PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs

Kai Ma and Xiaorui Wang
Department of Electrical and Computer Engineering
The Ohio State University
Columbus, OH 43210
{mak, xwang}@ece.osu.edu

ABSTRACT

Optimizing the performance of a chip multiprocessor (CMP) within a power cap has recently received a lot of attention. However, most existing solutions rely solely on DVFS, which is anticipated to have only limited actuation ranges in the future. Power gating shuts down idling cores in a CMP, such that more power can be shifted to the cores that run applications for better CMP performance. However, current preliminary studies on integrating the two knobs focus on deciding the power gating and DVFS levels in a tightly coupled fashion, with much less attention given to the direction of decoupled designs. By decoupling the two knobs that may interfere with each other, individual knob management algorithms can be less complex and more efficient to take advantage of the characteristics of different knobs. This paper proposes *PGCapping*, a decoupled design to integrate power gating with DVFS for CMP power capping. To fully utilize the power headroom that is reserved through power gating, *PGCapping* enables per-core overclocking on turned-on cores that run sequential applications. However, per-core overclocking may make some cores age much faster than others and thus become the reliability bottleneck in the whole system. Therefore, *PGCapping* also uses power gating to balance the core lifetimes. Our empirical results on a hardware testbed show that the proposed scheme achieves up to 42.0% better average application performance than five state-of-the-art power capping baselines for realistic multi-core applications, i.e., a mixed group of PARSEC and SPEC CPU2006 benchmarks. Furthermore, our extensive simulation results with real-world traces demonstrate that a lightweight lifetime balancing algorithm (based on power gating) can increase the CMP lifetime by 9.2% on average.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies; C.1.4 [**Processor Architectures**]: Parallel architectures

General Terms

Design, Management, Performance, Experimentation

Keywords

Chip multiprocessor, power capping, power control, power gating, lifetime balancing, control theory

1. INTRODUCTION

Power has become a first-class constraint in current microprocessor design due to packaging, cooling, and power delivery circuit limits. An important research challenge is to optimize the performance of a Chip Multiprocessor (CMP) within a given power constraint (i.e., power capping). Recently, many research studies have been conducted to utilize Dynamic Voltage and Frequency Scaling (DVFS) as a knob for power capping. Unfortunately, in recent generations of technology scaling, to keep leakage current under control, the decrease in the threshold voltage (V_{th}) of transistors has stopped [13]. This, in turn, has prevented the supply voltage (V_{dd}) from further decreasing. As a result, DVFS alone may no longer be able to fully address the power capping issue. Power gating is a technique that cuts off the power supply of a logic block by inserting a gate (or sleep transistor) in a series with the power supply [27]. Gating the power supply results in almost no power consumption in the gated block [15]. Power gating complements DVFS by providing an effective mechanism to reduce leakage power. Therefore, it is preferable to integrate power gating and DVFS in power capping for further improved CMP performance.

In this paper, we consider the case of per-core power gating (PCPG) because it has been implemented in mainstream processors (e.g., Intel Core i7 [15]). PCPG and DVFS have different characteristics in terms of their transition overheads and interactions with OS, which requires a decoupled design to address their differences. PCPG has a larger transition time and energy overhead [27, 23]. Furthermore, since PCPG changes the number of turned-on cores, the OS scheduler may reallocate the thread-core mapping in each power gating interval. Therefore, the algorithm designed for PCPG should track long-term trends and avoid actuation oscillations. In contrast, DVFS has a smaller transition overhead (e.g., $10\mu s$ in Nehalem processors [15]). Moreover, it does not change the on/off states of the cores. Therefore, DVFS is preferable to explore short-term workload variations (e.g., multiple DVFS adjustment intervals within one scheduling interval) [37, 28]. However, existing efforts [21, 3] on integrating power gating and DVFS for power capping simply treat the power gating state as an extra-low power state below the existing DVFS levels and consider power gating and DVFS state in a coupled fashion at a coarse time scale. These coupled designs cannot either take advantage of fine-grain DVFS or avoid unnecessary actuations for power gating. Furthermore, these coupled designs usually require manually disabling the OS scheduler to have a fixed thread-core mapping, because if OS changes the thread-core mapping, the current statistics of one core cannot be used to decide the power state for the next interval. Therefore, a decoupled design that can meet the different requirements of power gating and DVFS, as well as can be deployed with native OS, needs to be developed.

Through PCPG, the wasted leakage power of certain underutilized cores (in DVFS alone systems) can be proactively transformed into the dynamic power headroom for accelerating useful applications. Hardware overclocking provides CMPs with the capability of fully utilizing the dynamic power headroom for optimized performance. However, many existing CMPs have only homogeneous cores with *chip-level* overclocking capability (e.g., Intel TurboBoost [15]), which cannot fully explore the variations of different applications among cores during runtime. Since the benefit of per-core DVFS has been discussed in detail [20], we consider the case that with the per-core overclocking enabled, CMPs with homogeneous cores can mimic the functionality of heterogeneous cores to dynamically provide more powerful cores to meet the runtime requirement of applications. Even in the systems without physically implemented per-core DVFS (e.g., multi-power-island chips), Rangan et al. [34] have shown that thread migration on systems with only two power states can be used to approximate the functionality of per-core DVFS. Compared with TurboBoost, which only adjusts the *chip-level* DVFS levels, this paper addresses a different issue of coordinating the DVFS (overclocking) states for multiple on-chip cores to improve the CMP performance within a chip-level power cap.

While per-core DVFS and overclocking offer new opportunities to explore the power-performance trade-off, they also pose serious challenges to CMP reliability. Overclocking directly leads to a higher wear-out rate on the overclocked cores [19]. The practice of employing per-core DVFS/overclocking aggravates the case that some cores may age much faster than others and become the reliability bottleneck for the whole system, which thus significantly reduces the system service life [14]. Previous studies [9] have developed effective algorithms to use per-core DVFS to balance the service lifetime of on-chip cores. However, the solution of lifetime-balancing algorithms may conflict with the solution of power-capping algorithms. For example, lifetime-balancing per-core DVFS algorithms may throttle the cores running high-activity applications (e.g., high IPC) to reduce wear-out. However, those cores are probably the same cores that performance-power optimization algorithms usually seek to boost performance. It is unlikely to be able to coordinate those conflicting goals by using just one knob. Fortunately, power gating offers new opportunities to manage the core lifetime balancing issue, because the power-gated cores do not experience measurable wear-out [19]. Therefore, it is possible to achieve lifetime balancing through modulating the power gating state of each core.

Base on the above observations, this paper proposes *PGCapping* (Power Gating for Capping), a decoupled design to integrate power gating with per-core DVFS/overclocking for CMP power capping and also discuss how to use power gating to balance the core lifetimes. Specifically, *PGCapping* consists of a Proportional-Integral (PI) controller based on feedback control theory to manage power gating and a *Quicksearch* algorithm for DVFS/overclocking management. Both the PI controller and *Quicksearch* algorithm are invoked periodically. We select different intervals for the PI controller and *Quicksearch* to decouple them. The PI controller adjusts the number of turned-on cores to control the chip power at a coarse time scale with theoretically provable stability guarantee. At a finer time scale, *Quicksearch* employs per-core DVFS to fully handle the short-term workload variations. Core-level lifetime balancing is achieved by selecting which core(s) to be turned on/off after the power controller decides the number of turned-on cores.

