DPPC: Dynamic Power Partitioning and Capping in Chip Multiprocessors

Kai Ma[1,2], Xiaorui Wang[1,2], and Yefu Wang[2]
[1]*The Ohio State University* [2]*University of Tennessee, Knoxville*

***Abstract*—A key challenge in chip multiprocessor (CMP) design is to optimize the performance within a power budget limited by the CMP's cooling, packaging, and power supply capacities. Most existing solutions rely solely on DVFS to adapt the power consumption of CPU cores, without coordinating with the last-level on-chip (e.g., L2) cache. This paper proposes DPPC, a chip-level power partitioning and capping strategy that can dynamically and *explicitly* partition the chip-level power budget among different CPU cores and the shared last-level cache in a CMP based on the workload characteristics measured online. DPPC features a novel performance-power model and an online model estimator to quantitatively estimate the performance contributed by each core and the cache with their respective local power budgets. DPPC then re-partitions the chip-level power budget among them for optimized CMP performance. The partitioned local power budgets for the CPU cores and cache are precisely enforced by power capping algorithms designed rigorously based on feedback control theory. Our experimental results demonstrate that DPPC achieves better CMP performance, within a given power budget, than several state-of-the-art power capping solutions.**

I. Introduction

Chip multiprocessors (CMPs) are becoming an important platform in the development of computer systems. However, power consumption continues to be a major design constraint for the further throughput improvement of CMPs, as high power consumption may result in high die temperatures that can affect the reliability and performance of a CMP [3][18]. The power problem is also being exacerbated by the increasing levels of core integration and transistor density. Therefore, a key challenge in the CMP design is to optimize the performance of a CMP within a power budget limited by the CMP's cooling, packaging, and power supply capacities, such that the performance delivered per watt can be maximized.

While various power control algorithms have recently been proposed for CMPs [3][16][18], most solutions mainly rely on dynamic voltage and frequency scaling (DVFS) to adapt the power consumption of CPU cores, without coordinating with the last-level on-chip (e.g., L2) cache. Unfortunately, lowering the frequencies of CPU cores may severely degrade the performance of CPU-intensive workloads, while at the same time some on-chip L2 cache ways could become rarely accessed and so can be dynamically put into low-power modes, allowing more power to be shifted to the CPU cores for improved performance. Likewise, for memory-intensive workloads, power can be dynamically shifted from CPU cores to cache for performance optimization. Therefore, power control for CMPs should be conducted in a coordinated manner for optimized performance.

This work was supported, in part, by NSF under CCF-1143605, CCF-1017336, and CNS-0915959, and by ONR under N00014-09-1-0750.

Coordinated power control is challenging because the optimal power budget partitioning among different components in a CMP for achieving the best performance is unknown *a prior* and may change for different workloads. Even for the same workload, the optimal power partitioning can change during different execution phases. Thus, a systematic approach is needed to dynamically estimate the performance/power ratio of each component and then allocate more power to the components that can gain greater performance improvements for the CMP. In addition, after the chip-level power budget is partitioned into a set of local power budgets for the components, it is preferable that each component can precisely consume the power allocated to it, with guaranteed control accuracy and system stability despite possible errors in adopted power models, in order to enforce the optimized partitioning.

In this paper, we propose DPPC, a dynamic power partitioning and control strategy that shifts power among different CPU cores and the shared L2 cache in a CMP, while maintaining the total power of the CMP to stay within a given budget. The key novelty of DPPC lies in its explicit power budget partitioning and control based on the workload characteristics measured online. Specifically, this paper makes the following new contributions:

- We demonstrate that power, as a first-class resource, can be *explicitly* and dynamically partitioned among different CPU cores and shared L2 cache to achieve optimized CMP performance.
- We dynamically measure the performance contributed by each heterogeneous component (CPU core or cache) and then use optimization techniques to quantitatively partition the chip-level power budget among them. Compared with the commonly used trial-and-error and greedy search solutions, this systematic approach can lead to better power budget partitioning decisions.
- We design power control algorithms based on feedback control theory for CPU cores and cache, respectively, with analytic assurance of control accuracy and system stability, despite possible modeling errors. This rigorous design methodology is in sharp contrast to heuristic-based solutions that heavily rely on extensive manual tuning.

