Scalable Power Control for Many-Core Architectures Running Multi-threaded Applications

Kai Ma[†‡], Xue Li[†], Ming Chen[†], and Xiaorui Wang[†‡]
[†]Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
[‡]Department of Electrical and Computer Engineering
The Ohio State University, Columbus, OH 43210
{kma3, xli44, mchen11, xwang}@eecs.utk.edu

ABSTRACT

Optimizing the performance of a multi-core microprocessor within a power budget has recently received a lot of attention. However, most existing solutions are centralized and cannot scale well with the rapidly increasing level of core integration. While a few recent studies propose power control algorithms for many-core architectures, those solutions assume that the workload of every core is independent and therefore cannot effectively allocate power based on thread criticality to accelerate multi-threaded parallel applications, which are expected to be the primary workloads of many-core architectures. This paper presents a scalable power control solution for many-core microprocessors that is specifically designed to handle realistic workloads, i.e., a mixed group of single-threaded and multi-threaded applications. Our solution features a three-layer design. First, we adopt control theory to precisely control the power of the entire chip to its chip-level budget by adjusting the aggregated frequency of all the cores on the chip. Second, we dynamically group cores running the same applications and then partition the chip-level aggregated frequency quota among different groups for optimized overall microprocessor performance. Finally, we partition the group-level frequency quota among the cores in each group based on the measured thread criticality for shorter application completion time. As a result, our solution can optimize the microprocessor performance while precisely limiting the chip-level power consumption below the desired budget. Empirical results on a 12-core hardware testbed show that our control solution can provide precise power control, as well as 17% and 11% better application performance than two state-of-the-art solutions, on average, for mixed PARSEC and SPEC benchmarks. Furthermore, our extensive simulation results for 32, 64, and 128 cores, as well as overhead analysis for up to 4,096 cores, demonstrate that our solution is highly scalable to many-core architectures.

Categories and Subject Descriptors

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

ISCA'11, June 4–8, 2011, San Jose, California, USA.

General Terms

Design, Management, Performance, Experimentation

Keywords

Chip multiprocessor, many-core architecture, power control, power capping, thread criticality, scalability, control theory

1. INTRODUCTION

Power dissipation has become a first-class constraint in current microprocessor design. As the gap between peak and average power widens with the rapidly increasing level of core integration, it is important to control the peak power of a many-core microprocessor to allow improved reliability and reduced costs in chip cooling and packaging. Therefore, compared with the extensively studied power minimization problem, an equally, if not more, important problem is to precisely control the peak power consumption of a many-core microprocessor to stay below a desired budget while optimizing its performance.

Scalability is the first key challenge in controlling the power consumption of a many-core microprocessor. While various power control solutions have been proposed for multi-core microprocessors (*e.g.*, [18, 28, 45]), the majority of current solutions relies on centralized decision making and thus cannot be applied directly to many-core systems. For example, the MaxBIPS policy [18] uses an exhaustive search to find a combination of DVFS (Dynamic Voltage and Frequency Scaling) levels for all the cores of a microprocessor. The search is predicted to result in the best application performance while maintaining the power of the chip below the budget. While this solution works effectively for microprocessors with only a few cores, MaxBIPS does not scale well because the number of possible combinations increases exponentially with the number of cores. Therefore, highly scalable approaches need to be developed for many-core architectures.

The requirement to host multi-threaded applications is the second challenge for many-core power control. Although a few recent studies [47, 39, 29] present scalable control algorithms for many-core architectures based on per-core DVFS, they do *not* consider multi-threaded parallel applications and assume that the workload of every core is independent. As a result, these solutions may unnecessarily decrease the DVFS levels of the CPU cores running the critical threads in barrier-based multi-threaded applications. The unawareness of thread criticality can exacerbate the load imbalance in multi-core microprocessors and thus lead to unnecessarily long application execution times and undesired barrier stalls. This issue is particularly important for many-core architectures whose primary workloads are expected to be multi-threaded applications. Furthermore, many-core systems are likely to simultaneously host

a mixed group of single-threaded and multi-threaded applications, due to the increasing trend of server consolidation, to fully utilize the affluent core resource [27, 8]. Therefore, a power control algorithm must be able to handle such realistic workload combinations and utilize thread criticality to efficiently allocate power among the cores that are running different applications.

Another major challenge in multi-core or many-core power control is accurate power monitoring [34]. Although the power consumption of a microprocessor can be measured by sensing the current fed into the chip [46], direct power measurement of a single core on a multi-core or many-core die is not yet available. On-die current sensors have been proposed, but have rarely been used in production due to problems such as area and performance overhead and calibration drift introduced by process variations [11]. It is possible to estimate the core power at runtime by counting the component utilizations (*e.g.*, cache accesses) and computing power based on a per-component power model. However, such direct computation of core and structure power at runtime is complex due to a large number of performance statistics required [46]. Since many-core systems are expected to have many simple cores [8], it may not be desirable to adopt an approach that requires a lot of extra hardware and statistics collection. Recently, Kansal et al. [21] have shown that the CPU power consumption of each Virtual Machine (VM) on a server can be estimated by adaptively weighting only one metric (CPU utilization) of each VM. However, they did not explicitly consider the impact of DVFS on their model despite the fact that the power consumption is different under different DVFS levels even for the same application. We propose to extend their work to estimate the power consumption of each core in a DVFS environment by taking both DVFS level and utilization into consideration. As a result, many-core power control can be evaluated on a real hardware platform instead of just by simulations as in previous work [47, 39].

In this paper, we propose a novel and highly scalable power control solution for many-core microprocessors that is specifically designed to handle realistic workload combinations. Our control solution features a three-layer design. First, we adopt control theory to precisely control the power of the whole chip to its chip-level budget, with theoretically guaranteed accuracy and stability, by adjusting the aggregated frequency quota of all the cores on the chip. In a DVFS-enabled system, aggregated frequency is defined as the summation of the DVFS levels of all the cores normalized to the peak DVFS level of one core. Second, we dynamically group cores running the same applications and then partition the aggregated chip-level frequency quota derived from the chip-level power controller among different groups for optimized overall microprocessor performance. Finally, we partition the group-level aggregated frequency quota among the cores in each group based on measured thread criticality for a shorter application completion time. As a result, our solution can optimize the processor performance while precisely limiting the chip-level power consumption below the desired budget. Specifically, this paper makes the following major contributions:

- We propose a highly scalable power control solution for many-core architectures running multi-threaded applications. Our solution partitions the limited chip-level power budget among different applications and cores based on measured application performance and thread criticality.
- We adopt feedback control theory as a theoretical foundation to control the power consumption of a many-core chip to its desired power budget. This rigorous design methodology is in sharp contrast to heuristic-based solutions that rely on extensive manual tuning.
- Since the power consumption of a core cannot be directly measured in real multi-core microprocessors, we extend the technique of estimating the power consumption of a VM on a physical server to estimate the power consumption of a CPU core and validate the estimation model on a hardware testbed.
- We implement our control solution on a 12-core AMD Opteron processor and present empirical results to demonstrate that our solution achieves better application performance within a given power budget than two state-of-the-art solutions. Our extensive simulation results with 32, 64, and 128 cores, as well as overhead analysis for up to 4,096 cores, demonstrate the scalability of our solution in many-core architectures.

The rest of this paper is organized as follows. Section 2 discusses the system architecture of our control solution. Section 3 presents the chip-level power controller design. Section 4 describes dynamic aggregated frequency quota partitioning at the chip and group levels. Section 5 presents the per-core power estimation technique. Section 6 introduces our hardware testbed, simulation setups, and the implementation details of our solution. Section 7 presents our evaluation results. Section 8 discusses the related work and Section 9 concludes the paper.