Specifically, this paper makes the following major contributions:

- We propose a novel algorithm *PGCapping*, which integrates power gating, DVFS, and core overclocking to optimize the CMP performance within a power cap. *PGCapping* explores a novel decoupled design direction, which has not been addressed sufficiently by previous studies. PGCapping conducts power gating at a coarser time scale for reduced runtime overhead and DVFS at a finer time scale to handle short-term workload variations.

- Since overclocking may have negative impacts on the core aging rates and lead to unnecessarily shortened CMP lifetime, *PGCapping* integrates core lifetime balancing as an integral part of the proposed power capping framework and uses a power-gating-based balancing algorithm to maximize the CMP lifetime.

- While most existing work has only simulation results, we implement the proposed *PGCapping* solution on a 12-core AMD Opteron processor and present empirical results. Our results show that our decoupled design achieves up to 42.0% better average application performance than five state-of-the-art baselines for mixed PARSEC and SPEC CPU 2006 benchmarks.

The rest of this paper is organized as follows. Section 2 highlights the differences between this paper and related work. Section 3 describes the decoupled design. Section 4 introduces our hardware testbed, simulation setups, and the implementation details of our solutions. Section 5 presents our baselines and evaluation results. Section 6 concludes this paper.

2. RELATED WORK

A well-known industry practice to integrate DVFS, overclocking and power gating is the Intel TurboBoost technology [15]. However, TurboBoost, as well as the studies in Li et al. [25] and Lee et al. [21], discusses chip-wide DVFS rather than per-core DVFS, which limits the optimization space. Because chip-wide DVFS *cannot* fully explore the inter-core variations. Since per-core frequency scaling has been available in commercial processors (e.g., AMD [1]) and on-chip regulators has been proposed [20] for per-core DVFS, this paper focuses on integrating power gating and per-core DVFS. Compared with power gating and chip-level DVFS, this paper addresses a different but more challenging problem because we need to coordinate the power state (i.e.,

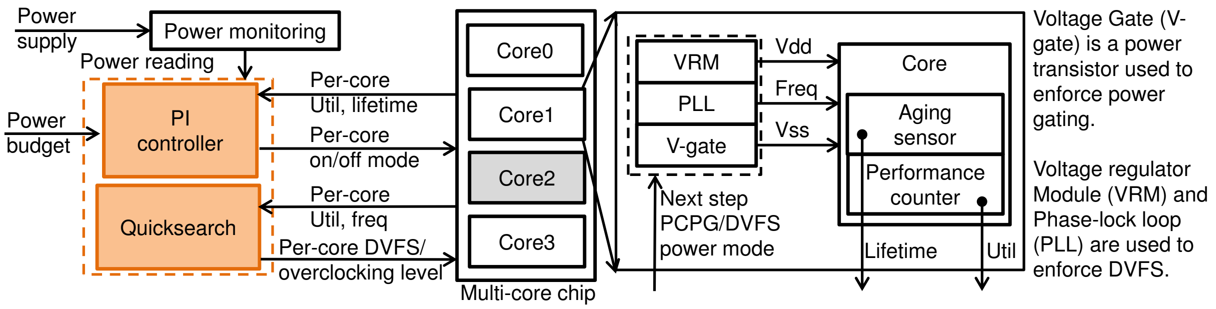


Figure 1: *PGCapping* features a decoupled design that takes the power budget, chip power measurement, per-core utilization, temperature, lifetime as inputs. *PGCapping* computes next-step power mode (e.g., on/off, DVFS levels, overclocking state) of each core to cap the entire chip power, boost performance, and balance core lifetimes.

power gating state and DVFS level) of each core for optimized performance. Many power-capping solutions [37, 28] based on the per-core DVFS have been proposed. However, they do not discuss the coordination with the power gating technology. Bircher et al. [3] use both the per-core DVFS and power gating. However, they focus on proposing and validating a workload phase predictor for the power management. The impact of the real-world OS scheduling, which is crucial to the workload-based power management, has not been sufficiently addressed in their work.

Karpuzcu et al. [19] have proposed to use per-core overclocking to boost sequential application execution. However, they only overclock one core each time until it runs out and then they cut that core off from the on-chip logic and power network. In contrast, our scheme allows multiple cores to simultaneously stay at the overclocking state within a desired power budget and we balance the service lifetime of each core.

Power capping has been previously addressed at different levels, such as CMP [40, 16, 3], server [22, 8], server cluster [38, 33], and data center [39]. However, all the prior studies rely only on DVFS without core lifetime balancing.

The core-level lifetime balancing issue of multi-core processors has been extensively studied. However, previous explorations mainly focus on the per-core DVFS and scheduling [14, 9]. Using per-core power gating as a reliability management knob and the impact of overclocking are rarely touched. Compared with previous art, this paper discusses using power gating as a reliability management knob and specifically considers the overclocking scenarios.

3. SYSTEM DESIGN

In this section, we introduce *PGCapping*, which integrates power gating, DVFS, and overclocking to optimize the CMP performance within a power cap. *PGCapping* is a novel decoupled design that conducts power gating at a coarse time scale for reduced runtime overhead and DVFS at a fine time scale to handle short-term workload variations.

As shown in Figure 1, we design one power gating management module (i.e., *PI controller*) and one DVFS/overclocking management module (i.e., *Quicksearch* algorithm) to conduct power capping at different time scales. *PI controller* adjusts the number of turned-on cores to control the chip power based on the power measurement and budget at a coarse time interval. Core-level lifetime balancing is achieved through assigning the on/off state of each core based on its utilization. At a fine-gained time interval, the power control and performance optimization are realized by adjusting the DVFS level of each core. We treat the overclocking states as extra DVFS levels higher than the labeled peak DVFS levels. Therefore, the assignment of overclocking state is also conducted in *Quicksearch*.

Both *PI controller* and *Quicksearch* are invoked periodically. We select different intervals for *PI controller* and *Quicksearch* to decouple them. The interval of *PI controller* is set to be much longer than the OS scheduling interval to give OS enough time for spreading and balancing workload among turned-on cores. The interval of *Quicksearch* is selected to be much shorter than that of OS scheduling. Therefore, before the next *PI controller* interval starts, *Quicksearch* has already settled at the optimal point with the current power gating setting. As a result, the *PI controller* does not introduce oscillations to the DVFS management. On the other side, since *Quicksearch* converges quickly, the impacts of DVFS on PCPG are observed to be negligible. Therefore, the two loops will not interfere with each other and can be designed independently. Moreover, both loops are decoupled from OS scheduling, so native OS scheduling without modification can be used in this decoupled design.

3.1 Design of PCPG Management Module

In this section, we introduce the design of *PI controller* that controls the power consumption of the entire chip to a desired power budget by adjusting the number of turned-on cores at a coarse time interval.

The power of the chip generally has a monotonic relationship with the number of turned-on cores. However, the runtime variations (e.g., different applications or DVFS) make it unlikely to develop an accurate model for all possible cases. Therefore, we adopt feedback control theory to design a PI controller to decide the number of turned-on cores. A key advantage of the control theory design approach is that it can tolerate a certain degree of modeling errors and adapt to online model variations based on dynamic feedback [10]. Therefore, our solution does not rely on power models that are perfectly accurate, which is in sharp contrast to open-loop solutions that would fail without an accurate model.