The rest of the paper is organized as follows. Section II introduces the overall architecture of our power control strategy. Section III describes dynamic power budget partitioning. Section IV introduces the system modeling, design, and analysis of our power controllers. Section V provides the implementation details. Section VI presents our evaluation results. Section VII highlights the distinction of our work by discussing the related work. Section VIII concludes the paper.

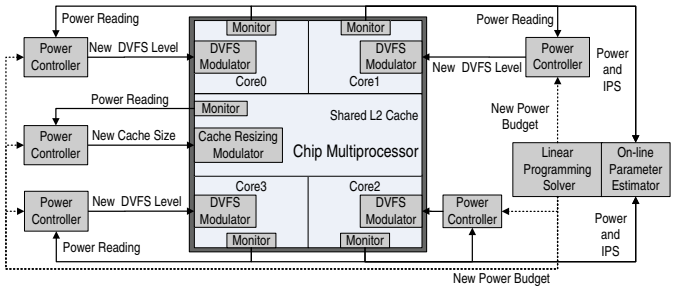

Fig. 1. DPPC power partitioning and control architecture (implemented in service processor firmware) for a 4-core CMP with shared L2 cache (for illustration and not a physical chip floorplan). A power monitor estimates the power consumption of a core or the cache based on the approaches in [4][7].

II. Power Partitioning and Control Architecture

In this section, we present a high-level description of the DPPC power control strategy, which dynamically partitions the power budget among different CPU cores and the shared L2 cache in a CMP based on measured workload characteristics and then precisely controls the power of each component to its respective local budget. IBM POWER7 [19] has demonstrated that sophisticated power control schemes can be implemented in service processor firmware to interact with various power, temperature, and performance (*e.g.*, IPS) sensors. In that way, the power and computation overheads of the firmware do not directly contribute to the power consumption and temperature of the main CMP. The proposed DPPC power control architecture is implemented in the same way.

Since performance is directly related to power, we explicitly partition and shift power among different components in a CMP to allocate more power to components that contribute more to the CMP performance. In this work, we treat each core in a CMP as a component and the entire shared L2 cache as another component, because both CPU cores and cache are major power consumers in a CMP. For example, L2 cache constitutes a large portion of transistor budget on the die and so consumes a significant amount of leakage power [15], [2]. We choose to use shared L2 cache because it is the dominant configuration in commercial CMPs, but our solution can also be used to manage private L2 cache or shared L3 cache. In addition, both CPU cores and cache allow effective power adaptation with per-core DVFS and dynamic cache way resizing, respectively. It is important to note that our solution does *not* depend on per-core DVFS for power control. For CMPs that do not support per-core DVFS, our solution can still adopt chip-wide DVFS and dynamic cache resizing to flexibly shift power between all the cores and the shared cache. We use per-core DVFS in this paper mainly because all our baselines are using per-core DVFS. In addition, our solution is not limited to cache way resizing and instead can use other cache resizing granularities, such as cache lines or blocks, with only minor changes and slightly different hardware overhead.

As shown in Figure 1, the key components in the power partitioning loop include an online parameter estimator, a simplified Linear Programming (LP) solver (with acceptable overhead as discussed in Section III-C), and a power monitor to estimate the power consumption of each component. The partitioning loop is invoked periodically. In each period, the estimator in the service processor firmware receives the performance measures from each core, and the power readings from the monitors on the components. The estimator then updates the model parameters based on these measured values and sends the parameters to the simplified LP solver (in firmware or FPGA). The LP solver computes the optimal power partitioning to get the local power budgets for the components. The solver then sends the local power budgets as new set points to the power controllers of the components. Although the power consumption of L2 cache and an individual core cannot be directly measured in today's CMPs, it has been shown that on-chip programmable event counters, together with chip-level power monitor or on-chip sensors (*e.g.*, as in ItaniumII [12]), can provide power estimations that are precise enough for control purpose [4][7]. Note that the overhead of a simplified LP solver can be *bounded* if we allow some degradation in solution quality, as discussed in Section III-C. In our experiments, we show that DPPC still outperforms the baselines even with an LP solver whose run-time overhead is smaller than 1.6%.

