Scalable Global Power Management Policy Based on Combinatorial Optimization for Multiprocessors

GUNG-YU PAN, National Chiao Tung University
JED YANG, University of Minnesota
JING-YANG JOU, National Central University and National Chiao Tung University
BO-CHENG CHARLES LAI, National Chiao Tung University

Multiprocessors have become the main architecture trend in modern systems due to the superior performance; nevertheless, the power consumption remains a critical challenge. Global power management (GPM) aims at dynamically finding the power state combination that satisfies the power budget constraint while maximizing the overall performance (or vice versa). Due to the increasing number of cores in a multiprocessor system, the scalability of GPM policies has become critical when searching satisfactory state combinations within acceptable time. This article proposes a highly scalable policy based on combinatorial optimization with theoretical proofs, whereas previous works take exhaustive search or heuristic methods. The proposed policy first applies an optimum algorithm to construct a state combination table in pseudo–polynomial time using dynamic programming. Then, the state combination is assigned to cores with minimum transition cost in linear time by mapping to the network flow problem. Simulation results show that the proposed policy achieves better system performance for any given power budget when compared to the state-of-the-art heuristic. Furthermore, the proposed policy demonstrates its prominent scalability with 125 times faster policy runtime for 512 cores.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design—*Real-Time Systems and Embedded Systems*; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design (CAD)

General Terms: Algorithms, Design, Management, Performance

Additional Key Words and Phrases: Combinatorial optimization, DVFS, multiprocessor systems

ACM Reference Format:
Gung-Yu Pan, Jed Yang, Jing-Yang Jou, and Bo-Cheng Charles Lai. 2015. Scalable global power management policy based on combinatorial optimization for multiprocessors. ACM Trans. Embed. Comput. Syst. 14, 4, Article 70 (December 2015), 24 pages.
DOI: http://dx.doi.org/10.1145/2811404

1. INTRODUCTION

Multiprocessor systems have become mainstream in both general-purpose and embedded computers to break the ILP wall; however, power consumption has remained the bottleneck for digital designs over the past decade [Pedram 1996]. Among many low-power techniques, system-level power management is widely applied, including dynamic voltage and frequency scaling (DVFS) [Jha 2001; Yao et al. 1995], dynamic

This work is supported by the National Science Council under grant NSC101-2221-E-008-137-MY3.
Authors' addresses: G.-Y. Pan (corresponding author) and B.-C. C. Lai, Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, 1001 University Road, Hsinchu City, Taiwan; emails: astro.sea@gmail.com, bclai@mail.nctu.edu.tw; J. Yang, School of Mathematics, University of Minnesota, Minneapolis, MN 55455; email: jedyang@ucla.edu; J.-Y. Jou, Department of Electrical Engineering, National Central University, 300 Jhongda Road, Taoyuan County, Taiwan; email: jingyangjou@gmail.com.

DOI: http://dx.doi.org/10.1145/2811404

power management (DPM) [Benini et al. 2000], cache resizing [Wang et al. 2011; Ma et al. 2011b], and task migration [Ge et al. 2010]. In this article, we focus on DVFS, where the supply voltage and clock frequency are adjusted dynamically to reduce power.

Many modern systems support DVFS, including Enhanced Intel SpeedStep Technology (EIST) [Intel 2013], AMD PowerNow! [AMD 2013], ARM Intelligent Energy Controller (IEC) [ARM 2005], and MIPS Cluster Power Controller (CPC) [Knoth 2009]. These technologies provide the power management interfaces for specific systems so that *power managers* can be implemented in software.

System-level power management is abstracted as a power state decision among several given modes, referring to the p-state (performance level) defined in the Advanced Configuration and Power Interface (ACPI) [ACPI 2011] for DVFS. Architectural characteristics can be measured beforehand, including the supply voltage and the operating frequency of each state, and the transition overhead induced by mode switching.

In designing power managers for multiprocessor systems, the *scalability* issues need to be extraordinarily considered due to the increasing number of cores within a system [Held et al. 2006]. Previous works show that global fine-grained power control can achieve more power saving [Sharkey et al. 2007; Kim et al. 2008], but the management overhead increases accordingly. Power management overhead includes voltage (frequency) switching delay and the time to run the policy itself; the former is linearly proportional to the difference in voltage [Isci et al. 2006], and the latter is proportional to the number of cores [Winter et al. 2010]. As the supply voltage scales down, the switching delay is shrinking (19.5μs in Isci et al. [2006] and 10μs in Ma et al. [2011b]), whereas the policy runtime (longer than 100μs for 256 cores in Winter et al. [2010]) now dominates the power management overhead because the number of cores scales up. Therefore, designing low-complexity (in terms of total cores) policies is critical in per-core global power management (GPM); our goal is to provide scalable algorithms while the optimality of the policy is guaranteed by mathematical proofs.