2. SYSTEM ARCHITECTURE

In this section, we present a high-level description of our three-layer power control solution. As shown in Figure 1, in the first layer, the chip-level power controller controls the power consumption of the whole chip to the chip power budget by adjusting the aggregated frequency quota (i.e., summation of normalized DVFS levels) of all the cores. The second layer, i.e., the chip-level frequency quota partitioning layer, partitions the chip-level aggregated frequency quota among the groups of cores, which host different applications proportionally to a metric called power efficiency (defined in Section 4.1.2). The third layer, i.e., the group-level frequency partitioning layer, further partitions the group aggregated frequency quota among all the cores in each group, which host coupled threads of the same application, based on thread criticality (defined in Section 4.2). The aggregated frequency quota is first partitioned among different applications (i.e., groups of cores) and then partitioned among coupled threads (i.e., individual cores) to achieve optimized performance. As a result, if the aggregated frequency quota of every core is enforced, the power of the entire chip can be controlled to stay within the desired power budget. In this paper, we adopt DVFS to enforce the frequency quota of each core, but our solution can also work with other frequency scaling techniques such as clock modulation. We assume that the frequency of each core can be adjusted individually in future many-core systems based on various industry practices and research studies [47, 39]. For example, IBM and AMD have implemented per-core DVFS on commercial massive multi-core microprocessors (POWER7 8-core and Opteron 12-core systems). Moreover, Intel has implemented per-tile DVFS on its 24-tile many-core experimental chips [17]. In addition, a 167-core computational platform with per-core DVFS support has been implemented recently [42]. Even in the systems without physically implemented per-core DVFS (*e.g.*, multi-power-island chips), Rangan et al. [36] have shown that thread migration on systems with only two power states can be used to approximate the functionality of continuous, per-core DVFS.

As shown in Figure 1, the key components in the chip-level power control layer include a power controller and a power monitor. The following steps are invoked at the end of every control period: 1) the power monitor (*e.g.*, an on-board power measurement circuit [46]) measures the power consumption of the chip in

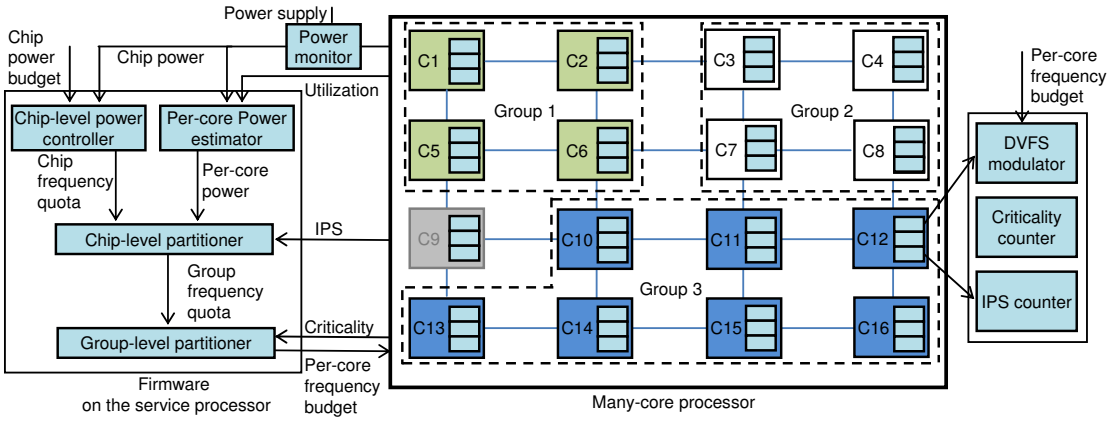


Figure 1: Three-layer power control architecture for a 16-core chip multiprocessor. Cores running the same multi-threaded applications are grouped together. Idle cores (e.g., C9) are transitioned into a low power mode.

the last control period and sends the value to the power controller and 2) the power controller computes the new aggregated frequency quota for all the cores of the chip based on the desired power budget and measured power consumption. The aggregated frequency quota is then partitioned to optimize the system performance in the partitioning layers.

The key components in the chip-level frequency quota partitioning layer include a single chip-level partitioner and an IPS (Instructions Per Second) counter on each core. In order to effectively partition the power budget, we need to be able to calculate the power efficiency of each core. We adopt IPS/Watt as our power efficiency metric, which has been used by Intel as the power efficiency metric [16]. The chip-level frequency quota is partitioned among multiple groups of cores periodically. At the end of each control period, the partitioner collects the grouping information of all the cores based on the OS scheduler (details are described in Section 4.1.1). Each group of cores hosts all the threads of the same application. If a group consists of only one core, we refer to it as a *single-threaded group*; otherwise, we refer to it as a *multi-threaded group*. If a core is idle, we transition it to a low-power mode. For example, Core 1, 2, 5, and 6 are grouped together since they run four threads of a parallel application based on the scheduling information from OS. Core 9 is transitioned into the low-power mode since it is idle. The chip-level partitioner computes the power efficiency based on the IPS and the estimated power of each core, then calculates the overall power efficiency of each group by summing up the efficiency of each core in the group. The chip-level partitioner partitions the aggregated frequency quota of the entire chip among the groups proportionally to the overall power efficiency of each group. Note that since the power control period at the chip level can be configured shorter than the OS scheduling period, we assume the mapping between the threads and cores does not change within each control period. The same assumption has been made in [47].

The group-level frequency quota partitioning layer includes a group-level partitioner in each group, a criticality counter, and a DVFS modulator in each core. At the end of each control period, the criticality counter on each core monitors the criticality metric (defined in Section 4.2) and forwards it to the partitioner. The partitioner receives the allocated group frequency quota from the chip-level partitioner and partitions the frequency quota among all the cores in the group based on the thread criticality of each core. Then, the DVFS modulator of each core changes the DVFS level of the core accordingly.

Because the computation of the controller may change the overall aggregated frequency quota and the recalculation of chip-level partitioner may change the group aggregated frequency quota, the three layers run sequentially at the end of every control period. Figure 1 shows a possible implementation that the three layers are integrated as firmware on the service processor, similar to IBM POWER7's power control module [46]. We also discuss other implementation possibilities in Section 6.3.

3. CHIP-LEVEL POWER CONTROL

In this section, we introduce the chip-level power controller that controls the power consumption of the entire chip to the desired power budget by adjusting the aggregated frequency quota (i.e., summation of normalized DVFS levels). A key advantage of the control-theoretic design approach is that it can tolerate a certain degree of modeling errors and adapt to online model variations based on dynamic feedback [15]. 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.

We first introduce some notation. T_c is the control period. M is the number of cores on this chip. $cp(k)$ is the power consumption of the entire chip in the k^{th} control period. $f(k)$ is the total aggregated frequency of all the cores on the chip in the k^{th} control period. The dynamic range of $f(k)$ is $L * M \leq f(k) \leq M$, relative to the peak of one core, where L is the lowest available DVFS normalized to the peak level. We assume that our target system is a homogeneous-core system, which is the dominant configuration of the current multi-core and many-core systems [17, 43, 4]. However, extending for heterogeneous-core systems is straightforward by scaling $f(k)$. For example, if we have a more powerful core in the system along with the normal cores, instead of taking the dynamic range of the more powerful core as L to 1 like the normal core, we count it as L' to H'. Both L' and H' are derived by scaling the available DVFS levels of the powerful core to the peak DVFS level of a normal core. $\Delta f(k) = f(k+1) - f(k)$. P_t is the power budget of the whole chip, which can be determined by the thermal and power supply constraints of the processor or specified by the user during runtime. $e(k)$ is the control error, specifically, $e(k) = P_t - cp(k)$. The control goal is to direct $cp(k)$ to converge to P_t within a certain number of control periods by adjusting $f(k)$.

System Modeling. We now model the dynamics of the controlled system, namely the relationship between the controlled variable, i.e., $cp(k)$, and manipulated variable, i.e., $f(k)$. Existing studies by both Raghavendra et al. [35] and Wang et al. [44] have shown that the processor power can be modeled as an approximately linear function to the DVFS level within the limited DVFS adaptation range available in real multi-core processors. In this paper, the power consumption of a processor is modeled similarly as:

$$cp(k) = a\Delta f(k-1) + cp(k-1), \quad (1)$$

where a is the generalized parameters that may vary for different chips and applications. a is also the scaling factor that characterizes the impact of DVFS change on chip power. In our design, we derive a by using the data sheet full power range (from the idle power to the maximum power of the chip [3]) divided by the dynamic range of $f(k)$. We conducted stability analysis [15] on our controlled system. The results show the stability range of a is from 0 to $2a$. Since we used the maximum possible a at design time, the variation of a could never exceed the range. The control loop is theoretically guaranteed to converge to the set point for all possible workloads.