Controller Design. Following standard PI control theory design procedures [10], our controller is designed as:

$$N(k) = N(k-1) + \frac{(P_t - cp(k))}{a}. \tag{1}$$

$N(k)$ is the number of turned-on cores on the chip in the k^{th} control period. P_t is the power budget of the entire chip, which can be determined by the thermal and power supply constraints of the processor or specified by the user during runtime. $cp(k)$ is the power consumption of the entire

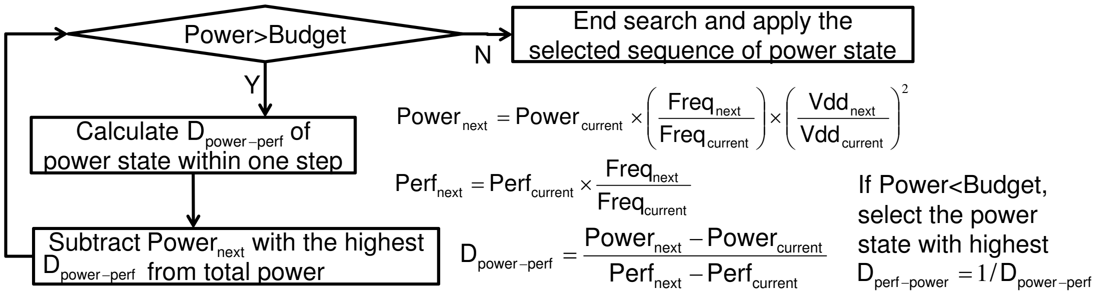

Power>Budget
N
End search and apply the selected sequence of power state
Y
Calculate $D_{power-perf}$ of power state within one step
Subtract $Power_{next}$ with the highest $D_{power-perf}$ from total power
$Power_{next} = Power_{current} \times \left(\frac{Freq_{next}}{Freq_{current}}\right) \times \left(\frac{Vdd_{next}}{Vdd_{current}}\right)^2$
$Perf_{next} = Perf_{current} \times \frac{Freq_{next}}{Freq_{current}}$
$D_{power-perf} = \frac{Power_{next} - Power_{current}}{Perf_{next} - Perf_{current}}$
If Power<Budget, select the power state with highest $D_{perf-power} = 1/D_{power-perf}$

Figure 2: *Quicksearch* algorithm flowchart. Only power-higher-than-budget case is presented for concision.

chip in the k^{th} control period. Parameter a characterizes the power consumption of one core, which may vary for different chips and applications. In our design, we derive a by using the datasheet full power range (from the idle power to the maximum power of the chip [2]) divided by the dynamic range of $N(k)$ (e.g., a=5.9).

Controller Analysis. A fundamental benefit of the control-theoretic approach is that it offers a mathematical framework to analyze the system stability and performance, even when the system power model may change at runtime due to variations. By applying pole analysis [10] on our controller, we prove that the closed-loop system is stable as long as $0 \leq a \leq 11.8$. In our experiments, the variation never exceeds the range. Please note in our PI controller design, we only use three datasheet numbers (idle power, TDP, the number of cores) without the need to profiling the power efficiency of each core. This design methodology is in sharp contrast with previous studies, which typically requires extensive profiling tests.

Actuation refinement. Through the controller calculation, we have derived the number of cores N_a that should be at the turned-on state in order to optimize the CMP performance within the desired power cap. Next, we get the number of unfinished tasks N_b from OS. We then enforce $min(N_a, N_b)$ cores at turned-on state for the next interval to transform saved leakage power to the dynamic power headroom that can be used to improve the CMP performance.

3.2 Design of DVFS Management Module

Since the power gating state of each core has been decided by the PI controller, we now introduce *Quicksearch* algorithm to explore short-term workload variations with per-core DVFS/overclocking.

Quicksearch uses the performance/power ratio to decide the power state (i.e., the DVFS/overclocking level) of each core. Specifically, we define $Perf{=}Util * Freq$ as our performance metric. As discussed in [19], the aggregated frequency $Freq$ is a high-level computational capacity metric. We extend their metric by weighting frequency with utilization $Util$ to deduct the idling cycles. As shown in Figure 2, *Quicksearch* starts with the current power states of all the cores. If the current power is higher than the chip-wide budget, the algorithm selects the application/core pair that would provide the highest $D_{power-perf}$ (i.e., power reduction to performance loss ratio). The chip-level power consumption with this new configuration is estimated by adding $Power_{next}$-$Power_{current}$ to the current chip power. If the new estimated chip power is still higher than the budget, *Quicksearch* is again invoked from the new configuration. This process is repeated until the power budget is met. If the current power is lower than the chip-wide budget, the algorithm selects the application/core pair that provides the highest ratio of performance gain to power increase $D_{perf-power}$. The chip power consumption of this new configuration is estimated by adding $Power_{next}$-$Power_{current}$ to the current chip power. If the new estimated chip power is still lower than the budget, *Quicksearch* is again invoked from the new configuration. This process is repeated until the power budget is met. *Quicksearch* extends the *SteepestDrop* algorithm [41] to consider overclocking states. In an overclocking-enabled system, the final peak DVFS level might not be fixed [1]. Starting from the peak configuration (as in original *SteepestDrop*) is not always possible. Therefore, *Quicksearch* uses the current configuration as the search starting point. Moreover, there are relatively fewer power mode transitions in the stable execution phases. Therefore, *Quicksearch* reduces the search time. As a greedy algorithm, *Quicksearch* cannot completely avoid ending at a local optimum. Fortunately, DVFS impacts the performance approximately in a linear way unless the application is extremely memory-bounded, which is not a common case.

Overclocking and parallel applications. In *Quicksearch*, we do not explicitly discuss the core acceleration because we consider the overclocking states as extra DVFS levels above the labeled peak DVFS level. For parallel applications, hardware or software spin loop detectors [26] can be used to identify the active waiting loops within applications to deduct spinning cycles from working cycles. In that case, the utilization-weighted frequency represents the amount of real work that a core is doing. Selecting the steps to maximize the performance/power ratio in *Quicksearch* is still valid. We can even further improve the parallel application performance by on-line detecting the critical thread and overclocking the critical thread [28]. Please note we address the core-to-core power efficiency variations by using per-core DVFS (or per-core frequency scaling) in *Quicksearch. Quicksearch* first dynamically estimates the perf/power ratio of each core; then it raises the frequency of a high perf/power ratio core if the current power is lower than the budget; if the current power is higher than the budget, it lowers the frequency of a low perf/power ratio core.

3.3 Lifetime Balancing

In this section, we introduce the core lifetime balancing algorithm in *PGCapping*.

We propose to organize all the cores to two lists. One list contains all the turned-on cores. The other list contains all the turned-off cores. When we need to turn on one core, we find the core with the longest estimated lifetime in the turned-off core list to turn on. When we turn off one core, we find the core with the shortest estimated lifetime in the turned-on core list to turn off. Wear-out information can be estimated by using aging sensors that dynamically measure the increase in critical path delays due to aging [5]. Please note lifetime balancing only takes place when some cores are turned on/off. Therefore, this balancing algorithm does not interfere with the power-capping algorithm introduced earlier. Our experiments have shown that this balancing strategy can achieve decent core-level lifetime balance results (in Section 5.4). Compared with previous sophisticated designs (e.g., [9]), which usually require on-line workload characteristic evaluation and then use scheduling or per-core DVFS to balance the lifetime of each core, the algorithm we present does not require any modification about OS scheduling or DVFS management. There are mainly two reasons why such a lightweight algorithm can work well: 1) in real production environment, there are always variations (e.g., processor utilization and power budget variations) for us to turn on/off cores; 2) the normal wear-out process takes place gradually since we eliminate catastrophic overheating through power capping.