The key components in the power control loop of each component include a power controller, a power monitor, and a DVFS modulator (or a cache resizing modulator). Each power control loop is also invoked periodically and its period is chosen based on a trade-off between the actuation overhead and the system settling time. The following steps are invoked at the end of every control period: 1) The power monitor sends the power consumption of the component in the last control period to the corresponding controller. The controllers can be implemented either in the service processor firmware or on chip due to their negligible computational overheads, as discussed in Section IV. The power consumption is the *controlled variable* of the control loop. 2) The power controller computes a new DVFS level or a new cache size for the component and then sends the level or size to the corresponding modulator. The new DVFS level or cache size is the *manipulated variable* of the control loop. 3) If the component is a CPU core, the DVFS modulator changes the DVFS level of the core accordingly; if it is the shared L2 cache, the cache resizing modulator changes the number of active cache ways accordingly.

III. Dynamic Power Partitioning

In this section, we first introduce the performance-power model and how to periodically estimate its parameters online. We then apply simplified linear programming to optimize power budget partitioning. Finally, we discuss the complexity of the simplified LP solver for a trade-off between solution quality and overhead.

A. Performance-Power Model and Online Parameter Estimation

As discussed before, our power partitioning decision is made based on an empirical model that estimates the CMP performance (*i.e.*, IPS) contributed by each core and the shared cache. In general, when a CMP is running under a tight power budget, the performance of the CMP is a monotonically increasing function of its power consumption. Unfortunately, the performance-power model is highly complex and hard to build in an analytic way, because such a model can often be

nonlinear and time-varying and depends on various factors, such as the workload, the specific CMP design, and the system configurations. In order to allow linear optimization, we approximate the nonlinear performance-power model using a piecewise linear function and then update the time-varying parameters online. Therefore, the performance of a CMP in the k^{th} period can be empirically modeled as: $perf(k) = d(k)p(k)$, where $p(k)$ is the power consumption of the CMP and $d(k)$ is a time-varying parameter that may change for different ranges of $p(k)$, workloads, or CMP chips.

Since the CMP performance is contributed by the collaborative work from each CPU core and the L2 cache, increasing the power of a CPU core or the L2 cache within certain ranges can lead to improved performance. As observed in our experiments, the CMP performance improvement as a function of the power budget increase for either a CPU core or the L2 cache can be approximated as a linear or piecewise linear function. Therefore, although the CMP performance (*i.e.*, IPS) contributions from each core and the cache are not independent from each other in nature (especially for multi-threaded workloads), to facilitate power partitioning, the CMP performance in short term can be approximated as the sum of the performance contributions of all the components. A similar linear approximation method has been adopted in [16] for CMP throughput modeling. As discussed before, the performance contribution of each component can then be approximated as a piecewise linear function of its power consumption. Hence, in a CMP with N cores and shared L2 cache, the performance of the CMP can be *empirically* modeled as:

$$perf(k) = \sum_{i=0}^{N} d_i(k)p_i(k) = \mathbf{d(k)p(k)} \quad (1)$$

where $p_i(k), 0 \leq i \leq N-1$, is the power consumption of Core i while $p_N(k)$ is the power of the entire shared L2 cache. d_i is the performance contribution parameter of Core i if $0 \leq i \leq N-1$ or the cache if $i = N$. $\mathbf{p(k)} = [\ p_0(k) \quad p_1(k) \quad \cdots \quad p_N(k)\ ]^T$ is the power vector in the k^{th} period. $\mathbf{d(k)} = [\ d_0(k) \quad d_1(k) \quad \cdots \quad d_N(k)\ ]$ is the parameter vector that needs to be periodically estimated and updated online.

There are three important things to note. First, the performance model (1) is only an empirical model that captures the *transient* performance-power behavior of each component in a CMP. It is designed to just facilitate dynamical power partitioning and *not* intended to be a theoretical model with well-established physical meanings. Second, **d(k)** in model (1) must be dynamically updated because the performance contribution of each component in the CMP varies significantly for different workloads and CMP chips, as well as for different ranges of $p(k)$ and different execution phases of the same workload. Third, for multi-threaded workloads, we adopt the technique proposed in [6] to detect instructions caused by spin locks and then deduct them from the IPS computation. We then use the real IPS values (that indicate the real progress of the programs running on the CMP) in the online estimation of **d(k)** and the LP solver.