Controller Design. Proportional-Integral (PI) control can provide robust control performance despite considerable modeling errors. Based on the system model (1), we design a PI controller as follows:

$$f(k) = f(k-1) + K_1 e(k) - K_1 K_2 e(k-1). \quad (2)$$

Following the standard pole placement method [15], we can choose our control parameters as $K_1 = 1/a$ and $K_2 = 0$, such that the controlled system is stable and has a zero steady state error. The detailed steps can be found in a standard control textbook and are skipped due to space limitations. The desired aggregated frequency quota of all the cores on the chip in the k^{th} control period can be computed accordingly as:

$$f(k) = f(k-1) + \frac{(P_t - cp_i(k))}{a}. \quad (3)$$

4. DYNAMIC AGGREGATED FREQUENCY PARTITIONING

We first introduce the details of the aggregated frequency (i.e., summation of normalized DVFS levels) partitioning schemes at the chip and group levels.

4.1 Chip-level Partitioning

A many-core microprocessor may host multiple applications simultaneously. For example, a virtualized many-core system may host multiple VMs and each of the VM may host a different application. If the power budget is limited, different power allocations among the applications may lead to different system performance. To achieve high overall performance is one of the most fundamental goals for many-core systems [8]. The goal of the chip-level partitioning is to dynamically partition the chip-level aggregated frequency quota computed in the chip power controller (Equation (3)) among different applications, such that we can achieve the optimized system performance. In this paper, we use Fair Speedup (FS) as the 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) [12, 7]. The FS achieved by a scheme can be expressed as $FS(scheme) = N_a / \sum_{i=1}^{N_a} \frac{ET_{app_i(scheme)}}{ET_{app_i(base)}}$. $ET_{app_i(scheme)}$ 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_a 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. It is also a metric of fairness. In the following sections, we first introduce how to group the cores that run the threads of the same application based on the scheduling information in the OS. We then present the aggregated frequency quota partitioning among the groups.

4.1.1 Core Grouping

In many-core microprocessors, different threads run simultaneously on different cores. We place the cores that host the threads of the same application into a group. Therefore, the number of groups is equal to the number of applications running on all the cores. The benefit of core grouping is to reduce the coupling of the power demand among different applications.

In this project, we assume that the mapping between the threads and cores does not change within a certain period (i.e., the scheduling period). Since the scheduling interval in operating systems is in tens of milli-seconds, if we conduct the power control in a shorter period, this assumption is valid. The same assumption is made in [47, 5, 7, 10, 28]. At the end of each scheduling period, the chip-level frequency partitioner may collect the grouping information from the OS. If we implement the algorithm as a loadable kernel module of OS, the grouping information can be derived from a system function. If we implement the controller as a piece of hardware on the chip, this information exchange between hardware and software can be achieved by adding special purpose registers on the chip. If the proposed solution is implemented as firmware running on the service processor on the motherboard, the information exchange between the main processor and service processor can be achieved via external ports [46].

4.1.2 Aggregated Frequency Partitioning

Before we discuss the chip-level power partitioning, we first introduce some notation. A many-core microprocessor has N groups of cores and group i runs application *i*, where $1 \leq i \leq N$. IPS_i is the average IPS of group i when running the i^{th} application on the many-core microprocessor without any power constraint. IPS_i can be derived by conducting application profiling on the desired number of cores at the peak DVFS level and then calculating the average IPS of each core. Note the profiling is only performed once for each application on the desired number of cores. The OS can send IPS_i to the controller via on-chip registers. $ips_i(k)$ is the measured IPS of the i^{th} group. $WT_i(k)$ is the estimated power of the i^{th} group. Since each group may consist of multiple cores, $ips_i(k)$ and $WT_i(k)$ are the accumulated IPS and power of all the cores in the i^{th} group.

To achieve optimized overall performance, the aggregated frequency quota partitioned among different groups should be proportional to the ratio between the performance and power consumption (i.e., $ips_i(k)/WT_i(k)$). However, this may lead to the following problem. Some applications intrinsically have a low IPS even without any power constraint. To partition power based on IPS is unfair to those applications if they run simultaneously with other applications that have intrinsically high IPSs. To address this problem, we use the relative IPS, $rips_i(k)$, as the performance metric in this paper, which is the measured IPS $ips_i(k)$ normalized to IPS_i. Specifically, $rips_i(k) = ips_i(k)/IPS_i$, similar to the fairness definition used in [22]. We define the power efficiency of the i^{th} group $e_i(k)$ as the ratio between $rips_i(k)$ and $WT_i(k)$. Specifically, $e_i(k) = rips_i(k)/WT_i(k)$.

In this paper, we partition the chip-level aggregated frequency quota among groups proportionally based on the power efficiency of each group to achieve the optimized performance

$$fg_i(k) = \frac{e_i(k-1)}{\sum_{j=1}^{N} e_j(k-1)} f(k), \quad (4)$$

where $f(k)$ is the aggregated frequency quota of the entire chip and $fg_i(k)$ is the aggregated frequency allocation for the i^{th} group in the k^{th} control period. In systems that need to support application priority, we can assign different weights to the co-scheduled applications when we calculate $fg_i(k)$.

Table 1: Workload mixes used in testbed and simulation experiments.

1. Physical testbed workload mixes		
Mixes	**PARSEC 2.1 , SPEC2006**	**Aggregate Effect**
mix1	12-perlbench	all separated applications
mix2	12-Streamcluster	high-barrier parallel workload
mix3	8-swaptions, 4-omnetpp	no-barrier parallel workload
mix4	4-x264, 8-fluidanimate	no-barrier, high-lock workload mix
mix5	4-(blackscholes, bodytrack, xalancbmk, povray)	low-barrier and high-barrier mix
mix6	4-(vips,facesim),1-(libquantum,astar,soplex,dealII)	random mix
2. Simulation workload mixes		
Mixes	**SPLASH-2, SPEC2006**	**Aggregate Effect**
mix1	water (nsquared)	all parallel application
mix2	dealII	all seperated applications
mix3	FFT,Ocean_non,LU_con,LU_non (each occupies 1/4 number of cores)	random mix

4.2 Group-level Partitioning

The goal of group-level aggregated frequency quota partitioning is to further partition the group frequency quota among all the cores running the threads of the same application, such that each thread has a balanced progress at the common barriers. For those one-threaded groups, the core quota is the same as the group quota. For those multi-threaded groups, the problem of achieving an optimized performance in a group is translated into discerning which running threads are more critical (i.e., slower) and then allocating more aggregated frequency to the critical threads to expedite the progress of the entire application.

In this paper, we adopt a thread criticality prediction approach proposed by Bhattacharjee and Martonosi [5], which considers both L1 and L2 cache misses. Compared with other approaches [10, 25, 26], the advantage of this predictor is that it can handle both barrier and non-barrier parallel workloads. The criticality of core j in the i^{th} group in the k^{th} period is

$$cr_{ij}(k) = N(L1_{miss}) + \frac{(L1L2_{penalty}) \times N(L1L2_{miss})}{L1_{penalty}}, \quad (5)$$

where $N(L1_{miss})$ is the number of L1 misses that hit in the L2 cache, $N(L1L2_{miss})$ is the number of L1 misses that also miss in the L2 cache, and $L1L2_{penalty}$ and $L1_{penalty}$ are L2 and L1 cache miss penalties, respectively. The cache miss penalty is measured in CPU cycles. Within a parallel working group, a higher criticality value implies a more poorly-cached, slower thread [5], which means that additional power needs to be shifted to that thread from the non-critical threads (with smaller criticality values) to reduce the runtime imbalance. In our design, we proportionally sub-partition the frequency quota of a multi-threaded group to its cores based on criticality as follows

$$fc_{ij}(k) = \frac{cr_{ij}(k-1)}{\sum_{k=1}^{M_i} cr_{ik}(k-1)} fg_i(k), \quad (6)$$

where $fc_{ij}(k)$ is the target frequency of Core j in Group i in the k^{th} control period. M_i is the number of cores in Group i.

5. CORE-LEVEL POWER ESTIMATION ON PHYSICAL TESTBED

In our power management solution, the chip-level partitioning is conducted according to the relative power efficiency of each group. Therefore, we need a reasonable estimation of the power consumption of each core. Besides, one of our baselines, *Steepest Drop* [47], also assumes the knowledge of the power consumption of each core, even though real microprocessors available in today's market cannot yet provide such information. In this section, we introduce our per-core power estimation method.