4. IMPLEMENTATION

In this section, we introduce the physical testbed for power capping evaluation and simulation environment for lifetime balancing evaluation.

4.1 Power Capping Evaluation Testbed

Our testbed is a 12-core AMD Opteron 6168 processor running OpenSUSE 11.3. 6168 does not support per-core DVFS and PCPG [2]. We use per-core p-state assignment to emulate per-core DVFS. 6168 has 5 p-states (0.8GHz/-0.975V, 1.0GHz/1.0V, 1.3GHz/1.025V, 1.5GHz/1.0625V, and 1.9GHz /1.1125V). When we assign a p-state to one core, the frequency of that core is enforced independently as the defined value. However, the core voltage is decided by the highest frequency among the cores because all the cores share the same voltage plane [2]. We use CPU Hotplug [31] to emulate PCPG as in [23]. CPU Hotplug was originally intended for systems with hardware support to install and remove CPU modules without interruption, which mimics precisely the behavior necessary to model per-core power gating. The per-core p-state assignment and CPU Hotplug can emulate the performance impact of per-core DVFS and PCPG.

A challenge we face is to estimate the power consumption because the direct measured CPU power does not factor in the power impact of core-grained voltage scaling and PCPG. Our hybrid power estimation (based on real-time measurement), which considers the power impact of per-core DVFS and PCPG, works as follows. First we measure the power consumption of 6168 as in [17]. An Agilent 34410A digital multimeter is used together with a Fluke i410 current probe to measure the current running through the 12V power lines that power the processor. The accuracy of this measurement is ±3.5% of reading. We first set all the cores to the same p-state and measure the idling power P_{idle}. Our measure is: 44.0W(0.8GHz), 46.4W(1.0GHz), 51.3W(1.3GHz), 55.1W(1.5GHz), 61.3W-(1.9GHz). Then we calculate the activity factor β in model (2) [18, 28].

$$mPow = \sum_{j=0}^{N-1} \beta * \frac{Freq(j)}{1.9} * (\frac{V_{DD}}{1.1125})^2 * Util(j) + P_{idle} \quad (2)$$

where $mPow$ is the *measured* power of the entire chip. Parameter $Freq(j)$ is the frequency of the jth core. V_{DD} is the Vdd corresponding to the highest frequency among all the cores because all the cores share one voltage plane. $Util(j)$ is the utilization of the jth core. $mPow$ is measured. P_{idle} at different chip-level p-states (decided by the peak frequency among the cores) has been measured. $Freq(j)$ and $Util(j)$ can be derived from OS. By using those inputs, we can compute the chip-level activity factor β. Using chip-level activity factor is a trade-off between estimation accuracy and estimation complexity [18, 28]. In our extensive validation experiments, the average error is 1.1W (the worst case is 3.6W) between the estimated and measured power for this estimation methodology. Once we derive β, we estimate the power of each core as:

$$Pow(j) = \beta * \frac{Freq(j)}{1.9} * (\frac{V_{DD}(j)}{1.1125})^2 * Util(j) + \frac{P_{idle}}{12} \quad (3)$$

where $Pow(j)$ is the estimated power of the jth core. Please note $V_{DD}(j)$ is the voltage of the jth core at current DVFS level. The chip power is estimated as $Pow=\sum_{j=0}^{N-1} Pow(j)$. We assume $Pow(j)$=0 if it is power-gated. We report Pow in our experiment part because it accounts for the power impact of per-core DVFS and PCPG. We do not explicitly consider the dynamic power of the uncore part because the uncore power is actually driven by the core part. We attribute uncore part power to the corresponding core part, which is has been shown to be sufficiently accurate for the power management purpose [18, 28].

Table 1: Workload mixes used on physical testbed.

Mixes	PARSEC 2.1, SPEC2006 [1]	Aggregate Effect
mix1	12-mcf	memory intensive
mix2	12-perlbench	CPU intensive
mix3	8-swaptions, 4-omnetpp	no-barrier parallel
mix4	4-(blackscholes, bodytrack) 2-(xalancbmk, povray)	low-barrier and high-barrier parallel
mix5	4-x264, 8-fluidanimate	high-lock parallel
mix6	4-(vips,facesim) 1-(libq,astar,soplex,dealII)	random mix

We evaluate the proposed solution with the PARSEC and SPEC workload mixes (Table 1), which cover a variety of different aggregate effects [28].

On our prototype testbed, we implement the control algorithm as an OS daemon process to control the target processor. The intervals of our designs are decided as follows. First, our physical testbed measurement shows that the overhead of DVFS is $35\mu s$ (close to $10\mu s$ in Nehalem [15]) and the average overhead of PCPG is 127ms and the worst case is 220ms (close to the 100ms measurement result in [23]). Second, we study the key intervals in the OS. Although the minimum Linux time quantum can reach 10ms, the default time slice is 100ms [6]. The default load balancing interval is 200ms. Third, we decide our design intervals. In the decoupled design, we select 2s as PCPG interval to give the kernel sufficient time to balance the workloads in the decoupled design. We select DVFS interval to be 50ms, which is smaller than the default time slice to test fine-grain DVFS.

4.2 Lifetime Balancing Evaluation Simulator

6168 does not have aging sensors. Therefore, we evaluate the lifetime balancing part in a simulator [29]. We model the core of Opteron 6168 (e.g., voltage, DVFS levels, frequency). In each time interval (e.g., DVFS interval), per-core power is modeled as in [21] with two parts: the dynamic power is based on utilization and DVFS level, the leakage power is based on temperature. The temperature is modeled

[1]The number-appname notation is the number of threads of the application with the name of appname for PARSEC; for SPEC2006 workload, it is the number of copies of the application with the name of appname.

as in [19] based on the total power. Since the total power impacts temperature and the temperature impacts leakage power. These two parts are entangled together. We run multiple iterations of calculation until both power and temperature readings converge (e.g., no more than 5% change between two consecutive iterations). We use RAMP2.0 to model the service lifetime. The different failure models and key parameters are the same as those used in [35].

4.3 Discussion on Per-core DVFS

We assume per-core DVFS as one possible power adjustment knob (which is also assumed in [20, 21, 24, 41]) and emulate per-core DVFS with per-core frequency scaling in our testbed. However, *PGCapping* does not rely on per-core DVFS. In fact, *PGCapping* already works with pre-core frequency scaling on our testbed. To physically enable fast voltage adjustment required by per-core DVFS, on-chip voltage regulators are needed [20]. Recently, IBM Watson Lab demonstrated a 2.5D on-chip integrated regulator [36], showing the new manufacturing break-through in enabling fine-grain DVFS. Moreover, even in the systems without physically implemented per-core DVFS, Rangan et al. [34] have shown that thread migration on systems with only two power states can be used to approximate the functionality of per-core DVFS. Therefore, this paper emulates per-core DVFS in our evaluation and explores the integration of per-core DVFS and power gating.