In this work, we use the recursive identification technique [18] to estimate the values of **d(k)** in the k^{th} period based on the value of **d(k-1)** and a set of observations in the previous periods. Specifically, we use a Recursive Least Square (RLS) estimator with directional forgetting to estimate and update the parameter vector **d(k)** in the system performance model (1). This recursive algorithm estimates the values of **d(k)** based on the currently and previously measured values with the assumption that the model does not suddenly change significantly in one period. In each period, the RLS estimator calculates the performance contribution parameter vector **d(k)** based on the following equations:

$$\mathbf{V(k)} = (\mathbf{V}^{-1}(k-1) + (1 + (\lambda - 1)\frac{\mathbf{p}^T(k)\mathbf{V}(k-1)\mathbf{p}(k)}{(\mathbf{p}^T(k)\mathbf{p}(k))^2})\mathbf{p}(k)\mathbf{p}^T(k))^{-1} \quad (2)$$

$$\mathbf{d(k)} = \mathbf{d(k\text{-}1)} + \frac{e(k)\mathbf{p}^T(k)\mathbf{V(k\text{-}1)}}{\lambda + \mathbf{p}^T(k)\mathbf{V(k\text{-}1)}\mathbf{p}(k)} \quad (3)$$

where $e(k) = perf(k) - \mathbf{d(k\text{-}1)p(k\text{-}1)}$ is the estimation error. **V(k)** is the covariance matrix and λ is the constant forgetting factor with $0 \leq \lambda \leq 1$. A smaller λ allows the estimator to forget the history faster. In our experiment, we use $\lambda = 0.6$ as a tradeoff. Equation (2) is used to update the correlation between the overall performance and the power of each component based on the history input and output variations. Equation (3) is used to update the estimated contribution parameters based on the correlation variations. The following iteration is invoked in every partitioning period: 1) The RLS estimator records the power vector **p(k)** and the CMP performance $perf(k)$. 2) The estimator calculates **d(k)** according to Equations (2) and (3). 3) The estimator updates **d(k)** in the system performance-power model for the simplified LP solver.

B. Simplified LP for Power Partitioning

Linear programming has been widely used in computer architecture research for solving linear optimization problem (*e.g.*, [16]). We now formulate the power budget partitioning problem as a linear programming problem. Specifically, the problem we want to optimize is: Given $N+1$ on-chip components (N cores and the shared L2 cache) and the chip-level power budget P, in the k^{th} period, we need to find the best power budget partitioning, $\{p_i(k)|0 \leq i \leq N\}$, for the cores and cache that maximizes the overall CMP performance subject to two constraints: 1) the total chip power is equal to or smaller than P, and 2) the power of each component is between its upper bound $PU_i(k)$ and lower bound $PL_i(k)$. Based on the CMP performance model (1), our optimization objective function is:

$$\max(\sum_{i=0}^{N} d_i(k)p_i(k)) \quad (4)$$

subject to the following constraints:

$$\sum_{i=0}^{N} p_i(k) \leq P \quad (5)$$

$$PL_i(k) \leq p_i(k) \leq PU_i(k), 0 \leq i \leq N \quad (6)$$

At the end of every period, the simplified LP solver computes the optimal power budget partitioning point based on the contribution parameters **d(k)** estimated periodically online. Note that $p_i(k)$ in the objective function is the power budget allocated to component i, while it represents the real power consumption in the performance model (1). Therefore, it is

important to ensure that the optimal partitioning is precisely enforced, such that the real power consumption of a component is exactly the power budget allocated to it. In Section IV, we introduce how our power controllers can precisely control the power of a component to its local budget. Note that DPPC can be extended for thermal management by adding temperature constraints as functions of power consumption, similar to [18].