Although the power consumption of each individual core cannot be directly measured in today's microprocessors, previous work by Kansal et al. [21] has shown the CPU power consumption of each VM on a server can be estimated by adaptively weighting the CPU utilization of the VM. However, they did not explicitly consider the impact of DVFS in their model despite the fact that power consumption scales with different DVFS levels. We extend their work to estimate the power consumption of each core under DVFS environment by taking both DVFS level and utilization into consideration. The utilization metric represents the high-level workload characteristics, while the DVFS level represents the hardware working condition of the core. Power consumption is the interactive result of both the hardware and software parts. We adopt the commonly used multiplication operation to model the interaction among different parts [31]. Therefore, the total power consumption of the chip is modeled as:

$$CP = \sum_{i=1}^{N} U_i * f_i * W + C, \quad (7)$$

where CP is the total power consumption of the entire chip. U_i is the utilization of the i^{th} core, $1 \leq i \leq M$. M is the number of cores. f_i is the DVFS level of Core i. W is the weight and C is the idle power of the chip. In this model, the static power of the chip has been captured by C. We do not explicitly consider the dynamic power of the uncore part because the uncore power is actually driven by the core part. For example, the last level cache accesses are introduced by the cache misses or writebacks from the cache of each individual core. Instead of modeling the dynamic power of the uncore part explicitly, we attribute it to the corresponding core part since the purpose of our per-core power estimation is to support power allocation among the cores. Therefore, we simply sum the power of all the cores as the power of the entire chip. CP can be measured with a multimeter (as described in Section 6.1). U_i and f_i can be measured with the performance monitoring registers. W and C are updated by linear regression at runtime. Note this simple, yet effective, estimation method has only two unknown variables in the regression. Moreover, the number of unknown variables does not scale with the number of cores. Since our testbed is a homogeneous-core system, the estimated power of each core CP_i is

$$CP_i = W * U_i * f_i + \frac{C}{M}. \quad (8)$$

For heterogeneous-core systems, we could extend the model by scaling f_i (as described in Section 3). Intuitively, the workload characteristics could be more accurately captured by using different

Table 2: Simulator configuration parameters for SESC.

CMP	32,64,128 cores	Private L1 d/i-cache	2-way, 64 bytes block 32KB, 2 cycles hit latency
Core	Alpha 21264 like	Private L2 cache	8-way, 64 bytes block 256KB, 12 cycles hit latency
Peak Frequency/Vdd	1GHz/1.0V	On-chip network	(4,8,16)x8 mesh, wormhole switching
Technology/Temperature	65nm/80°	Network latency	1 cycle local port latency 4 cycles cross router latency
fetch/issue/commit width	4/4/4	Main memory latency	400 cycles

weights (W_i instead of W) for different cores. However, this approach would make the complexity of the on-line regression problem increase linearly with the number of cores, which might not be favorable for many-core architectures.

6. IMPLEMENTATION

In this section, we first introduce the physical testbed and the simulation environment used in our experiments. Next, we discuss the possible implementation of our solution on a physical chip.

6.1 Testbed

Our testbed is a 12-core AMD Opteron 6168 processor, which supports per-core DVFS with five different levels [3]. The operating system is OpenSUSE 11.2 with Linux kernel 2.6.31. To evaluate our power control policy, we simultaneously run different combinations of selected benchmarks from the PARSEC 2.1 [6] and SPEC CPU2006 suites on our physical testbed. We use the SPEC suite subset identified in [33] to represent the major characteristics of SPEC CPU2006. The constructed workload combinations of PARSEC and SPEC cover a variety of different aggregate effects and are listed in Table 1. In Table 1, the number-appname notation is the number of threads of the application with the name of appname for PARSEC and SPLASH-2 workloads; for SPEC2006 workload, it is the number of copies of the application with the name of appname.

To measure the power consumption of the processor, we use the approach proposed in [14, 19]. 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 probe is clamped to the 12V lines and produces a voltage signal proportional to the current running through the lines with a coefficient of 1mv/A. The resultant voltage signal is then measured with the multimeter. The measured value is read by the server through a USB cable using a USBTMC device driver. The accuracy of the current probe is (3.5% of reading + 0.5A). The power consumption of each core is estimated periodically (described in Section 5).

on our prototype testbed, we implement the control algorithm as an OS daemon process to control the target processor, though the controller can be implemented in the service processor firmware in a real system. The controller periodically reads the power consumption from the power monitor and the performance statistics from OS. It then executes the control algorithm presented in Section 3. As the outputs of the control algorithm, new DVFS levels are calculated and enforced in the next control period. The control period T_c for the controller is set to 1 second because the timer resolution in Linux is 10ms. Note that much shorter control periods could be used in a real firmware or on-chip implementation.

Since the new aggregated DVFS level periodically received from the power controller can be any value, it may not be exactly one of the DVFS levels supported by the processor. Therefore, a DVFS modulator is needed to approximate the desired level with a series of supported DVFS levels. For example, to approximate 0.9 during a control period, the modulator would output the sequence, 0.8, 1.0, 0.8, 1.0 etc, on a smaller timescale. To implement the approximation, we use the first-order delta-sigma modulator [24] to generate the sequence in each control period. This type of modulator is commonly used in analog-to-digital signal conversion. Clearly, when the sequence has more numbers during a control period, the approximation will be better, but the actuation overhead may become higher. On our prototype testbed, we use 10 subintervals to approximate the desired DVFS level and each subinterval is 100ms. As a result, the effect of actuation overhead on system performance is no more than 0.02% ($20\mu s$ of DVFS overhead [40] divided by $100ms$), even in the worst scenario when the DVFS level needs to be changed in each subinterval.

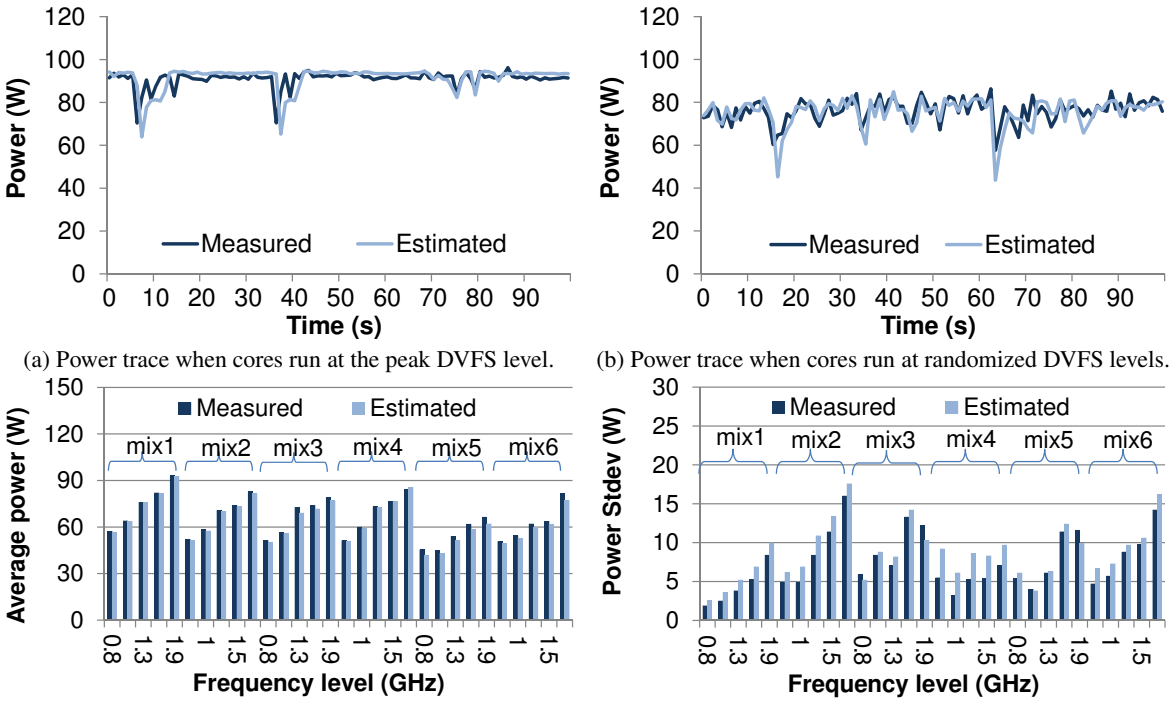
6.2 Simulation Environment

To simulate different applications running in the many-core system, we simultaneously run different combinations of selected benchmarks from the SPLASH-2 [48] and SPEC CPU2006 suites in our simulator. We use SPLASH-2 in our simulation because there is no known report about cross-compiling PARSEC for SESC. We are working with the PARSEC authors on it.

