest algorithm	Application level

Table 2: Application-specific parameters.

App.	System Service Model	Latency (μs)	K
CRC	$Y = 81109 \cdot X + 26662$	0.008·size+0.3	1.5
TL	$Y = 571389 \cdot X + 328743$	0.8	1.4
Route	$Y = 253707 \cdot X + 87975$	1.8	1.4
DRR	$Y = 74965 \cdot X + 54945$	5.5	1.4
URL	$Y = 1496 \cdot X + 628$	0.131·size+73.2	1.6
MD5	$Y = 76016 \cdot X + 35024$	0.005·size+3.2	1.8

Table 2 shows application-specific parameters. We profile each application and obtain their system service model, where X represents the cumulative core frequency [2] and Y represents the service rate (packets/sec), equivalent to the arrival rate. To quantify the workload for weighted load balancing scheduling, we also obtain the per-packet latency for each application when running on a single core with 2GHz frequency. It is worth noting that our method also applies to non-linear applications, as long as we can model and translate the traffic arrival rate to cumulative core frequency (step 1 in Figure 3). This is because step 2 and step 3 are solely based on the result of step 1. We derive the dynamic power parameter K for each application based on Equation 2 and Equation 3 by substituting known frequency and measured power consumption. In addition, to calculate the static power P_s, we refer to manual specification, and use $V_{dd} \in (1.06V, 1.35V)$ and $I_{leak} \in (4.2A, 5.3A)$ as the $65nm$ technology parameters [16]. Hence, we take the average of 5.8W as the input for our power model.

To achieve traffic variation, we experiment with both synthetic and real-world workloads. For synthetic workload, we set the required cumulative core frequency (F) for incoming traffic to be one of the following five cases (as shown in Table 3). For real-world workload, we take the 24-hour traffic as shown in Figure 1. We consider the total volume as the arrival traffic for packet processing, and sample 24 different average traffic rates at each hour to obtain the required cumulative core frequency. Without loss of generality, the cumulative core frequency is then scaled according to our system capacity from 1GHz to 16GHz. For both workloads, we change the traffic rate every minute and set the power migration frequency to be 1 second.

Table 3: Synthetic workload for different traffic rate.

Traffic	extra low	low	medium	high	extra high
F	1GHz	4GHz	8GHz	12GHz	16GHz

In the experiment, we first compare our scheme to a traffic-unaware native system without power management. In addition, we compare our scheme with three other traffic-aware schemes, i.e., PG [17], which turns off cores when traffic is light using power gating, C-DVFS [22], which assumes a unified frequency adjustment across all cores using chip-wide DVFS, and C-Hybrid [15], which combines both chip-wide DVFS and power gating.

[2]In our experiment, X is between 1GHz and 16GHz.

3.2 Power Savings

Figure 11 shows power savings percentage for our scheme under different synthetic workloads compared to a native system. We observe that our scheme can achieve power savings in four out of five rates ranging from 18.0% for the CRC application in high traffic rate, to as high as 90.0% for the URL application in extra low traffic rate. The only exception is the extra high traffic rate, where all the cores must be running at the maximal 2GHz. Overall, our scheme reduces an average of 41.0% power consumption for the six applications and their five different workloads. In addition, we find our scheme especially useful when the traffic is light, e.g., in medium, low and extra low cases. This is because under light load we have more potential to apply per-core DVFS, power gating and power migration to achieve power savings.

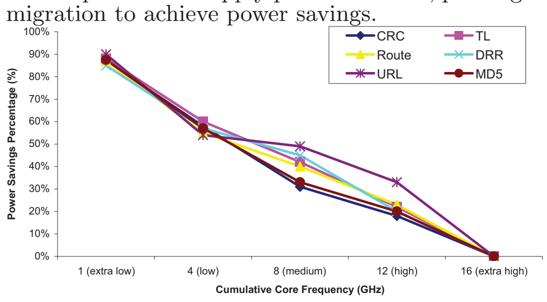


Figure 11: Power savings under different workloads.

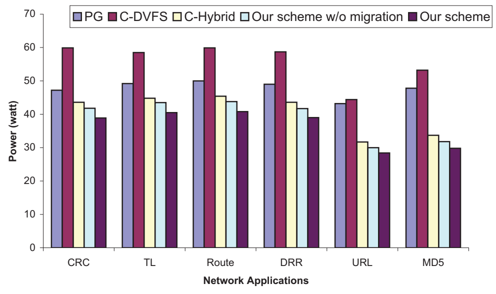


Figure 12: Power consumption with three other schemes.