5. EVALUATION

In this section, we compare the power control accuracy, application performance, core-level lifetime balancing results of *PGCapping* and baselines in both physical testbed and simulator.

5.1 Baselines

In this section, we introduce the five studied policies.

Our first baseline is *per-core-DVFS-only*. It adopts the original *SteepestDrop* algorithm [41], without PCPG or overclocking states. We select *SteepestDrop* as an example of state-of-the-art per-core DVFS based solutions. However, the dynamic range of *per-core-DVFS-only* is limited due to the lack of overclocking states and PCPG.

Our second baseline is *PCPG-only*. We disable DVFS and use PCPG as the only knob to control power. When the current power $cp(k)$ is lower than the budget P_t, we need to turn on cores. In order to achieve a fast convergence, we turn on multiple cores each time. We randomly pick a core in the turned-off core list, then we use the power when it was turned off as an estimation and add that power to $cp(k)$. If $cp(k)$ is still lower than P_t, we pick another core and so on, until $cp(k)$ is higher than P_t. When the current power $cp(k)$ is higher than the budget P_t, we do just the opposite. We allow turn on/off multiple cores each time to achieve fast convergence because PCPG takes place at a coarse time interval. We design *PCPG-only* as an example of using PCPG as the only knob to control the chip-level power consumption in a heuristic way. However, this simple heuristic cannot avoid the oscillation.

Our third baseline *ETurboBoost* is an extended version of Intel TurboBoost [15]. It uses chip-level DVFS and PCPG. Original TurboBoost does not turn off cores when all the cores are running applications and adjusts the DVFS level of the entire chip to fit the power budget by one level at each DVFS interval. *ETurboBoost* extends original TurboBoost by using the PI controller to control the number of cores when the assigned power budget is lower than the power consumption that all the cores are running at the lowest frequency levels. We implement *ETurboBoost* as an example of state-of-the-art power capping solution through power gating and chip-level DVFS.

Our fourth baseline *Per-core-DVFS+PCPG* is using PI controller for PCPG and per-core DVFS with *Quicksearch* but without overclocking states. We select this baseline to show the necessity to include overclocking states to fully explore the power headroom.

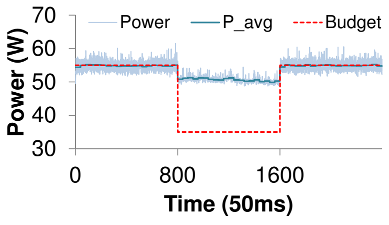
We design our fifth baseline *EQuicksearch* (i.e., Enhanced Quicksearch) to be a very competitive coupled solution to integrate per-core DVFS/overclocking with power gating. *EQuicksearch* extends power gating state to *Quicksearch*. We take the power gating state as the extra-low power state under the labeled lowest DVFS level as in previous coupled design [21]. We assume the power and performance of a turned-off core to be zero. We estimate $Power_{next}$ and $Perf_{next}$ of a currently turned-off core as the average power and performance of all the turned-on cores at the last interval. We also add the transition power to $Power_{next}$ and deduct the transition time from $Perf_{next}$ to minimize the undesirable oscillation. We take *EQuicksearch* as an example of coupled solutions because it demonstrates two key features of a coupled solution: 1) the coupled solution has the opportunity to select from the entire available adjustment space; 2) due to the huge search space, pruning strategies (e.g., greedy algorithm like *SteepestDrop*) usually need to be employed. We consider *EQuicksearch* as a state-of-the-art coupled design because its predecessor *SteepestDrop* has been shown to outperform many other recently published solutions [30, 37] and intuitive baselines [41]. Since *EQuicksearch* may change the power gating states of multiple cores, we set the interval to be sufficiently long to conduct the required actuations. In our experiment, we invoke *EQuicksearch* every 1s to get the trade-off between actuation overhead and system responsiveness. Please note coupled solutions (either *EQuicksearch* or previous studies like [21]) may change the on/off states of cores at each control interval. As a result, OS might reschedule the thread-core mapping, which makes the last interval statistic based decision-making invalid. Therefore, we have to manually assign thread-core affinity when we use coupled solutions. In contrast, our proposed decoupled design can work with native OS without any manual intervention because the PCPG and DVFS parts have been decoupled with OS through interval selection.

5.2 Power Control

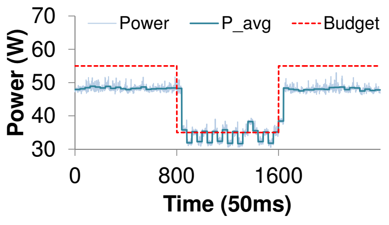
In this section, we present the physical testbed results. We set 1.3GHz as the normal peak working state. We have three DVFS states (0.8GHz, 1.0GHz, 1.3GHz) and two overclocking states (1.5GHz, 1.9GHz). The overclocking setting agrees with the overclocking potential analysis in [12] (e.g., normally 10-40%).

In Figure 3, we run 7 copies of a randomly selected benchmark *gobmk* from SPEC CPU2006 on Opteron 6168. We reduce the power budget at 40s from 55W to 35W to emulate a power budget cut scenario due to various reasons (e.g., thermal emergency). The power budget is then raised back to 55W at 80s after the emergency is resolved. Initially, we turn on all cores at the lowest DVFS level as in Linux default case.

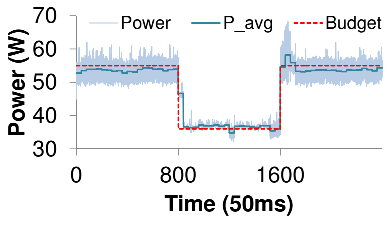
Figure 3a shows the results of *per-core-DVFS-only*. Be-



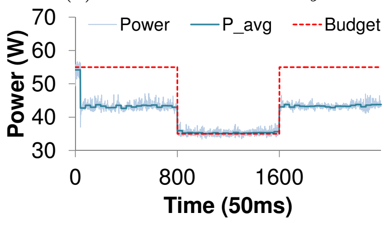
(a) *Per-core-DVFS-only.*



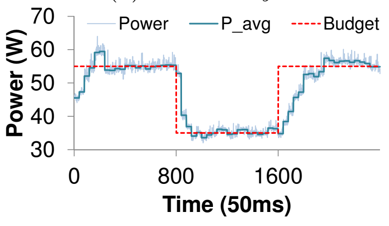
(b) *PCPG-only.*



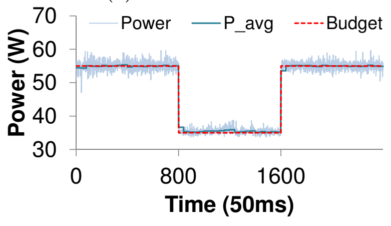
(c) *ETurboBoost.*



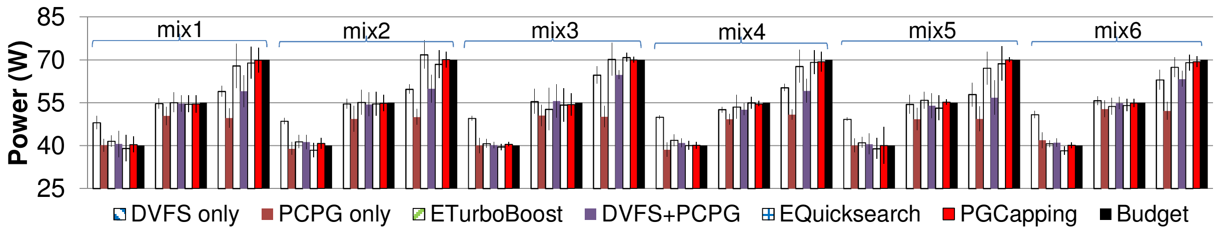
(d) *Per-core-DVFS+PCPG.*



(e) *EQuicksearch.*



(f) *PGCapping.*



(g) Power comparison with different budgets and different benchmarks.