To stress test our power management solution in a many-core microprocessor with a large-scale configuration (i.e., a large number of cores), we conduct simulations using SESC [37] simulator with modifications for per-core DVFS support. The cores are configured based on an Alpha 21264 (EV6) scaled to current process technology. Its main parameters are listed in Table 2. Each core has 4 DVFS levels normalized to the maximum frequency (i.e., 1, 0.9, 0.8 and 0.7). We integrate the MESI cache coherence protocol on the top of the basic Network on Chip (NoC) module of SESC to construct a fully functional NoC so that our simulated many-core microprocessor can provide a coherence guarantee among all the cores. The power of the core part is derived as the combination of the dynamic power reported by Wattch [9] and leakage power estimated by HotLeakage [49]. We use Orion 2.0 [20] to estimate the power consumption of the NoC. To simulate the DVFS actuation overhead, we assume that no instruction is executed during synchronization [18]. Based on recent studies [23], the DVFS actuation overhead can be decreased to tens of ns in the near future. Therefore, we choose to use a $250ns$ transition overhead [5] and 10 subintervals in a control period of $250\mu s$, which leads to a DVFS overhead of up to 1% in each control period. Our scheme allows a higher DVFS overhead by having a longer control period. For example, if we change control period from $250\mu s$ to 5ms, the DVFS overhead is still up to 1% with a $5\mu s$ transition time as in Nehalam processors [1].

6.3 Discussion on Hardware Implementation

Since power management algorithm might need to be implemented on-chip [38] due to the future system integration trend, we discuss the possible on-chip implementation of our solution in this section. The dominant circuit in our solution is the fixed-point division in the partitioner and the fixed-point multiplication in the controller. The division could be implemented as a multiplier with 10% extra circuits [32]. Since the controller and two partitioners are sequential in our algorithm, they could share the same piece of hardware that could be deployed on an on-chip PMU [38]. For the total power budget on the order of hundreds of Watts (a 10-bit representation is sufficiently accurate for power management purpose), we conservatively use a 16-bit fixed-point multiplier. Bitirgen et al. [7] have estimated that a 16-bit fixed-point multiplier yields an



(a) Power trace when cores run at the peak DVFS level.

(b) Power trace when cores run at randomized DVFS levels.

(c) Average measured and estimated power.

(d) Standard deviation of measured and estimated power.

Figure 2: Power estimation accuracy experiments on a 12-core hardware testbed.

area of $0.0057mm^2$ with 65nm process technology. For a typical $200mm^2$ die, the hardware overhead is only 0.003%. To approximate the power consumption by the fixed-point multiplier circuits in the hardware, we assume the power density of IBM POWER6's FPU [13], which is $0.56W/mm^2$ at 100% utilization with nominal voltage and frequency values (1.1V and 4 GHz). The extra power consumed by the fix-point multiplier is only 0.003W. Additionally, we need to add two programmable registers on each core: one is to record the base IPS (IPS_i defined in Section 4.1.2) of the thread scheduled on this core; the other is to record the grouping information. The performance counters recording the instruction count and cache miss count are already available on most modern microprocessors. The delta-sigma modulator consists of one accumulator and one comparison, which is a very light-weight piece of circuit. Furthermore, if the DVFS level is fine-gained enough (*e.g.*, [46]), the modulator could be eliminated. Therefore, our solution is a hardware-efficient solution even for on-chip implementation.

7. EVALUATION

First, we introduce two state-of-the-art baselines. Next, we present our physical testbed and simulation results for many-core architectures. Finally, we provide both the theoretical and experimental analysis of the scalability of our solution.

7.1 Baselines

Our first baseline, referred to as *Priority*, is a heuristic-based power controller for CMPs proposed by Isci et al. [18]. The control scheme of *Priority* is briefly summarized as follows. 1) Every core is assigned a priority. 2) In each control period, if the total power consumption of the chip is lower than the set point, *Priority* chooses the core with the highest priority to increase its DVFS level by one. If the core is already running at its highest DVFS level, the core with the next highest priority will be tried. Alternatively, if the power of the chip is above the set point, *Priority* chooses a core (starting from the lowest priority core) to decrease its DVFS level by one. 3) *Priority* repeats step 2 until the system stops. *Priority* represents a typical centralized solution that adopts a light-weight trial-and-error approach to exploit the combinations of the power states of all the cores. We compare our power management solution against *Priority* to show that heuristic-based, though light-weight, solutions may be insufficient for many-core architectures due to the exponential explosion of the number of global power states.

Our second baseline, referred to as *Steepest Drop*, is a heuristic-based optimization algorithm for many-core architectures proposed by Winter et al. [47]. When the chip power is over the budget, the algorithm selects the application/core pair that would provide the biggest ratio of power reduction to performance loss if the DVFS level was dropped by one step. This process is repeated until the total power is smaller than the budget. Although *Steepest Drop* has been demonstrated to outperform a variety of state-of-the-art algorithms and have a small computation overhead even for a 256-core system, it is based on the assumption that all the cores run independent workloads. This assumption is invalid since it oversimplifies the coupling among the cores hosting the same parallel application. We compare it with our solution to show that it is necessary to consider the couplings among all the cores in power management for improved application performance.

In the following subsections,we refer to our solution as *FreqPar*.

7.2 Estimation Accuracy

In this experiment, we test the accuracy of our online power estimator presented in Section 5. In Figure 2(a), we set all the cores to the peak frequency and run *mix1* on the 12 cores. Figure 2(a) shows that the estimated total chip power can track the measured actual power during the execution time with only small errors. To stress test our estimation scheme, in Figure 2(b), we randomly modulate the DVFS levels of all the cores. Figure 2(b) shows our scheme can track the total power with a reasonable accuracy. In Figures 2(c) and (d), we plot the average and standard deviation of the measured and estimated power respectively, under different frequency levels and benchmark mixes. On our physical testbed, there are five DVFS levels: 0.8GHz, 1.0GHz, 1.3GHz, 1.5GHz, and 1.9GHz. For the estimated average power, the average difference between esti-

(a) Total frequency and power trace: *Priority*.

(b) Total frequency and power trace: *Steepest Drop*.

(c) Total frequency and power trace: *FreqPar*.

(d) Control accuracy under different benchmarks and budgets.

Figure 3: Power control accuracy comparison. In (a)-(c), the frequencies are relative to the peak of a selected core. In (d), the power values are relative to the peak power in each test case.

mated and measured power is within 1 Watt in most cases, with the worst case being *mix5* at 1.9GHz with a 3.6W difference. For the standard deviation, the average difference between estimated and measured power is within 2 Watt in most cases, with the worst case being *mix4* at 0.8GHz with a standard deviation difference of 3.2W.

7.3 Testbed Results

In this subsection, we compare our solution with the two baselines on our physical testbed, in terms of power control accuracy and application performance.

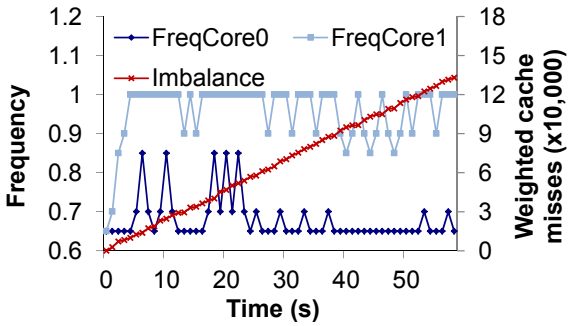
7.3.1 Power Control Accuracy

In this subsection, we first perform a case study to investigate different power management algorithms to show that the algorithms based on the feedback control theory can achieve better power control accuracy. Then, we present the average power control results of *FreqPar* and the baselines under different power budgets.