C. Discussion on the Overhead of LP Solver

In our prototype system, we implement the simplified LP solver based on the `lipsol` function in Matlab, which is an optimized algorithm based on a primal-dual interior-point method [13]. This algorithm has a worst-case complexity of $O(M^{3.5})$ [13], where M is the number of inequalities in the problem. In our problem, M is equal to the number of cores plus one (*i.e.*, cache). In practice, `lipsol`'s performance is often much better than this worst-case analysis. More importantly, `lipsol` allows one to specify an upper bound on how many iterations to execute and then searches for the best solution within the bound. While finding the optimal solution can easily take more than 10 iterations, we test two iteration bounds, 1 and 3, in our experiments for significantly reduced overhead. Our results show that the function with even a bound of one can outperform the baselines. It has also been shown that one iteration for a 20-entry problem (*e.g.*, a 19-core CMP) takes approximately 27,700 cycles [21]. Therefore, the function with a bound of 3 results in a worst-case execution time of $4.1ms$ on a 20MHz service processor (*e.g.*, Renesas H8 used in IBM BladeCenter HS20 for power management [5]), which is much shorter than the partitioning period of $50ms$ used in our experiments. Therefore, our solution is suitable for CMPs with tens of cores. We plan to develop more scalable power control solutions for many-core CMPs with hundreds of cores in our future work. For the purpose of comparison, we also implement the LP solver based on the `linprog` function in Matlab, which finds the global optimal solution without any simplifications. Hereinafter, we refer to DPPC without simplifications as DPPC-O and DPPC with the iteration bound being 1 and 3 as DPPC-1 and DPPC-3, respectively.

In order to further reduce the computational overhead, we do not activate the LP solver and the online model estimator in periods when the performance-power model only has slight changes. Specifically, we compare the estimated IPS and the measured IPS in each period. If the difference is smaller than an *update threshold*, *e.g.*, 25% of the measured IPS, the estimator and LP solver are not invoked. Our results show that the LP solver is only invoked 3 (out of 5) times, on average, every 250ms (*i.e.*, 5 partitioning periods) in our experiments. Therefore, the average run-time overhead of DPPC-3 and DPPC-1 on a 20MHz service processor for a *19-core* CMP is $(4.1ms \times 3)/250ms = 4.9\%$ and $4.1ms/250ms = 1.6\%$, respectively. Such an overhead is acceptable to most systems.

IV. Power Controllers

We now present the power controllers designed for a CPU core. The power controller of the cache is designed similarly.

We first introduce some notation. T is the control period. $p_i(k)$, $0 \leq i \leq N-1$, is the power consumption of Core i in the k^{th} control period. $f_i(k)$ is the DVFS level of Core i in the k^{th} control period. $\Delta f_i(k) = f_i(k+1) - f_i(k)$. P_i is the power budget of Core i, which is decided by the LP solver as $p_i(k)$ in (4). Since the period of the LP solver is much longer than the control period T, P_i can be treated as a constant within one period of the LP solver. The control goal is to guarantee that $p_i(k)$ converges to P_i within a given settling time.

In order to have an effective controller design, we now model the dynamics of the controlled system, namely the relation between the controlled variable $\{p_i(k), 1 \leq i \leq N-1\}$ and the manipulated variable $\{f_i(k), 1 \leq i \leq N-1\}$. Theoretically, it is well-known that DVFS can allow cubic reduction in power density relative to performance loss for each core in a CMP. However, a cubic power model leads to a high complexity for controller design and a large runtime overhead. On the other hand, in practice, real CMP processors usually only provide a limited DVFS range. Within the small range, previous studies [16][18] have shown that the relationship between power and DVFS level can be approximated with a linear or piecewise linear function. Therefore, the power consumption of a core is modeled as:

$$p_i(k) = a_i f_i(k) + c_i \tag{7}$$

where a_i and c_i are the generalized parameters that may vary for different cores and different applications. In our experiments, we perform a set of experiments with randomly selected applications and use curve fitting to decide these parameters. We find that the a_i changes significantly when the workload changes, while c_i remains almost the same with only negligible variations. Therefore, the dynamic model of the system as a difference equation is:

$$p_i(k) = p_i(k-1) + a_i \Delta f_i(k-1) \tag{8}$$

We apply PID (Proportional-Integral-Derivative) control theory to design a P (Proportional) controller for desired specifications, such as stability, zero steady-state error, and short settling time. An integral (I) term is not used in the controller function because the actuator includes an integration step (as part of the delta-sigma modulator introduced in Section V), such that zero steady-state error can be achieved without an integral part. A derivative (D) term is not included because having it unnecessarily might amplify the noise in the power readings. The time-domain form of our P controller is:

$$f_i(k) = f_i(k-1) + K_p(P_i - p_i(k)) \tag{9}$$

where K_p is the control parameter, whose value can be analytically chosen using the standard Root Locus method. We use $K_p = 0.045$ in this paper based on the system model (8) with a nominal parameter $a_i = 10.59$, which is the average value of the model parameters resulting from a set of SPEC CPU2006 workloads tested in our experiments. Therefore, the computational overhead of the controller is just one multiplication and one addition, which is sufficiently small for an architectural-level implementation.