Figure 12 shows the average power consumption for different applications comparing our scheme with three other schemes for the five different synthetic workloads. We observe that our scheme performs the best across all applications with an average of 35.2% power savings over C-DVFS, 24.3% over PG and 10.5% over C-Hybrid. C-DVFS performs the worst due to significant over-provisioning and excessive static power consumption, as it always keeps all the cores actively running. PG improves upon C-DVFS by turning off unnecessary cores to mitigate over-provisioning and save static power. However, without frequency scaling, it still suffers from excessive power consumption during extra low traffic. C-Hybrid outperforms both C-DVFS and PG due to its more flexible power management scheme using both chip-wide DVFS and power gating. But, C-Hybrid fails to achieve the best power savings because it does not consider static power or support more advanced per-core DVFS and power migration. Our scheme outwins all other schemes by providing the optimal system operating level and dynamically changing the per-core frequency configuration.

In addition, to emphasize the importance of power migration, we also experiment with our scheme without migration. Our scheme with power migration achieves an additional 2.5W reduction of power on average over our scheme without power migration. This highlights the advantage of including power migration in our power optimization scheme. In particular, when the traffic rate is extra low, low and medium, power migration can significantly reduce peak core temperature and hence effectively reduce static power.

3.3 Energy Savings

Figure 13 shows normalized energy consumption compared to a native system for different schemes using the real-world workload (24-hour traffic in Figure 1). We observe that all four schemes can achieve energy savings, ranging from the least energy consumption of 0.45 for the URL application in our scheme, to the most energy consumption of 0.71 for the Route and DRR application in C-DVFS scheme. However, upon averaging out all six applications over the 24-hour period, we still find our scheme outperforms PG, C-DVFS and C-Hybrid by 22.0%, 19.1% and 8.4%, respectively. The poor performance of PG and C-DVFS is due to the following two reasons: 1) PG always lets the cores run at full speed without frequency scaling, and 2) C-DVFS always has all 8 cores actively running without power gating. Compared to PG and C-DVFS, C-Hybrid improves the energy performance by combining both chip-wide DVFS and power gating. However, because C-Hybrid is unable to provide the optimal system operating level and ignores static power, it fails to achieve the best energy savings.

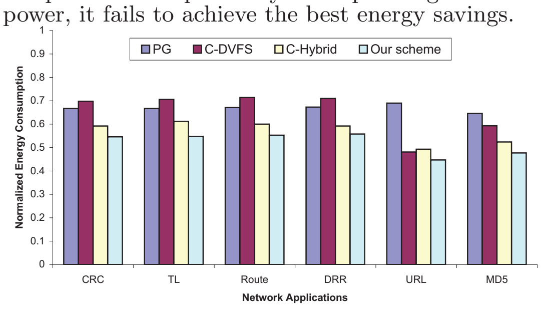


Figure 13: Normalized energy with three other schemes.

3.4 Reconfiguration Overhead

First, we individually measure the overhead for DVFS and power gating. For DVFS, it takes 0.008 seconds to change the per-core frequency level. For power gating, it takes 0.11 seconds to turn off a core and 0.08 seconds to turn on a core. In addition, we notice that in power gating, turning off a core does not add overhead as that power-gated core will be inactive in the next second. Also, not every core changes status every second. Therefore, we measure the average per-core reconfiguration overhead over the 24-hour traffic periodic at each hour as shown in Figure 14. This figure shows the result for TL, Route and DRR, which have the same system operating level with the same K value. The other three applications, CRC, URL and MD5 have very similar performance. Every second, we count the invoked number of DVFS and power gating for all the cores and divide the aggregated total overhead by 8. From this figure, we observe that the overhead ranges between 0.2% and 3.3% with an average of 1.7%, which is negligible. It is also easy to see that during low traffic hours (i.e., 8:00-14:00 and 17:00-23:00), the overhead is higher due to more frequent power migrations.

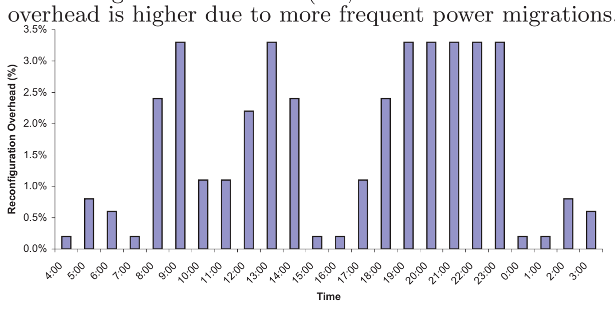


Figure 14: Overhead versus time in our scheme.

3.5 Thermal Behavior

Finally, to demonstrate the effectiveness of power migration in reducing peak core temperature, we use IPMItool utility to read processor thermal sensor and obtain the temperature for each processor every second. Figure 15 shows the maximal temperature increase at extra low (XL), low (L) and medium (M) traffic rate, where power migration is playing a significant role. Since all the starting temperatures are the same, we can see our scheme has the minimal peak core temperature in all cases. In this figure, DVFS represents DVFS-based schemes, including both C-DVFS and C-Hybrid, as they have the same thermal behavior.