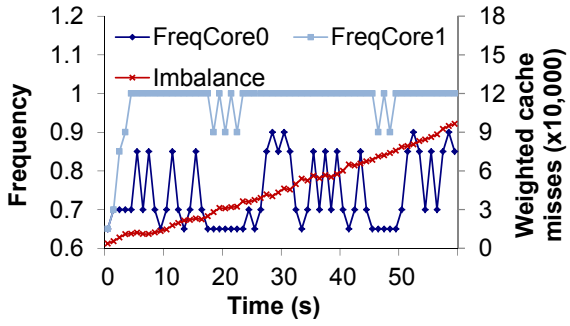
In Figures 3(a)-(c), we schedule 12 copies of *lbm* from SPEC CPU2006 suite on the 12 cores and all algorithms start with all cores set to the lowest frequency level, which is the default setting policy of Linux if the cores are idle. Since the power is lower than the set point (80W for all the policies in this test) at the beginning, all the policies try to raise the DVFS levels of the cores to improve performance. In Figure 3(a) *Priority* responds by increasing the DVFS level of one core at a time, until the power is higher than the set point in the 20^{th} control periods. Then it directs the DVFS level of a core up/down around the set point. There are two parts of this simple algorithm that can be improved. The first one is that a large number of cores will lead to a very long settling time because *Priority* only steps up/down one level of a core in each control period. The second one is that *Priority* always oscillates between two adjacent DVFS levels of a core, even in the steady state. As a result, it never settles to the set point. Since *Priority* has steady state errors, it may be undesirable to use *Priority* in a real system because a positive steady state error (i.e., average power is above the set point) may violate the power budget. Lefurgy et al. [24] have identified this issue and addressed it by having a safety margin when the power budget is assigned. To ensure that the safety margin is safe for all the benchmarks, we run the most hungry benchmark (*mix1*) on every set point between 60W to 85W with 1W step to get the maximum positive steady-state errors, which is 2.42W. By applying such a safety margin when assigning the power budget, we ensure that the average power is always within the power budget. We refer to the *Priority* policy with a safety margin as *Improved Priority*. Note that *Improved Priority* is actually not feasible in practice because it is difficult to have such a priori knowledge about the safety margin before actually running the workload with all the possible power set points. In the following experiments, we use *Improved Priority* as a baseline that can achieve the best possible performance in an ad hoc way and yet does not violate the power constraint.

In order to address the long settling time, *Steepest Drop* [47] has been proposed to explore multiple steps in each control period, based on an analytical model of the power consumption and performance contribution of a core. Figure 3(b) plots the typical run of *Steepest Drop*. *Steepest Drop* takes advantage of the direction of its model, going directly to the estimated optimal DVFS level each time. This policy successfully addresses the long settling time issue in *Priority*. However, even in the ideal case, the power managed under *Steepest Drop* is always lower than the set point because the algorithm terminates the search until the estimated power is just lower than the budget, resulting in an unnecessarily lower total running frequency.

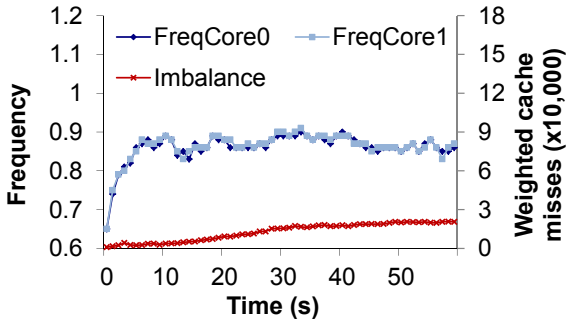
Figure 3(c) shows that *FreqPar* can precisely control the power of the chip by receiving a desired DVFS level from the controller, and then using the DVFS modulator to generate a series of supported DVFS levels on a finer timescale to approximate the desired level. One may think that *Priority* could be improved by also using a series of DVFS levels for each core. However, *Priority* would still have the same steady-state error because, without a desired DVFS level precisely determined based on control theory, *Priority*



(a) Frequency allocation trace: *Improved Priority*.



(b) Frequency allocation trace: *Steepest Drop*.



(c) Frequency allocation trace: *FreqPar*.

Figure 4: Group-level (thread criticality-aware) frequency quota (i.e., sum of normalized DVFS levels) allocation traces of *FreqPar* and the baselines.

can only oscillate between two DVFS levels of a core. Compared with *Steepest Drop* in Figure 3(b), *FreqPar* on average runs at a higher frequency (9.9 relative to the peak of one core, compared to 9.4 for *Steepest Drop*), because the precise control policy of *FreqPar* can use all the available power budget.

Figure 3(d) plots the average power with standard deviations (as error bars) for *FreqPar* and the baselines. In all the tests, we run the application mixes to the end. The power readings are relative to the peak power in each test case. As discussed previously, *FreqPar* can precisely achieve the desired power budgets while the other two methods waste the budgets.

7.3.2 Application Performance

In this subsection, we first provide two case studies to discuss the underlying reasons that *FreqPar* has better performance at both the group and chip levels. Then, we present the average performance of *FreqPar* and the baselines under different power budgets.

Figures 4(a)-(c) show the reason that *FreqPar* can outperform the baselines for parallel workloads. In this experiment, we run two threads of *streamcluster* (PARSEC) on $Core0$ and $Core1$ and leave the other cores unused with a 45W total chip power budget. Under the management of *Steepest Drop*, the algorithm always favors the cores hosting a high raw IPC, which is not necessarily the critical thread that needs more frequency quota within the parallel workload at runtime. Considering the spinning lock, in which the thread has a very high IPC without real progress, a policy that simply favors the high IPC will make the imbalance worse [2]. Bhattacharjee et al. [5] have identified that the weighted cache misses are positively correlated with criticality and proposed to take weighted cache misses as the imbalance indicator. In Figures 4(a) and (b), we observe that the imbalanced frequency allocation policies like *Priority* and *Steepest Drop* increase the weighted cache miss difference between the two cores within a short period of time (60s in this test). The cross-marked curves in Figure 4 are the aggregated weighted cache misses. In contrast, with *FreqPar*, Figure 4(c) shows that the frequency quota is fairly shared by the two cores and always shifted to the cores with a higher criticality within the working group, which dramatically reduces the imbalance between the two threads. As a comparison, we also test the case of evenly dividing frequency quota between the two cores in our initial design as *Even*. Our results show that by dynamically allocating the frequency to the critical thread, *FreqPar* is better than *Even*. The average absolute weighted cache misses differences between the cores in this test are 7843, 4437, 1703, and 739 for *Priority*, *Steepest Drop*, *Even*, and *FreqPar*, respectively.