1.1. Global Power Management

Specifically, the GPM problem for multiprocessors is formulated in Isci et al. [2006]: given per-core available DVFS, determine the power state of each core to maximize throughput under the power budget constraint. Isci et al. [2006] assume that the power manager is separated from thread scheduling, and the cores are filled with multiprogrammed contexts and kept busy. They propose a policy called *MaxBIPS* to solve the problem; cubic and linear scaling with respect to the supply voltage are adopted to characterize the dynamic power and operating frequency, respectively. MaxBIPS tries to maximize the summation of the frequency of each core while maintaining the summation of the power of each core below the budget. It also takes a priority in the order of the CPU boundedness of the contexts. Their experimental results show that the estimation metrics are accurate in both throughput (2% to 4% error rate) and power (0.1% to 0.3% error rate). This model has demonstrated its accuracy, and the performance of MaxBIPS is shown better than other policies [Isci et al. 2006]; therefore, the same model is widely used by other following works.

Since MaxBIPS exhaustively searches the power state combinations, several heuristics are proposed to enhance scalability. In Teodorescu and Torrellas [2008], linear approximation between power and performance is used, and the combination is determined by the linear programming algorithm LinOpt. Greedy neighboring search is used for chip multiprocessors in Meng et al. [2008], and the steepest drop (SD) algorithm for many-core architectures considering process variation is proposed in Winter et al. [2010]. Winter et al. [2010] also show that thread scheduling and GPM can be considered separately. In Ma et al. [2011b], the problem is solved by simplified linear

Table I. Comparisons of GPM Policies

Policy Description	Year	Time Complexity	Optimal Transition Cost
MaxBIPS [Isci et al. 2006] Exhaustive search	2006	$O(nm^n)$	Yes
LinOpt [Teodorescu and Torrellas 2008] Linear programming with linear approximation	2008	$O(n^4)$	No
Greedy [Meng et al. 2008] Greedy search for neighboring state	2008	$O(mn)$	No
Steepest drop [Winter et al. 2010] Greedy search for neighboring state	2010	$O(mn \lg n)$	No
DPPC [Ma et al. 2011b] Linear programming with empirical models	2011	$O(n^{3.5})$	No
Ours (general) Combinatorial optimizations	2013	$O(m^3 \lg (n/m))$	Yes
Ours (special) Combinatorial optimizations	2013	$O(lg(mn) + m)$	Yes

programming with empirical models. All of these policies targeting the GPM problem are compared in the following section.

1.2. Our Contributions

Like previous works, we inherit the same problem formulation of GPM for homogeneous multiprocessor systems [Isci et al. 2006] and focus on designing the policy for power state decisions. We make two main contributions in this article. First, our policy is highly scalable for multiprocessor systems. Second, the combinatorial algorithms behind the policy are optimum. Both scalability and optimality are proved theoretically and verified by cycle-accurate simulations using commercial benchmarks [Henning 2006].

The comparisons of different GPM policies are listed in Table I. To provide fair comparisons, we remove the optimization considerations other than throughput maximization with power budget (e.g., process variation in Teodorescu and Torrellas [2008] and [Winter et al. 2010]) and the power management techniques other than per-core DVFS (e.g., cache resizing in Meng et al. [2008] and Ma et al. [2011b]). Besides, the third column is the time complexity of online policies, whereas MaxBIPS and our policy prebuild some tables offline. Since the number of performance levels m is limited according to ACPI [2011] while the number of cores n may scale to a large number, the policy scalability is dominated by n. In short, our policy achieves the same optimality as MaxBIPS and the best scalability among all policies. Furthermore, the transition costs are taken into consideration in our policy as MaxBIPS. In addition to the general cases, our policy is able to further speed up in some common conditions; the details are discussed in later sections.

In this article, we provide a table-lookup method to obtain the state combination having maximum performance under the given power budget. Based on combinatorial optimization (CO), we propose an optimum offline algorithm for table construction in pseudo–polynomial time. We then provide another linear-time (in the number of cores) algorithm to assign the next state for each core with minimum transition cost. Simulation results show that the proposed algorithms provide better power-performance trade-off than the previous state-of-the-art method. Moreover, our policy achieves up to 125 times speedup for 512 cores.