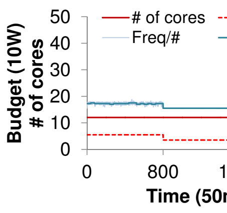
Figure 3: Hardware testbed results to show that *PGCapping* can precisely enforce the power budget by using PCPG, DVFS and overclocking in both the high and low power budget cases. We calculate the average power with a 2s window as P_avg to clearly present the general trend.

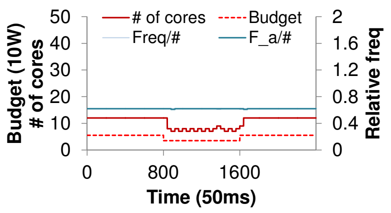
cause the frequencies of all the cores have been set to the lowest level, *per-core-DVFS-only* cannot further reduce power to enforce power budget cut at 40-80s. This might introduce system crash or other undesirable failures. In Figure 3b, with PCPG, the system can successfully reach the low power budget. However, the frequently turning on/off cores introduces large power oscillations and potential large thermal cycling. Figure 3c shows the results of *ETurboBoost*. *ETurboBoost* can enforce the desired power budget. However, *ETurboBoost* uses chip-level DVFS and introduces large power oscillations. More importantly, *ETurboBoost* always oscillates between two adjacent DVFS levels of the entire chip, even in the steady state. As a result, even on average, it never settles to the set point. Please note there is averagely an 2.5W power gap between the 2s-window average power and the budget curve during 0-40s and 40-80s. Potential performance benefits could be gained by fully utilizing the power budget. Moreover, the processor is running on average 0.7W higher than the budget during 40-80s, which may possibly introduce undesirable failures. In Figure 3d, we show the case of per-core DVFS with PCPG but no overclocking. Our PI controller with refinement (Section 3.1) can identify that there are only 7 cores are used in our experiment and turn off the rest 5 cores. However, without overclocking, even with all the cores set to the peak DVFS level, the power budget still cannot be fully utilized. Figure 3e shows that *EQuicksearch* can enforce the power budget. However, *EQuicksearch* has larger oscillations due to coarser time interval. Figure 3f shows that *PGCapping* can precisely enforce the power budget. In steady state, *PGCapping* oscillates between two adjacent DVFS levels of one core, which is much smaller than oscillating between two adjacent DVFS levels of the entire chip in *ETurboBoost*. As more and more cores are expected to be integrated on-chip with future process technology scaling, the benefit of one core adaptation instead of whole chip adaptation will become more significant. We also notice although *PGCapping* tracks the power budget closely in Figure 3f, it exceeds the budget sometimes for very short time periods. We believe those small and short noise will not introduce serious issues because modern computer systems are extensively equipped with capacitors in the system to address potential voltage or power variations. Moreover, the power overdraw protection is usually implemented by circuit breakers, which are usually designed to tolerate short period of time of overdraw (typically seconds [11]). Therefore, an extremely short period power overdraw can be masked by those factors. In our physical testbed results, the power overdraw is at the order of milliseconds and the average power is strictly controlled to the set point.

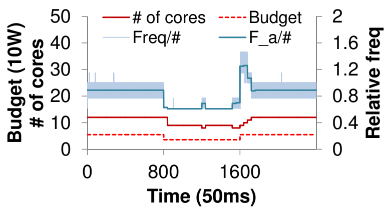
Figure 3g shows that *per-core-DVFS-only*, *PCPG-only*, *ETurboBoost*, and *Per-core-DVFS+PCPG* cannot adjust the processor to the desired power budget in certain cases for the reasons analyzed above. In contrast, *PGCapping* and *EQuicksearch* can always control the power to the assigned setting point for different benchmark mixes with different power budgets.

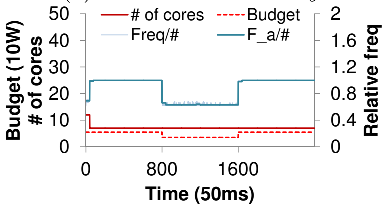
5.3 Application Performance

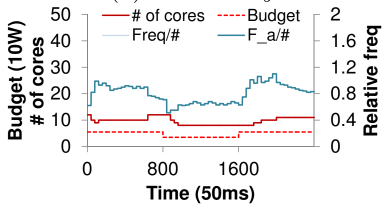
In Figure 4, we present the corresponding frequency, number of turned-on cores traces of Figure 3 to show the performance impact of different policies. The frequencies of the turned-off cores are counted as 0. In Figure 4a, *per-core-DVFS-only* policy has no PCPG. Therefore, during 40-80s, the frequencies of all the cores have been set to the low-

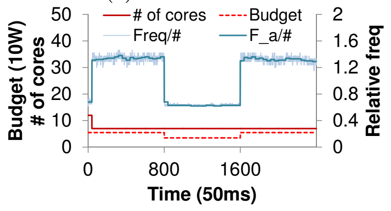

(a) *Per-core-DVFS-only.*

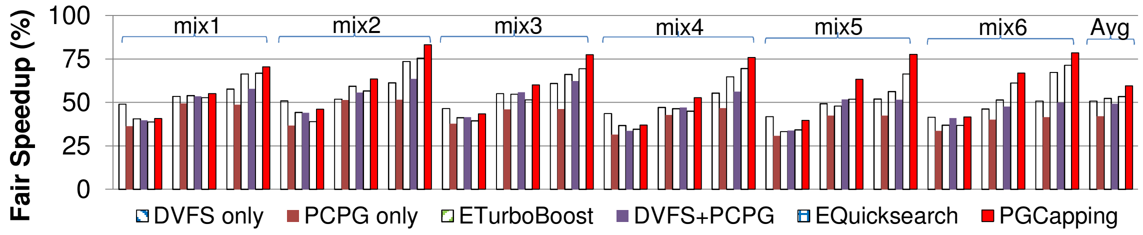

(b) *PCPG-only.*


(c) *ETurboBoost.*


(d) *Per-core-DVFS+PCPG.*


(e) *EQuicksearch.*


(f) *PGCapping.*



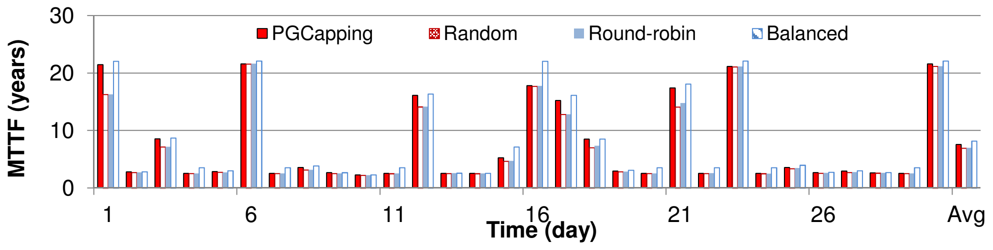
(g) Performance comparison with different budgets and different benchmarks.