V. System Implementation

To evaluate our solution, we conduct simulations using SESC. Four CPU cores are configured based on Alpha 21264 (EV6) (65nm) with a peak frequency of 3GHz. We configure each core to have 4 linearly scaled DVFS levels (1, 0.866,

0.733, 0.6 as normalized to the peak level). In our simulation environment, we use Wattch (with CACTI) to calculate the dynamic power and Hotleakage to calculate the leakage power with a typical temperature setting of 80°C. The L2 cache is configured to be 8MB (8-way set associative). The main memory latency is 300 cycles. The L2 cache replacement algorithm is LRU (Least Recently Used). The L1 i-cache and d-cache are both 32KB (2-way, 64 byte block size) with a 3-cycle hit latency.

The configured CMP in our simulation only supports 4 discrete DVFS levels. However, the new DVFS level periodically received from the power controller can be any value, which may not be exactly one of the DVFS levels supported by the CMP. Therefore, a DVFS modulator is needed to approximate the desired level with a series of supported DVFS levels. For example, to approximate 2.8GHz during one control period, the modulator would output the sequence, 2.7, 2.7, 3, 2.7, 2.7, 3, etc, on a smaller timescale. To implement this, we use the first-order delta-sigma modulator [5] to generate the sequence in each control period. A tradeoff is that when the sequence has more numbers during one control period, the approximation is more accurate but the actuation overhead becomes higher. In this work, the overhead of DVFS scaling is conservatively set to $10\mu s$ by assuming a common off-chip switching regulator [14]. We choose to use 20 DVFS scaling subintervals in a control period of 5ms to approximate the desired DVFS level, which leads to a subinterval of $250\mu s$. We also conservatively assume that no instruction can be issued during the $10\mu s$ of DVFS scaling transaction time. As a result, the actuation overhead is 4% ($10\mu s$/$250\mu s$) even in the worst case when the DVFS level needs to be changed in every subinterval. This overhead is acceptable to most CMPs.

We control cache power by dynamically switching selected cache ways between high- and low-power modes [9]. We adopt the Gate-Vdd technology [11] to implement the low-power mode, but note that other low-power cache technologies, such as drowsy caches, can also be used in our control solution. We adjust the cache address mapping and replacement based on the number of high-power mode ways to ensure that cache addressing is always valid despite cache resizing. Dirty data write back process is also simulated when we put cache ways into the low-power mode. In addition, we conservatively assume that when performing cache resizing, the whole tag part is always turned on; only the data array is resizable [8].

As discussed in Section II, in a real system, our algorithms can be implemented in service processor firmware, similar as other power control solutions (*e.g.*, Intel ItaniumII [12] and IBM POWER7 [19]).

VI. Evaluation

In this section, we present our evaluation results. We fast-forward 2 billion instructions before each simulation test and stop the simulation when any of the cores finishes executing 1 billion instructions. The period of power control for core and cache is 5ms and the period of power partitioning is 50ms.

We compare DPPC with three state-of-the-art baselines: MPC, IMPC, and Greedy. MPC [18] is a recent study that relies solely on per-core DVFS to control the power of a CMP. Greedy [8] tries to simply search, in a greedy way, for a combination of core DVFS levels that would lead to better CMP performance. IMPC is an improved version of MPC to partition the power budget between cache and CPU cores in a trial-and-error way. In IMPC, the power of all the cores is controlled by MPC, while the cache power is controlled by the controller used by DPPC. In an example period, IMPC may try to shift 2W from the CPU cores to the cache. If the shifting results in an increased IPS, IMPC continues to shift power at a 2W step in this direction; otherwise, IMPC reverses the direction to shift 2W from the cache to the CPU cores. Note that the 2W step is selected based on experiments such that IMPC has the best performance with the best trade-off between system stability and the response speed to workload variations. With the trial-and-error method, IMPC is sensitive to the variations in the performance-power model and may easily end up with a suboptimal partitioning point. IMPC is similar to the MPC-EW algorithm proposed in [17].