The remainder of the article is organized as follows. Section 2 illustrates the block diagram and formally defines the problems. Sections 3 and 4 provide the proposed algorithms for the problems. Section 5 shows the simulation results, including

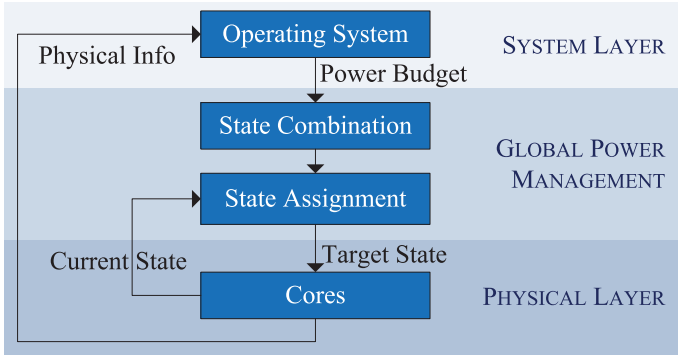

Fig. 1. The system block diagram. The operating system decides the power budget according to the measured physical information. Then the global power manager determines the target power states without exceeding the budget.

performance and runtime comparisons. Finally, Section 6 gives the survey on related works, and Section 7 summarizes the article.

2. PROBLEM FORMULATION

2.1. System Model

We follow the closed-loop system model in Isci et al. [2006], as depicted in Figure 1. The GPM layer is in between the operating system and the physical cores. The power states of all of the cores are managed by the global power manager, which can be implemented as system software, firmware, or hardware.

For each power management epoch, the target states of cores are determined by the global power manager according to the power budget and the current states. The state decision process is separated into two phases: first we determine the state combination and then assign the states to the cores.

The workload characteristics are unknown in advanced, and the physical information (power/performance) measured by monitors is sent back to the operating system. The operating system takes the measured actual power and the current power states to estimate the future power consumption. Note that the power budget can be determined by controller(s) or user-defined value(s) where the power manager is independent of the value(s); we take the simple controller in Isci et al. [2006] to generate the power budget value for simulation in Section 5.

2.2. Notations

The notations of this article are listed in Table II, with the first sector for the architecture characteristics, the second sector for the constraint(s) made by the operating system, and the last sector for the algorithms behind the global power manager. These notations are used for analysis, whereas the real values of the architecture characteristics (m, n, f, p, C) are obtained from commercial systems for simulation in Section 5.

There are n cores in the homogeneous multiprocessor system and m DVFS states in each core. Fine-grained power control is available, so each core can be in any state.

The power states are numbered 0 to $m-1$, with corresponding operating frequency $f = (f_0, \ldots, f_{m-1})$, $f \in Z_+^m$ and power consumption $p = (p_0, \ldots, p_{m-1})$, $p \in R_+^m$; f_i and p_i denote the frequency and the power of the ith state, respectively, while the supply voltage is implicitly defined. Without loss of generality, we assume $f_i \geq f_j, p_i \geq p_j$ for $0 \leq i < j < m$. Note that f is a positive integer vector representing the relative performance levels; this is common in system design, such as the ARM IEC [ARM 2005].

Table II. Notations

Notation	Type	Meaning
m	Z_+	Number of power states
n	Z_+	Number of cores
f	Z_+^m	Operating frequency
p	R_+^m	Power consumption
C	$R_+^{m\times m}$	Transition cost
P	R_+	Aggregate power budget
F	Z_+	(Transformed) aggregate frequency constraint
w	Z_+^m	Current state combination
x	Z_+^m	Next state combination
Y	$Z_+^{m\times m}$	Transition combination

The positive real vector p represents the relative power consumption of different states as well.

The transition cost is denoted by an $m \times m$ matrix C, where $c_{ij} \in R_+$ is the state switching cost from the ith state to the jth state; it can be defined as power, delay, or any mixture thereof. In the simulation section, we use the transition delay as the cost, which is adapted from Isci et al. [2006].

The power budget P is determined by the operating system within the range $np_{m-1} \leq P \leq np_0$. Note that P is also a relative value as the power consumption vector p. The aggregate power (summation of power over all cores) should not exceed P. The aggregate frequency constraint F is only an intermediate value of our algorithm in Section 3.

For homogeneous systems, the characteristics of power states are the same for all the cores. In other words, the *combination* instead of the permutation of core states is important in this article. Therefore, the dimensions of w, x and Y are in terms of the number of states m instead of the number of cores n. The three variables are formally defined in the following sections.

2.3. Statement of Problems

Based on the preceding system model, we focus on the design of algorithms for the two phases for GPM.

The former phase is called `State Combination`: given the system characteristics (m, n, f, p) and aggregate power budget P, determine the state combination such that the steady-state performance is maximized. The state combination problem is formally formulated as mathematical equations in Section 3.

The latter phase is called `State Assignment`: given the system characteristics (m, n, C), the current states of cores, and the state combination, assign the target states to the cores such that the transition cost is minimized. The state assignment problem is formally formulated as mathematical equations in Section 4.

Note that the transition delay of DVFS (in the order of 10μs in Isci et al. [2006]) is much shorter than the period of power management (0.5ms in Isci et al. [2006] and 10ms in Winter et al. [2010]); therefore, we can optimize the steady-state performance first and then minimize the transition cost.