Figure 4: Hardware testbed results to show that *PGCapping* can reserve power headroom by using PCPG and accelerate cores running useful workload by using overclocking. The frequencies are normalized to the peak frequency of one core (i.e., Relative freq). The Freq/# is calculated by dividing the total aggregated relative frequency of the entire chip by the number of turned-on cores, which can be interpreted as a high-level computing capability that each turned-on core can offer (the higher is preferable). We also calculate average Freq/# with a 2s window as F_a/#.

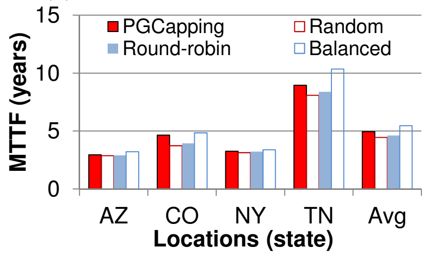
est level continuously and could not go lower to enforce desired power budget. Figure 4b shows the results of *PCPG-only*. Because *PCPG-only* does not change the DVFS levels of the cores, it only adjusts the number of turned-on cores to fit the power budget. We can observe that the cores are always turned on/off frequently. This is not desirable because PCPG has a high overhead compared with DVFS. Frequently turning on/off cores negatively impacts performance. In Figure 4c, *ETurboBoost* adjusts the DVFS level of the entire chip up/down around the power budget, which could not explore the inter-core variations. Moreover, due to the large granularity of chip-level DVFS, DVFS and PCPG have large interference at transition time, leading to the large overshoot at 80s. Figure 4d shows the results of per-core DVFS with PCPG but no overclocking. We can see from 0-40s and 40-80s, when the power budget is high, due to the lack of overclocking state, the DVFS levels of the cores could not be raised further, which fails to achieve optimal performance. In Figure 4e, *EQuicksearch* purely relies on a heuristic algorithm to decide the power state of each core in a coupled fashion (e.g., using DVFS or on/off state to adjust power consumption). Therefore, the adaptation is limited to a coarse level. Furthermore, *EQuicksearch* cannot avoid unnecessary oscillation of power state due to the system noises and workload variations. Please note we already minimize the power gating oscillation by considering the transition power and time overhead in *EQuicksearch* algorithm. In Figure 4f, *PGCapping* can identify the idling cores and turn them off to reserve the power budget headroom. It then uses overclocking to fully utilize the headroom. Please note that a core is running at a higher DVFS level on average when managed by *PGCapping* than *ETurboBoost* and *EQuicksearch* because *PGCapping* can fully utilize the power budget. In addition, *PGCapping* can fully explore the runtime variations among different cores by using per-core DVFS and per-core overclocking. Figure 4g shows the performance comparison for different policies on different benchmark mixes. In this paper, we use Fair Speedup (FS) as our performance indicator. The FS of a partitioning scheme is defined as the harmonic mean of per-application speedup with respect to the equal resource share case (i.e., peak frequency for all applications) [4]. The FS achieved by a scheme can be expressed as

$$FS(scheme) = N_{app} / \sum_{i=1}^{N_{app}} \frac{ET_{app_i(scheme)}}{ET_{app_i(base)}} \quad (4)$$

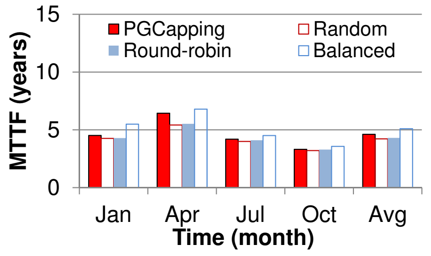
$ET_{app_i(scheme)}$ is the execution time of the i^{th} application under a certain power management scheme. $ET_{app_i(base)}$ is the execution time of running the i^{th} application of the peak frequency level all the time. N_{app} is the number of applications in the system, i.e., the set of applications that execute together. FS is an indicator of the overall improvement in execution efficiency gained across the applications. Figure 4g shows that *PGCapping* outperforms *per-core-DVFS-only*, *PCPG-only*, *ETurboBoost*, and *Per-core-DVFS+PCPG* by 16.9%, 42.0%, 13.9%, 21.4% on average, respectively. In general, the reason that *PGCapping* can outperform baselines is that it fully utilizes the power budget. Therefore, the cores are generally running at a higher frequency level. Specifically, *PGCapping* reserves power headroom for cores running useful workload by turning off unused cores. There-



(a) Lifetime-balancing results with real-world data center utilization traces.



(b) Lifetime-balancing with solar power budget.



(c) Combined test.

Figure 5: *PGCapping* achieves very close core lifetime balancing performance with *Balanced* (a state-of-the-art core lifetime balancing scheme [9]). *PGCapping* outperforms *Random* and *Round-robin* baselines.

fore, it outperforms *per-core-DVFS-only*. Then, *PGCapping* uses per-core DVFS/overclocking to fully utilize the power headroom and explore the runtime inter-core variations for optimized performance. Therefore, *PGCapping* can outperform *ETurboBoost*, and *Per-core-DVFS+PCPG*. Althought *EQuicksearch* has a full set of the possible configuratioin combination to explore for optimized performance at a coarse time interval, it cannot eliminate unnecessary oscillation of power gating due to system noises. In contrast, *PGCapping* benefits from fine-time-grain DVFS and robust PI controller in the power gating part. Therefore, *PGCapping* achieves 11.5% better performance than *EQuicksearch* on average.

5.4 Lifetime Balancing

In this section, we first introduce three baselines for our lifetime balancing part. Next, we present our simulation results.

We use three baselines to compare with our power-gating-based lifetime balancing algorithm in *PGCapping* (Section 3.3). The first baseline is *Random*: when we need to turn on cores, we randomly select the turned-off cores to turn on; when we turn off cores, we randomly select the turned-on cores to turn off. The second baseline is *Round-robin* [19]: when we turn off cores, we start from the first core, then the second, and so on. When we turn on cores, we also start with the first. If it is already turned on, we turn to the next. The third baseline *Balanced* is an extension of the lifetime balancing algorithm discussed in [9]. *Balanced* sorts the cores in lifetime descended order and the next step available pre-core DVFS levels or on/off states in descended order. Next, it maps the two lists. *Balanced* strictly enforces lifetime balancing among cores at each DVFS and PCPG interval without considering the actuation overhead. *Balanced* negatively impacts performance. Therefore, it is not suitable for real-world implementation. However, it has been suggested as a reasonable approximation for the best-effort per-core DVFS based lifetime balancing [9].

In Figure 5, we conduct three test cases to evaluate the lifetime balancing with real-world workload and power budget variations.