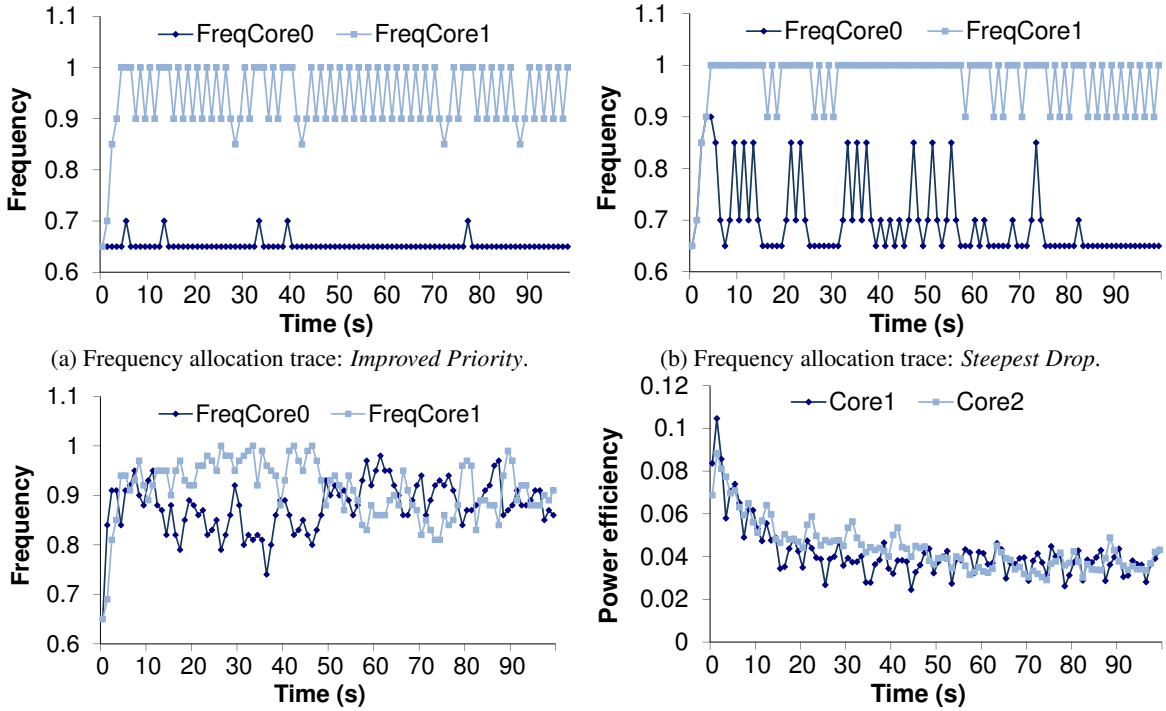
Figure 5 illustrates the reason why *FreqPar* can outperform the baselines in terms of FS (Fair Speedup) at the chip level. In this experiment, we run one copy of *milc* (SPEC CPU2006) on $Core0$ and one copy of *zeusmp* (SPEC CPU2006) on $Core1$. We leave the other cores unused. Figures 5(a)-(c) show the frequencies of the two cores under different policies with a 45W chip power budget. *Steepest Drop* always favors the core hosting high raw IPC because that core generally has a higher IPC/Watt gradient, hence sacrificing fairness. In Figure 5(b), $Core1$ has advantage because it has a higher IPC by nature than $Core0$. Therefore, $Core0$ is always placed into a low frequency level except when $Core1$ has reached its peak frequency and power consumption is still lower than the budget. *Improved Priority* in Figure 5(a) behaves in a similar way because it has a preset static priority. In contrast, with *FreqPar*, Figures 5(c) and (d) show that the frequency quota is always shifted to the core with a higher relative power efficiency. Both cores have the opportunity to get their fair share of the frequency quota under the power budget. *FreqPar* outperforms the baselines because *FreqPar* uses relative power efficiency as the frequency quota partitioning criterion to achieve optimized performance with fairness consideration. In *FreqPar*, $rips_i(k)$ (defined in Section 4.1.2) captures the relative application progress and eliminates the instruction count inflation caused by the natural characteristics of the applications. In contrast, the fairness-blind baselines favor some applications too much over other co-scheduled applications, resulting in degraded overall performance. Based on $rips_i(k)$, *FreqPar* allocates more frequency quota to high-efficiency applications to optimize overall performance with fairness consideration.

Figure 6 plots the overall performance comparison among different power budgets and benchmarks in terms of FS. Due to the reasons discussed in the previous case studies, we observe that *FreqPar* outperforms the *Improved Priority* and *Steepest Drop* by 17% and 11% on average, respectively.

7.4 Simulation Results

In this subsection, we compare *FreqPar* with the two baselines in our simulation environment.

In our simulator, for each number of core configuration, we run the constructed benchmarks (Table 1) with a power budget of 75%



(a) Frequency allocation trace: *Improved Priority*.

(b) Frequency allocation trace: *Steepest Drop*.

(c) Frequency allocation trace: *FreqPar*.

(d) Power efficiency trace: *FreqPar*.

Figure 5: Chip-level (power efficiency-aware) frequency quota (i.e., sum of normalized DVFS levels) allocation traces and power efficiency of *FreqPar* and the baselines.

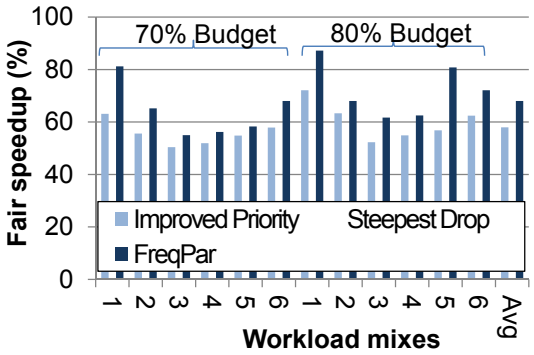


Figure 6: Overall performance comparison between *FreqPar* the baselines on a 12-core hardware testbed.

of the peak power of each benchmark mix and report the average. We fastforward 1 billion instructions and simulate 3, 5, and 8 billions for 32, 64, and 128 core configuration, respectively. Figure 7 shows the power control accuracy and performance of each policy. Power readings are relative to the power budget. The FS numbers are relative to the FS of *FreqPar* in each test case. We observe that *FreqPar* can precisely control the power of the chip to the desired set point while both the two baselines waste certain amount of available power budget. Note that when we calculate the average power, we skipped the initial phase and compute the steady phase for all the policies. Figure 7 also shows that *FreqPar* has better performance than the baselines for each core configuration. Furthermore, *FreqPar*'s power and performance improvements over the baselines increase with the number of cores. This is because *Improved Priority* spends a long time under the budget because of its long settling time. For *Steepest Drop*, with more cores, the raw-IPC-directed optimization without fairness and criticality considerations worsens the imbalance among the cores.

7.5 Discussion on Algorithm Complexity and Scalability

The computational complexity in terms of the algorithm execution time determines the algorithm scalability. We first analyze the computational complexity of studied algorithms and then provide experimental results. Suppose there is an M-core system that supports per-core DVFS and the number of available DVFS levels is L. *Priority* adjusts the DVFS level of a core by one step at a time. It is a $O(1)$ algorithm. However, its long convergence time makes it unlikely to be implemented in many-core systems. In the worst case, if the current power state is that every core is running at the peak state, *Priority* takes L^M control periods to reach the lowest power set point. During that time, undesirable overheating may occur. *Steepest Drop* checks all the cores to determine the optimal one step action based on an analytical model in one iteration, and it iterates multiple times until the estimated power of the analytical model is less than the budget. In the worst case, the search goes up-/down $L*M$ times. Therefore, it is a $O(L*M^2)$ algorithm. Winter et al. [47] propose a special data structure to reduce the implementation complexity to $O(L*Mlg(M))$. Since IBM has implemented a digital phase-locked loop [41], which achieves a near-continuous set of output frequencies without skipping processing cycles, an algorithm with an execution time linearly increasing with the number of available DVFS levels might be unfavorable. In contrast, even in the worst case, *FreqPar* proceeds through the cores twice (one for the chip-level and one for the group-level). Therefore, it is a $O(M)$ algorithm independent from L. Furthermore, if the available DVFS levels are fine enough, the delta-sigma modulator could be eliminated, resulting in an even simpler design.

We now conduct experiments to measure the execution times of the three algorithms on our testbed. We examine the algorithm execution time as the number of cores increases in Figure 8. In the experiment, we invoke the investigated algorithms 500 times on our

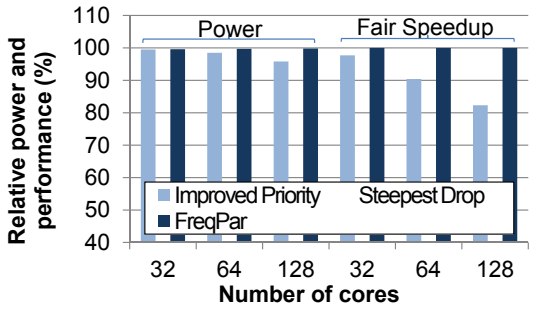


Figure 7: Power and performance comparison in simulations under different number of cores.

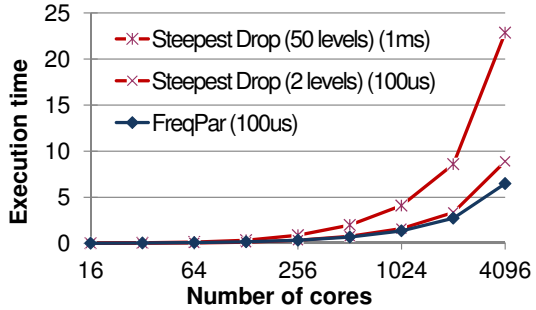


Figure 8: Execution time experiments show that *FreqPar* is more scalable than *Steepest Drop*.

physical testbed with random generated power and DVFS levels as the inputs and present the average execution time.