3. STATE COMBINATION

In this section, the `State Combination` problem is formulated in linear equation and then solved using CO techniques through several equivalent transformations. Note that the power budget P is the only dynamic input while the system characteristics (m, n, f, p) are statically given. Thus, we can speed up the runtime by constructing a

state combination table offline with the aggregate power as index and then look up the table for the state combination online. The table construction process can be further accelerated for specific power modes.

3.1. Problem Transformations

According to the system model of GPM [Isci et al. 2006], performance maximization is done by maximizing the aggregate frequency. The summation of the operating frequency over all cores is maximized while the summation of the power over all cores is below the budget.

Since we focus on homogeneous systems, we are only interested in the aggregate number of cores in each state instead of the separate state of each core. Therefore, we define an m-dimensional vector $x = (x_0, \ldots, x_{m-1})$ for the next state combination to be determined. In other words, x_i is the number of cores that should be in the ith state. Then the `State Combination` problem is formulated as

$$\text{maximize} \quad \sum_{i=0}^{m-1} f_i x_i \tag{1}$$

$$\text{subject to} \quad \sum_{i=0}^{m-1} p_i x_i \leq P \tag{2}$$

$$\sum_{i=0}^{m-1} x_i = n. \tag{3}$$

The performance maximization is expressed in (1), and the aggregate power constraint is stated in (2) (3). Note that (3) is a necessary constraint to ensure that the summation of the cores in each state is exactly the total number of cores; otherwise, the solutions may not be valid.

The preceding problem formulation is NP-hard as explained in Meng et al. [2008], so solving it using exhaustive search is not scalable. Nevertheless, we can reduce the time complexity to pseudopolynomial by reformulating the `State Combination` problem in another direction: given the system characteristics (m, n, f, p) and the aggregate frequency constraint F, determine the state combination such that the aggregate power consumption is minimized, where the aggregate frequency constraint is within the range $nf_{m-1} \leq F \leq nf_0$. This problem is formulated as

$$\text{minimize} \quad \sum_{i=0}^{m-1} p_i x_i \tag{4}$$

$$\text{subject to} \quad \sum_{i=0}^{m-1} f_i x_i \geq F \tag{5}$$

$$\sum_{i=0}^{m-1} x_i = n. \tag{6}$$

The formulation is very similar, but the constraint and objective function are swapped. Because the optimal aggregate power is a nondecreasing function of the optimal aggregate frequency (and vice versa), the entries in the look-up table remain in the same order regardless of the index in P or F. In other words, suppose that $\check{x}$ is an optimum solution in (4) through (6) with aggregate frequency $\check{F} \geq F$ and minimum

aggregate power $\check{P}$; we cannot find another solution $\check{x}'$ with aggregate frequency $\check{F}' > \check{F} \geq F$ but aggregate power $\check{P}' < \check{P}$ for (1) through (3); otherwise, the solution $\check{x}$ is not optimum for (4) through (6). Therefore, we can take advantage of the integer character of f and reduce the time complexity using some combinatorial algorithms. We will explain the method to look up the table using P (without F) in the third section.

Note that the problem formulated in (4) through (6) is different from the bounded knapsack problem (BKP), because the number of items is limited by an aggregate amount ($\sum_{i=0}^{m-1} x_i = n$) instead of a distinct amount for each item ($x_i \leq c_i$ for $i = 0, 1, \ldots, m-1$, where c is an m-dimensional vector). Therefore, this problem cannot be solved using existing algorithms for BKP directly. We need to make some transformations to this problem first and then solve it with other existing algorithms.

Definition 3.1.

$$\hat{f}_i = f_0 - f_i, \hat{p}_i = p_0 - p_i \text{ for } i = 0, 1, \ldots, m-1 \tag{7}$$

$$\hat{F} = n \cdot f_0 - F \tag{8}$$

We can deem $\hat{f}_i$ and $\hat{p}_i$ as the frequency degradation and the power saving for the ith state, respectively, and $\hat{F}$ as the total frequency degradation constraint. By substituting (7) (8) into (4) through (6), we can obtain the maximization problem with inverted inequalities:

$$\text{maximize} \quad \sum_{i=0}^{m-1} \hat{p}_i x_i \tag{9}$$

$$\text{subject to} \quad \sum_{i=0}^{m-1} \hat{f}_i x_i \leq \hat{F} \tag{10}$$

$$\sum_{i=0}^{m-1} x_i = n. \tag{11}$$

Then we define the offset of frequency and power ($\bar{f}, \bar{p}$), and create the shifted frequency degradation f_i', the shifted power saving p_i', and the shifted total frequency degradation F'.