Note that our baselines, MPC and Greedy, outperform the MaxBIPS algorithm proposed in [3]. MPC also outperforms Priority [3] (and the similar Intel Turbo Boost technology), while both MaxBIPS and Priority outperform chip-wide DVFS solutions, such as Intel's Foxton technology [12]. We test three configurations of DPPC. DPPC-O is the optimal solution without any simplifications. DPPC-1 and DPPC-3 have the iteration bound as 1 and 3, respectively, and their run-time overheads for the 4-core CMP should be much lower than 1.6% and 4.9%, which are estimated for a 19-core CMP.

A. Power Control Capability

In this section, we use SPEC CPU2006 to test whether DPPC and the baselines can effectively limit the power consumption of the CMP. Every core is configured to run a different benchmark. To construct mixed workloads, we consider both workload redundancy [10] and cache characteristics of different workloads. Specifically, mix1 includes *mcf*, *libquantum*, *dealII*, and *soplex*; mix2 includes *libquantum*, *dealII*, *soplex*, and *lbm*; mix3 includes *dealll*, *soplex*, *lbm*, and *namd*; mix4 includes *soplex*, *lbm*, *namd*, and *bzip2*; mix5 includes *lbm*, *namd*, *bzip2*, and *gobmk*; mix6 includes *namd*, *bzip2*, *gobmk*, and *milc*.

We conduct a set of experiments to test the CMP power consumption under the six schemes for different power budgets and workload mixes. Figure 2 shows that all the five control-based schemes can always precisely control power to the desired budget, while Greedy often stays below the budget, leading to unnecessarily degraded CMP performance, as we will discuss in Section VI-B. Each value in Figure 2 is the averaged power in the steady state of each scheme, which is calculated by eliminating the transient power values at the beginning of the run. In summary, all the schemes successfully limit the peak CMP power.

B. Performance Comparison

The goal of power control is to achieve optimized CMP performance within a given power budget. Therefore, we now run the same SPEC CPU2006 benchmark mixes to compare the three DPPC solutions with the three baselines in terms of system performance.

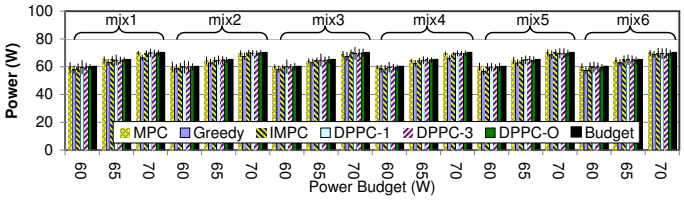

Fig. 2. DPPC and baselines all successfully limit the CMP power consumption for different SPEC CPU2006 benchmark mixes.

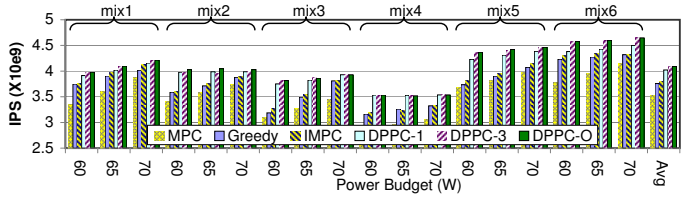

Fig. 3. DPPC achieves better CMP performance (IPS) than the baselines for different SPEC CPU2006 benchmark mixes.

Figure 3 shows that the IPS values of all the six schemes under different workloads generally increase when the power budget increases from 60W to 80W. The three configurations of DPPC all outperform the three baselines at every point. In particular, DPPC-O's performance improvements over MPC, IMPC, and Greedy are up to 23.54%, 16.91%, and 19.73% (all for mix3 at 60W). DPPC-1's improvements over MPC, IMPC, and Greedy are up to 21.43%, 14.92%, and 17.70% (all for mix3 at 60W). The average improvements are as follows. First, by coordinating per-core DVFS and dynamic cache resizing, DPPC-O and DPPC-1 achieve 15.85% and 13.84% better average performance than MPC. Second, with online model estimation and global power partitioning optimization, DPPC-O and DPPC-1 outperform IMPC by 7.74% and 7.05% on average. Finally, by integrating optimization with feedback control to precisely control power to the budget, DPPC-O and DPPC-1 achieve 8.94% and 5.87% better performance than Greedy on average. Interestingly, Greedy outperforms MPC even though it does not use up the power budget. This demonstrates that it is important to dynamically resize L2 cache because L2 cache constitutes a large portion of transistor budget on the die and so consumes a significant amount of leakage power. For example, in Intel's dual-core 64-bit Xeon processor, more than 80% of the transistors are used for cache [15]. Among the three DPPC solutions, DPPC-O outperforms DPPC-1 and DPPC-3 by 1.77% and 0.24%, on average, at the cost of higher overhead.