In Figure 8, the execution time scaling trends of the algorithms confirm our theoretical analysis. As a $O(L * Mlg(M))$ algorithm, *Steepest Drop* achieves a modest execution time scaling with the number of cores when the number of available DVFS levels is small (e.g., 2 levels). However, when the number of DVFS levels is large, the scalability is questionable. Suppose that the dynamic power adaptation range of one core is from 0.5 to 1 relative to the peak, with a 1% step size (achievable for POWER7 [41]), the average execution time of *Steepest Drop* uses more than 20ms on a 4096-core system. In contrast, *FreqPar* is a scalable $O(M)$ algorithm independent from the number of DVFS levels. Even for managing a 4096-core system, the average execution time of *FreqPar* is only $650\mu s$. The key reason that *FreqPar* outperforms *Steepest Drop* (a classic down-hill search algorithm with optimized implementation) on scalability is that, with the delta-sigma modulator, *FreqPar* uses a numerical computation to replace the discrete search used by *Steepest Drop* in the decision making part.

8. RELATED WORK

Power dissipation has been one of the major design concerns for computing systems. Much prior work has focused on minimizing the power consumption within a specified performance guarantee. For example, Li et al. [25] propose a solution called thrifty barrier that places the faster cores into a lower power mode at the barriers (i.e., joint point) while waiting for the slower cores so that power can be saved. Liu et al. [26] use per-core DVFS to slow down the faster cores, such that both the idle time due to waiting and power consumption are reduced. Cai et al. [10] extend [26] by adding meeting points within the execution of the parallel loops and solve the same problem at a finer granularity. However, all of the solutions cannot provide any explicit guarantees for the power consumption to stay below a desired budget though the performance is guaranteed to some extent. Our work is different in that we focus on a different, but equally important, problem, i.e., power capping to avoid power overload or thermal violations and prevent over-provision of cooling, packaging, and power supply capacities at the processor design time.

Some work has been performed to manage peak power or temperature for CMPs. Intel Foxton technology [38] has successfully controlled the power and temperature of a microprocessor using chip-wide DVFS. Isci et al. [18] propose a closed-loop algorithm called Priority and a prediction-based algorithm called MaxBIPS to limit the power of a CMP. Wang et al. [45] also apply advanced control theory to develop a power control algorithm for improved CMP performance. However, the application of these solutions on many-core systems is prohibited either by the exponential explosion of the number of possible global power management states in many-core architectures, *e.g.*, [38, 18], or by the high control delay and computation overhead due to centralized decision making, *e.g.*, [45]. As a result, none of them are scalable to the large number of cores in many-core architectures.

A recent study by Winter et al. [47] presents a global power management algorithm called *Steepest Drop* for many-core systems with a light overhead. Sartori et al. [39] discuss to use hierarchical structure to cap the power of many-core systems. Another related piece of work by Mishra et al. [30, 29] uses absolute BIPS to allocate the chip power budget to each power island and performs per-island power control. However, these solutions assume the independence of workloads among all the cores. Therefore, it may impair the coupling of workloads among all the cores and result in degraded system performance. In contrast, our highly scalable solution can dynamically shift the power budget among the groups of cores that host different applications based on power efficiency, and then further among all the cores in the same group that host the coupled threads from the same application based on thread criticality.

9. CONCLUSION

The majority of existing solutions on power control of multi-core architectures does not scale well for many-core architectures. More importantly, those solutions cannot effectively allocate power based on thread criticality to accelerate multi-threaded parallel applications, which are expected to be the primary workloads of many-core architectures. In this paper, we have presented a highly scalable power control solution that is specifically designed to handle realistic workloads, *i.e.*, a mixed group of single-threaded and multi-threaded applications. Our solution features a three-layer design. First, we adopt control theory to precisely control the power of the entire chip to its chip-level budget by adjusting the aggregated frequency of all the cores on the chip. Second, we dynamically group cores running the same applications and then partition the chip-level aggregated frequency quota among different groups for optimized overall processor performance. Finally, we partition the group-level aggregated frequency quota among the cores on each group based on the measured thread criticality for shorter application completion time. As a result, our solution can optimize the processor performance while precisely limiting the chip-level power consumption below the desired budget. Empirical results on a physical testbed show that our control solution can provide precise power control, as well as 17% and 11% better application performance than two state-of-the-art solutions, on average, for mixed PARSEC and SPEC benchmarks. Furthermore, our extensive simulation results for 32, 64, and 128 cores, as well as overhead analysis for up to 4,096 cores, demonstrate that our solution is highly scalable to many-core architectures.

10. ACKNOWLEDGEMENTS

This work was supported, in part, by NSF under Grants CCF-1017336, CNS-0915959, and CNS-0845390 (CAREER Award),

and by ONR under Grant N00014-09-1-0750. We would also like to thank the anonymous reviewers for their valuable comments.

11. REFERENCES

[1] Intel turbo boost technology. http://www.intel.com/technology/turboboost/, 2007.

[2] A. R. Alameldeen and D. A. Wood. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4), 2006.

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

[4] S. Bell et al. Tile64 processor: A 64-core SoC with mesh interconnect. In *ISSCC*, 2008.

[5] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *ISCA*, 2009.

[6] C. Bienia et al. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, 2008.

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

[8] S. Borkar. Thousand core chips: a technology perspective. In *DAC*, 2007.

[9] D. Brooks et al. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, 2000.

[10] Q. Cai et al. Meeting points: using thread criticality to adapt multicore hardware to parallel regions. In *PACT*, 2008.

[11] T. Calin et al. Built-in current sensor for IDDQ testing in deep submicron CMOS. In *VLSITS*, 1999.

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

[13] B. Curran et al. 4GHz+ low-latency fixed-point and binary floating-point execution units for the POWER6 processor. In *ISSCC*, 2006.

[14] D. Economou et al. Full-system power analysis and modeling for server environments. In *MOBS*, 2006.

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

[16] E. Grochowski et al. Energy per instruction trends in intel microprocessors. Technical report, Intel Microarchitecture Research Lab, 2006.

[17] J. Howard et al. A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *ISSCC*, 2010.

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

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

[20] A. Kahng et al. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *DATE*, 2009.

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

[22] S. Kim et al. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *PACT*, 2004.

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

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

[25] J. Li et al. The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors. In *HPCA*, 2004.

[26] C. Liu et al. Exploiting barriers to optimize power consumption of CMPs. In *IPDPS*, 2005.

[27] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *ISCA*, 2007.

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

[29] A. K. Mishra et al. CPM for CMPs: Coordinated power management for chip-multiprocessors. In *SC*, 2010.

[30] A. K. Mishra et al. Poster: Coordinated power management of voltage islands in CMPs. In *SIGMETRICS*, 2010.

[31] D. B. Noonburg and J. P. Shen. Theoretical modeling of superscalar processor performance. In *MICRO*, 1994.

[32] S. Oberman. Floating point division and square root algorithms and implementation in the AMD-K7TM microprocessor. In *CA*, 1999.

[33] A. Phansalkar et al. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. *SIGARCH Comput. Archit. News*, 35(2), 2007.

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

[35] R. Raghavendra et al. No Power Struggle: Coordinated Multi-level Power Management for the Data Center. In *ASPLOS*, 2008.

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

[37] J. Renau et al. SESC simulator, 2005. http://sesc.sourceforge.net.

[38] M. Rich et al. Power and temperature control on a 90-nm Itanium family processor. *IEEE Journal of Solid-State Circuits*, 41(1), 2006.

[39] J. Sartori and R. Kumar. Distributed peak power management for many-core architectures. In *DATE*, 2009.

[40] K. Skadron et al. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1), 2004.

[41] J. Tierno et al. A DPLL-based per core variable frequency clock generator for an eight-core POWER7 x2122 microprocessor. In *VLSIC*, 2010.

[42] D. N. Truong et al. A 167-processor computational platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4), 2009.

[43] S. R. Vangal et al. An 80-tile sub-100-w teraflops processor in 65-nm CMOS. *IEEE Solid-state circuits*, 43(1), 2008.

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

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

[46] M. Ware et al. Architecting for power management: The IBM POWER7 approach. In *HPCA*, 2010.

[47] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *PACT*, 2010.

[48] S. C. Woo et al. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA*, 1995.

[49] Y. Zhang et al. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical report, University of Virginia, 2003.