Definition 3.2.

$$\bar{f} = n \cdot f_0 + 1, \qquad \bar{p} = n \cdot p_0, \tag{12}$$

$$f_i' = \hat{f}_i + \bar{f}, \qquad p_i' = \hat{p}_i + \bar{p}, \text{ for } i = 0, 1, \ldots, m-1 \tag{13}$$

$$F' = \hat{F} + n \cdot \bar{f} \tag{14}$$

Since $\bar{f}$ and $\bar{p}$ are constants, substituting $\hat{p}_i$ with p_i' in (9) does not change the objective function, and adding $n\bar{f}$ to the both sides of inequality (10) yields $\sum_{i=0}^{m-1} \hat{f}_i x_i + n\bar{f} = \sum_{i=0}^{m-1} \hat{f}_i x_i + \sum_{i=0}^{m-1} \bar{f} x_i = \sum_{i=0}^{m-1} f_i' x_i \leq \hat{F} + n\bar{f} = F'$, which does not change it. Then we can rewrite the problem as

$$\text{maximize} \sum_{i=0}^{m-1} p_i' x_i \tag{15}$$

$$\text{subject to} \sum_{i=0}^{m-1} f_i' x_i \leq F'. \tag{16}$$

Note that (11) is removed, so the preceding problem (15) (16) can be solved using existing algorithms. The remaining problem is the validity proof of the second problem transformation where (11) is embedded in (15) (16). In other words, suppose that the solution to the preceding problem (15) (16) is $\check{x}_i$; we need to prove that the summation of $\check{x}_i$ over the m different states is exactly equal to n.

LEMMA 3.3. $\sum_{i=0}^{m-1} \check{x}_i \leq n$.

PROOF. Assume that $\sum_{i=0}^{m-1} \check{x}_i \geq n+1$, then we substitute (12) through (14) in (16) to obtain $\hat{F} + n^2 f_0 + n \geq \sum_{i=0}^{m-1} \check{x}_i(\hat{f}_i + nf_0 + 1) \geq \sum_{i=0}^{m-1} \check{x}_i(nf_0+1) \geq (n+1)(nf_0+1) = n^2 f_0 + n + nf_0 + 1$. After simplification, we derive $\hat{F} \geq nf_0 + 1$, which contradicts the fact that $\hat{F} \leq nf_0$. As a result, $\sum_{i=0}^{m-1} \check{x}_i \leq n$ is proved by contradiction. □

LEMMA 3.4. $\sum_{i=0}^{m-1} \check{x}_i \geq n$.

PROOF. Suppose that we find an optimal solution with $\sum_{i=0}^{m-1} \check{x}_i \leq n-1$, then we substitute (12) and (13) in (15) to obtain $\sum_{i=0}^{m-1} \check{x}_i(\hat{p}_i + np_0) = np_0 \sum_{i=0}^{m-1} \check{x}_i + \sum_{i=0}^{m-1} \check{x}_i \hat{p}_i \leq n(n-1)p_0 + (n-1)p_0 < n^2 p_0 = n\bar{p}$. Note that the right-hand side is the minimum value for (15) by setting $x_0 = n, x_i = 0$ for $i = 1, 2, \ldots, m-1$, which is feasible in (16) as $f'_0 n = \tilde{f} n \leq F'$. It contradicts the optimal assumption because the summation for this setting is n; because any solution with $\sum_{i=0}^{m-1} \check{x}_i \leq n-1$ is not optimal, optimal solutions must satisfy $\sum_{i=0}^{m-1} \check{x}_i \geq n$. □

THEOREM 3.5. *The solution of the transformed problem (15) (16) is also a solution to the state combination problem (9) through (11).*

PROOF. From Lemmas 1 and 2, any feasible solution satisfies $\sum_{i=0}^{m-1} \check{x}_i \leq n$, and any optimal solution satisfies $\sum_{i=0}^{m-1} \check{x}_i \geq n$; thus, we can obtain (11), which completes the original problem formulation. □

3.2. Table Construction

The preceding formulation (15) (16) is exactly the unbounded knapsack problem (UKP). With integer coefficients, UKP can be solved using dynamic programming. Define $P'_{\tilde{f}}$ as the maximum total power saving subject to the constraint that the total frequency degradation is at most $\tilde{f}$. We can establish the recursive relation

$$P'_{\tilde{f}} = \max\left(P'_{\tilde{f}-1}, \max_{f'_i \leq \tilde{f}} \left(P'_{\tilde{f}-f'_i} + p'_i\right)\right). \tag{17}$$

To have the maximum power saving for each index $\tilde{f}$, one of the states is picked and the corresponding p'_i is added, or $P'_{\tilde{f}} = P'_{\tilde{f}-1}$ if no state is picked. Starting with $P'_0 = 0$, the solution is obtained by increasing the index until F'.