In Figure 5a, we evaluate our proposed life-balancing solution in *PGCapping* with real-world workload variations. We assign a fixed power cap (e.g., 62W) to the processor while using a continuous 30-day real-world server utilization trace [42]. We derive the 62W budget by calculating the overall 30-day average utilization and test run the simulator with the average utilization at peak frequency. *PGCapping* outperforms *Random* and *Round-robin* by 9.2% and 8.1%. *PGCapping* achieves 92.3% of *Balanced* in the mean time to failure (MTTF). In Figure 5b, we evaluate our proposed life-balancing solution with real-world power budget variations. We assume the processor is powered by a solar panel (e.g. BP3180N [7]) and hosting HPC applications, which is a newly proposed green-energy solution [24]. Therefore, the power budget changes with the solar radiation but the utilization is fixed to 100%. We use meteorological data from the Measurement and Instrumentation Data Center (MIDC) [32] at the locations that have different solar energy resource potentials (e.g., AZ, CO, NY, TN) across different seasons (e.g. the middle of Jan., Apr., Jul. of 2011 and Oct. of 2010) to calculate the maximum available solar power envelopes. Then, we run our simulator with the power budget envelopes. On average, *PGCapping* outperforms *Random* and *Round-robin* by 11.0% and 7.1%. *PGCapping* achieves 90.7% of *Balanced* in MTTF. In Figure 5c, we assume the processor is powered by a solar panel with real-world server utilization trace. We exhaustively test the 30-day trace with 4 different seasons and 4 different locations and present the result in Figure 5c (e.g., totally 4x4 30-day tests). *PGCapping* outperforms *Random* and *Round-robin* by 9.2% and 7.2%. It achieves 90.4% of *Balanced* in terms of MTTF in this test case. Interestingly, we observe that certain location (e.g., TN) has longer MTTF and greater differences among different policies. From the analysis of the solar radiation traces [32], the available solar power is relatively low (longer MTTF) and irregular (greater difference among different policies) in Tennessee (TN), possibly due to the valley location and related weather.

When *PGCapping* uses PCPG to enforce the power budget, it always turns on the longer-lifetime cores and turns off the shorter-lifetime cores. Therefore, none of the on-chip cores becomes significantly worn out. That is the reason why *PGCapping* can outperform *Random* and *Round-robin*. Due to real-world power budget and utilization variations, there are always certain chances to turn on/off some cores. Therefore, although *PGCapping* does not strictly balance lifetime among cores at each DVFS and PCPG interval, it still achieves a CMP lifetime that is 90.4% of that of *Balanced*.

6. CONCLUSION

Both DVFS and power gating are anticipated to be important power adaptation knobs. However, current studies on integrating power gating and DVFS focus on deciding the power gating and DVFS levels in a tightly coupled fashion, with much less attention given to the direction of decoupled designs. In this paper, we have explored *PGCapping*, a decoupled design to integrate per-core power gating with DVFS/overclocking for CMP power capping. However, both per-core DVFS and overclocking may make some cores age much faster than others and thus become the reliability bottleneck in the whole system. Therefore, *PGCapping* also balances the lifetimes of the CMP cores using power gating. Our empirical results show that *PGCapping* achieves up to 42.0% better average application performance than five state-of-the-art baselines. Furthermore, our extensive simulation results with real-world traces demonstrate that *PGCapping* can increase the CMP lifetime by 9.2% on average.

7. ACKNOWLEDGMENT

This work was supported, in part, by NSF under grants CCF-1143605, CNS-0915959, and CNS-1143607 (NSF CAREER Award), and by ONR under grant N00014-09-1-0750.

8. REFERENCES

[1] AMD. AMD dragon platform technology performance tuning guide, 2009.

[2] AMD. AMD family 10h server and workstation processor power and thermal data sheet, 2010.

[3] W. L. Bircher et al. Predictive power management for multi-core processors. In *WEED*, 2010.

[4] R. Bitirgen et al. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*, 2008.

[5] J. Blome et al. Self-calibrating online wearout detection. In *MICRO*, 2007.

[6] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, Third Edition*. O'Reilly, 1997.

[7] BP. BP3180N datasheet. http://www.solardepot.com/pdf/BP3180N.pdf, 2009.

[8] M. Chen et al. Coordinating processor and main memory for efficient server power control. In *ICS*, 2011.

[9] A. K. Coskun et al. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *MMCS*, SIGMETRICS, 2009.

[10] G. F. Franklin et al. *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[11] X. Fu, X. Wang, and C. Lefurgy. How much power oversubscription is safe and allowed in data centers? In *ICAC*, 2011.

[12] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale cmps through core overclocking. In *PACT*, 2007.

[13] M. Horowitz. Scaling, power, and the future of CMOS. In *VLSID*, 2007.

[14] L. Huang et al. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In *DATE*, 2009.

[15] Intel. Intel Core i7 processor extreme edition and Intel Core i7 processor datasheet. Technical report, Intel Corporation, November 2008.

[16] C. Isci et al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO*, 2006.

[17] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*, 2003.

[18] A. Kansal et al. Virtual machine power metering and provisioning. In *SoCC*, 2010.

[19] U. R. Karpuzcu et al. The bubblewrap many-core: popping cores for sequential acceleration. In *MICRO*, 2009.

[20] W. Kim et al. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, 2008.

[21] J. Lee and N. S. Kim. Optimizing throughput of power- and thermal-constrained multicore processors using dvfs and per-core power-gating. In *DAC*, 2009.

[22] C. Lefurgy et al. Server-level power control. In *ICAC*, 2007.

[23] J. Leverich et al. Power management of datacenter workloads using per-core power gating. *IEEE Comput. Archit. Lett.*, 8, 2009.

[24] C. Li et al. Solarcore: Solar energy driven multi-core architecture power management. In *HPCA*, 2011.

[25] J. Li et al. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *HPCA*, 2006.

[26] T. Li et al. Spin detection hardware for improved management of multithreaded systems. *IEEE Trans. Parallel Distrib. Syst.*, 17(6), 2006.

[27] A. Lungu et al. Dynamic power gating with quality guarantees. In *ISLPED*, 2009.

[28] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. In *ISCA*, 2011.

[29] N. Madan et al. A case for guarded power gating in multi-core processors. In *HPCA*, 2011.

[30] K. Meng et al. Multi-optimization power management for chip multiprocessors. In *PACT*, 2008.

[31] Z. Mwaikambo and A. Raj. Linux kernel hotplug CPU support. *Linux Symposium*, 2, 2004.

[32] NREL. Measurement and instrumentation data center. http://www.nrel.gov/midc, 2011.

[33] R. Raghavendra et al. No power struggles: Coordinated multi-level power management for the data center. In *ASPLOS*, 2008.

[34] K. Rangan et al. Thread motion: Fine-grained power management for multi-core systems. In *ISCA*, 2009.

[35] J. Srinivasan et al. Lifetime reliability: toward an architectural solution. *Micro, IEEE*, 2005.

[36] N. Sturcken et al. A 2.5d integrated voltage regulator using coupled-magnetic-core inductors on silicon interposer delivering 10.8a/mm2. In *ISSCC*, 2012.

[37] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *ISCA*, 2008.

[38] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *HPCA*, 2008.

[39] X. Wang et al. SHIP: Scalable hierarchical power control for large-scale data centers. In *PACT*, 2009.

[40] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA*, 2009.

[41] J. A. Winter et al. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *PACT*, 2010.

[42] C. Zhuo et al. Process variation and temperature-aware reliability management. In *DATE*, 2010.