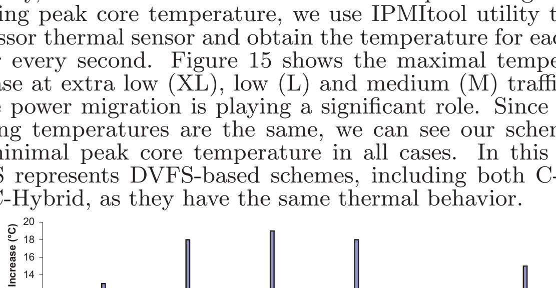


Figure 15: Temperature under different workloads.

More specifically, we see that PG causes the highest temperature increase (up to 19°C for the Route application in medium traffic rate) because it lets all the cores run at the maximal frequency all the time. DVFS-based schemes, on the other hand, achieve better thermal behavior with frequency scaling, especially in extra low traffic rate. However, it still suffers from 1°C to 3°C higher peak core temperature compared to our scheme when the traffic rate is low or medium, as it always stresses the same active cores. We observe that our scheme on average reduces peak core temperature by 6°C compared to PG in all traffic rates and by 2°C compared to DVFS in low and medium traffic rate. This observation clearly shows that our scheme is able to achieve thermal balancing and keep a lower peak core temperature through power migration.

4. CONCLUSION

We design, implement, and evaluate a traffic-aware and power-efficient multicore server system that appropriately changes the system operating level according to varying traffic rate and dynamically adjusts the per-core frequency configuration using a combination of per-core DVFS, power gating, and power migration techniques to minimize power consumption. Our experimental results show that on an average our system saves 41.0% power compared to a native system. It also consumes less power than three other approaches, C-DVFS [22], PG [17], and C-Hybrid [15], by 35.2%, 24.3%, and 10.5% respectively.

5. REFERENCES

[1] AMD Opteron Processor. http://www.amd.com/opteron.
[2] Cavium OCTEON Processor Family. http://www.caviumnetworks.com/OCTEON_MIPS64.html.
[3] Cisco AON Technology. http://www.cisco.com/en/US/products/ps6692/Products_Sub_Category_Home.html.
[4] Equinix-sanjose. http://www.caida.org/data/monitors/passive-equinix-sanjose.xml.
[5] EXTECH Power Analyzer. http://www.extech.com/instruments.
[6] IBM BladeCenter System. http://www-03.ibm.com/systems/bladecenter/.
[7] International Technology Roadmap for Semiconductors. http://public.itrs.net.
[8] K. Greene. Data centers' growing power demands. *MIT Technology Review*, 2007.
[9] C. Hughes, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proc. of Micro '01*, 2001.
[10] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proc. of Micro '06*, 2006.
[11] R. Kokku, U. B. Shevade, N. S. Shah, M. Dahlin, and H. M. Vin. Energy-efficient packet processing. *UT-Austin Technical Report TR04-04*, 2004.
[12] J. Koomey. Estimating total power consumption by servers in the us and the world. *Analytics Press*, 2007.
[13] J. Kuang, D. Guo, and L. Bhuyan. Power optimization for multimedia transcoding on multicore servers. In *Proc. of ANCS '10*, 2010.
[14] R. Kumar and G. Hinton. A family of 45nm ia processors. In *Proc. of ISSCC '09*, 2009.
[15] J. Lee and N. S. Kim. Optimizing throughput of power- and thermal-constrained multicore processors using dvfs and per-core power-gating. In *Proc. of DAC '09*, 2009.
[16] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power management of datacenter workloads using per-core power gating. *HP Labs Technical Report HPL-2009-326*, 2009.
[17] Y. Luo, J. Yu, J. Yang, and L. Bhuyan. Conserving network processor power consumption by exploiting traffic variability. *ACM TACO*, 2007.
[18] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger. Power and temperature control on a 90-nm itanium family processor. *Journal of Solid-State Circuits*, 2006.
[19] G. Memik, W. H. Mangione-Smith, and W. Hu. Netbench: A benchmarking suite for network processors. In *Proc. of ICCAD '01*, 2001.
[20] R. Mishra, N. Rastogi, and D. Zhu. Energy aware scheduling for distributed real-time systems. In *Proc. of IPDPS '03*, 2003.
[21] J. W. Pratt. F. y. edgeworth and r. a. fisher on the efficiency of maximum likelihood estimation. *The Annals of Statistics*, 1976.
[22] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic voltage scaling for portable systems. In *Proc. of DAC '01*, 2001.
[23] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM TACO*, 2004.