The overall algorithm for table construction is listed in Algorithm 1. Equivalent problem transformations are done in lines 1 through 6. Then the table is constructed by scanning from 0 to $\max(F')$ using dynamic programming in lines 7 through 11. The best state combination, as well as the transformed aggregate power and frequency, are recorded in the table during each step. After constructing the table, we only need to keep the last $n(f_0 - f_{m-1}) + 1$ entries because $nf_{m-1} \leq F \leq nf_0$. Besides, those entries having the same state combination can be merged (since redundant entries are created during dynamic programming). At last, we need to restore the actual values of aggregate power and frequency by subtracting the aggregate offset $n\bar{p} = n^2 p_0$ and $n\bar{f} = n^2 f_0 + n$, respectively.

Table III. General State Combination Table

F	P	x
nf_0	np_0	$(n, 0, \ldots, 0)$
.	.	.
.	.	.
$(n-1)f_0 + f_1$	$(n-1)p_0 + p_1$	$(n-1, 1, 0, \ldots, 0)$
.	.	.
.	.	.
.	.	.
$f_{m-2} + (n-1)f_{m-1}$	$p_{m-2} + (n-1)p_{m-1}$	$(0, \ldots, 1, n-1)$
.	.	.
.	.	.
nf_{m-1}	np_{m-1}	$(0, \ldots, 0, n)$

ALGORITHM 1: Table-Construction

```
1  f̄ ← nf_0 + 1;
2  p̄ ← np_0;
3  for i ← 0 to m − 1 do
4  |   f'_i ← f_0 − f_i + f̄;
5  |   p'_i ← p_0 − p_i + p̄;
6  end
7  P'_0 ← 0;
8  for f̃ ← 1 to n^2 f_0 + n + nf_0 − nf_{m−1} do
9  |   P'_f̃ = max(P'_{f̃−1}, max_{f'_i ≤ f̃} (P'_{f̃−f'_i} + p'_i));
10 |   record best state combination in the table;
11 end
12 truncate the table and restore P from P';
```

The most time-consuming part is the maximization operation in line 9, which compares m times. The loop in lines 8 through 11 executes at most $F' = \hat{F} + n\bar{f} = n\hat{f}_0 - F + n(nf_0 + 1) \leq n(f_0 - f_i + nf_0 + 1) = O(n^2 f_0)$ times. Therefore, the time complexity is $O(mF') = O(mn^2 f_0)$ for offline table construction.

3.3. Table Lookup

An example state combination table is shown in Table III. Since the aggregate power is monotonically increasing as the aggregate frequency from the bottom of the table, we can binary search through the table for the maximum aggregate frequency without exceeding the budget and obtain the optimum state combination.

The table has at most $n(f_0 - f_{m-1}) + 1$ rows. Using binary search, the time complexity is $O(lg(n(f_0 - f_{m-1}) + 1)) = O(lg(nf_0))$ for online table lookup.

Due to the observation that the relative performance levels are represented as fixed point in real systems, we can transform and solve the NP-hard problem in pseudo–polynomial time. Note that we cannot solve the original problem with power budget directly using dynamic programming unless we quantize p into an integer vector.

3.4. Arithmetic Sequence Condition

The preceding sections discuss the algorithm that solves the general state combination problem. In some systems, the given performance levels usually form arithmetic sequences [Meng et al. 2008; Winter et al. 2010; Ma et al. 2011a; Sartori and Kumar 2009]. Under the cases, the optimization algorithm can be further simplified.

The specific situation is called the `Arithmetic Sequence Condition`, where the differences between neighboring performance levels are the same. Besides, the dependency

between power and frequency is superlinear. Then the relationship between neighboring states is mathematically formulated as follows:

$$\begin{cases} f_i - f_{i+1} = f_{i+1} - f_{i+2} \\ p_i - p_{i+1} \geq p_{i+1} - p_{i+2} \end{cases} \quad \text{for } 0 \leq i < m-2. \tag{18}$$

By the transitive law, it is rewritten in general form:

$$\begin{cases} f_i - f_{i+1} = f_j - f_{j+1} \\ p_i - p_{i+1} \geq p_j - p_{j+1} \end{cases} \quad \text{for } 0 \leq i < j < m-2. \tag{19}$$

Under this condition, we find a useful property to simplify the general state combination problem and reduce both table construction and lookup time.

Definition 3.6. The `nonzero-interval` of a state combination x is the maximum distance between any two nonzero columns of x. It is denoted as $\lambda(x) = \max_{0 \leq l \leq r < m}(r-l)$, $x_i = 0$ for $i < l$ or $i > r$.