VII. Related Work

Extensive prior work has been proposed to manage power or temperature for CMPs. However, most studies do not explicitly limit the power consumption of a CMP to stay within a budget determined by the CMP's cooling, packaging, and power supply capacities. In this paper, we try to optimize the performance of a CMP within such a given power budget. Intel's Foxton technology [12] has successfully controlled the power and temperature of a processor using chip-wide DVFS. Some recent studies [3][18][20][7] have also proposed various power control algorithms that rely solely on per-core DVFS to adapt the power consumption of CPU cores. In this paper, we coordinate DVFS and dynamic cache resizing to shift power among different CPU cores and L2 cache in a CMP, while maintaining the power of the CMP to stay within a given budget. Several coordinated power management strategies have been previously proposed [1][8][16]. In contrast, our solution integrates optimization with feedback control to precisely control power for better CMP performance. Furthermore, our work uses online model estimation to allow explicit power shifting.

VIII. Conclusions

This paper presented DPPC, a chip-level power partitioning and control strategy that dynamically and explicitly shifts power among different CPU cores and the shared L2 cache in a CMP. The key novelty of DPPC lies in its explicit power budget partitioning and control based on the workload power characteristics measured online. Our power control algorithms for a CPU core and cache are designed rigorously based on control theory for precise power control despite considerable modeling errors. The overhead of DPPC has been analyzed and demonstrated to be acceptable to most existing CMPs. Our experimental results demonstrate that DPPC achieves better CMP performance, within a given power budget, than several state-of-the-art power control solutions. We plan to design more scalable power control solutions for many-core CMPs in our future work.

References

[1] W. Felter *et al.*, "A performance-conserving approach for reducing peak power consumption in server systems," in *ICS*, 2005.
[2] H. Homayoun and A. Veidenbaum, "Reducing leakage power in peripheral circuits of L2 caches," in *ICCD*, 2007.
[3] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *MICRO*, 2006.
[4] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *MICRO*, 2003.
[5] C. Lefurgy *et al.*, "Server-level power control," in *ICAC*, 2007.
[6] T. Li, A. R. Lebeck, and D. J. Sorin, "Spin detection hardware for improved management of multithreaded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 6, pp. 508–521, 2006.
[7] K. Ma *et al.*, "Scalable power control for many-core architectures running multi-threaded applications," in *ISCA*, 2011.
[8] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *PACT*, 2008.
[9] Y. Meng *et al.*, "Exploring the limits of leakage power reduction in caches," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 3, 2005.
[10] A. Phansalkar *et al.*, "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite," in *ISCA*, 2007.
[11] M. Powell *et al.*, "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *ISLPED*, 2000.
[12] M. Rich *et al.*, "Power and temperature control on a 90-nm itanium family processor," *IEEE J. of Solid-State Circuits*, vol. 41, no. 1, 2006.
[13] P. Rong and M. Pedram, "Battery-aware power management based on markovian decision processes," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1337–1349, 2006.
[14] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, 2004.
[15] S. Tam *et al.*, "A 65nm 95w dual-core multi-threaded Xeon processor with L3 cache," in *ASSCC*, 2006.
[16] R. Teodorescu *et al.*, "Variation-aware application scheduling and power management for chip multiprocessors," in *ISCA*, 2008.
[17] X. Wang, K. Ma, and Y. Wang, "Adaptive power control with online model estimation for chip multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, 2011.
[18] Y. Wang *et al.*, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *ISCA*, 2009.
[19] M. Ware *et al.*, "Architecting for power management: The IBM POWER7 approach," in *HPCA*, 2010.
[20] J. Winter *et al.*, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *PACT*, 2010.
[21] C.-H. Wu *et al.*, "FPGA implementation of the interior-point algorithm with applications to collision detection," in *17th IEEE Symposium on Field Programmable Custom Computing Machines*, 2009.