LEMMA 3.7. *For any state combination $\check{x}$ with nonzero-interval $\lambda(\check{x}) > 1$, corresponding aggregate frequency $\check{F}$, and aggregate power $\check{P}$, we can find another state combination $\check{x}'$ such that its nonzero-interval is $\lambda(\check{x}') \leq \lambda(\check{x}) - 1$, with the same aggregate frequency $\check{F}' = \check{F}$ and nonincreasing aggregate power $\check{P}' \leq \check{P}$ under the Arithmetic Sequence Condition.*

PROOF. Let $\check{x}_l$ and $\check{x}_r$ denote the left-most and right-most nonzero column of a state combination $\check{x}$, respectively. Let $\check{d} = \min(\check{x}_l, \check{x}_r)$; we can create $\check{x}'$ by setting $\check{x}'_l = \check{x}_l - \check{d}$, $\check{x}'_{l+1} = \check{x}_{l+1} + \check{d}$, $\check{x}'_r = \check{x}_r - \check{d}$, $\check{x}'_{r-1} = \check{x}_{r-1} + \check{d}$, and $\check{x}'_i = \check{x}_i$ for the rest columns $l+1 < i < r-1$. At least one of $\check{x}'_l$, and $\check{x}'_r$ becomes zero and the nonzero-interval decreases by one or two. Note that $\check{F}' = \check{F} - \check{d} \cdot (\check{f}_l + \check{f}_r - \check{f}_{l+1} - \check{f}_{r-1}) = \check{F}'$ and $\check{P}' = \check{P} - \check{d} \cdot (\check{p}_l + \check{p}_r - \check{p}_{l+1} - \check{p}_{r-1}) \leq \check{P}$ because of the Arithmetic Sequence Condition (19).

When the state combination $\check{x}$ is optimum, then the created $\check{x}'$ is also optimum with $\check{x}'$ has $\check{F}' = \check{F}$ and $\check{P}' = \check{P}$; they are referred to as an `equivalence` of each other.

Using the preceding property, we can shrink the solution space by finding the optimum `congregate` state combination x under any given F.

Definition 3.8. A state combination x is `congregate` if and only if $\lambda(x) \leq 1$.

THEOREM 3.9. *Under the Arithmetic Sequence Condition, there exists one optimum congregate state combination x for any given F.*

PROOF. We can prove the theorem by finding a congregate equivalence for any optimal solution $\check{x}$ with aggregate frequency $\check{F}$ and aggregate power $\check{P}$. If $\lambda(\check{x}) \leq 1$, then we are done. Assume that we can find a congregate equivalence of any state combination x with $\lambda(x) = k$, then we can find a congregate equivalence of any state combination x with $\lambda(x) = k+1$ according to Lemma 3.7 and the optimum assumption of $\check{x}$. Then the theorem is proved by mathematical induction. □

Using the preceding theorem, we can add the congregate constraint to restrict the solution space to speed up the table construction process; now we are able to find the congregate state combination only. Starting from $(n, 0, \ldots, 0)$, the left-most nonzero column decrements and its right column increments. Due to the Arithmetic Sequence Condition, all possible state combinations can be reached by this method.

An example congregate state combination table is shown in Table IV. The table size is reduced to $(m-1)n$ entries, the offline table construction time is reduced to $O(mn)$, and the online table lookup time is reduced to $O(lg(mn))$. Note that the greedy neighboring

Table IV. Congregate State Combination Table

F	P	x
nf_0	np_0	$(n, 0, \ldots, 0)$
$(n-1)f_0 + f_1$	$(n-1)p_0 + p_1$	$(n-1, 1, 0, \ldots, 0)$
$(n-2)f_0 + 2f_1$	$(n-2)p_0 + 2p_1$	$(n-2, 2, 0, \ldots, 0)$
.	.	.
.	.	.
$f_0 + (n-1)f_1$	$p_0 + (n-1)p_1$	$(1, n-1, 0, \ldots, 0)$
nf_1	np_1	$(0, n, 0, \ldots, 0)$
$(n-1)f_1 + f_2$	$(n-1)p_1 + p_2$	$(0, n-1, 1, 0, \ldots, 0)$
.	.	.
.	.	.
.	.	.
$2f_{m-2} + (n-2)f_{m-1}$	$2p_{m-2} + (n-2)p_{m-1}$	$(0, \ldots, 2, n-2)$
$f_{m-2} + (n-1)f_{m-1}$	$p_{m-2} + (n-1)p_{m-1}$	$(0, \ldots, 1, n-1)$
nf_{m-1}	np_{m-1}	$(0, \ldots, 0, n)$

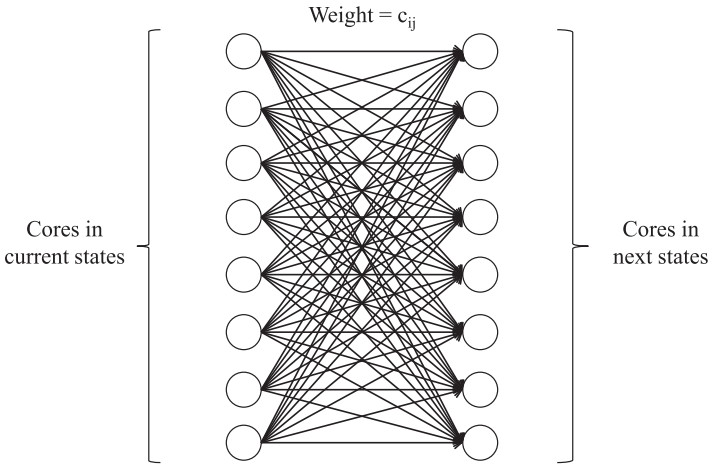

Fig. 2. Model state assignment as the minimum weight perfect bipartite matching problem. The complexity is proportional to the number of cores.

search algorithm [Meng et al. 2008] and SD [Winter et al. 2010] result in the same solution under this special condition.

4. STATE ASSIGNMENT

The `State Combination` process explained in the previous section determines the number of cores that should be put in each state, and the `State Assignment` process in this section assigns each core to its new state; different assignments result in different transition costs, which should be minimized.

4.1. Problem Mapping

The `State Assignment` process can be mapped to several existing graph problems. For example, it turns into the minimum weight perfect bipartite matching problem with each vertex representing a core and the edge weight being the transition cost, as shown in Figure 2 with $n = 8$, for example. It can be solved by the Hungarian algorithm [Kuhn 2010] in $O(n^3)$, but the scalability becomes a serious issue for an online policy.

Since the number of states m is usually limited while n may scale to a large number, the main factor of scalability and complexity is n. For example, at most 16 performance

levels can be specified in ACPI [2011], but multiprocessor systems may contain more than a hundred cores [Held et al. 2006]. Therefore, the online algorithm should reduce the order of n in the complexity.

We propose another mapping that reduces the complexity for homogeneous architectures. Similar to the next state x, we define the aggregate current state vector $w = (w_0, \ldots, w_{m-1})$, $w \in Z_+^m$, where w_i is the number of cores currently in the ith state. Note that w and x satisfy $\sum_{i=0}^{m-1} w_i = \sum_{j=0}^{m-1} x_j = n$ inherently. We also define the aggregate transitions as an $m \times m$ matrix Y, where $y_{ij} \in Z_+$ denotes the number of cores that are switched from the ith state to the jth state; y_{ij} is bounded by both w_i and x_j. Moreover, $\sum_{i=0}^{m-1} \sum_{j=0}^{m-1} c_{ij} y_{ij}$ is the total transition cost, which should be minimized. Then the problem is formulated as

$$\text{minimize} \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} c_{ij} y_{ij} \tag{20}$$

$$\text{subject to} \sum_{j=0}^{m-1} y_{ij} = w_i \text{ for } i = 0, 1, \ldots, m-1 \tag{21}$$

$$\sum_{i=0}^{m-1} y_{ij} = x_j \text{ for } j = 0, 1, \ldots, m-1. \tag{22}$$

We map the problem to the minimum cost maximum flow problem on a directed acyclic graph $G = (U \cup V \cup \{s, t\}, E_W \cup E_X \cup E_Y)$. The vertex sets $U = \{u_0, u_1, \ldots u_{m-1}\}$ and $V = \{v_0, v_1, \ldots, v_{m-1}\}$ correspond to the current and the target power states, respectively, with additional source s and sink t. The four sets of vertexes are connected by the edge sets $E_W = \{(s, u) : u \in U\}$, $E_X = \{(v, t) : v \in V\}$, and $E_Y = \{(u, v) : u \in U, v \in V\}$.

There are three functions defined on the edges. To fulfill constraints (21) and (22), the capacity function Capacity : $E \to Z_+$ is defined as $\text{Capacity}(s, u_i) = w_i$, $\text{Capacity}(v_j, t) = x_j$, and $\text{Capacity}(u_i, v_j) = n$, for $i, j = 0, 1, \ldots, m-1$. To minimize the objective (20), the weight function Weight : $E \to R_+$ is defined as $\text{Weight}(s, u_i) = \text{Weight}(v_j, t) = 0$ and $\text{Weight}(u_i, v_j) = c_{ij}$, for $i, j = 0, 1, \ldots, m-1$. The goal is finding the minimum cost flow with required (also maximum) flow n. The resultant flow function Flow : $E \to Z_+$ between U and V is our solution $y_{ij} = \text{Flow}(u_i, v_j)$, for $i, j = 0, 1, \ldots, m-1$.

An example mapped graph is shown in Figure 3 with $m = 4$. As stated earlier, the graph is much simpler than bipartite matching in Figure 2 because each vertex represents a *state* instead of a *core*.

4.2. Problem Solving

The minimum cost maximum flow problem can be solved by the successive shortest path algorithm (SSPA) with optimum solution [Edmonds and Karp 1972] as in Algorithm 2. SSPA iteratively finds the minimum-cost path between s and t on the residual network, and augments the flow under the capacity constraint. Since the transition cost C assigned for the edge weight is a nonnegative matrix, the minimum-cost *path* can be found using Dijkstra's algorithm for better efficiency.

The total number of edges is $E_W + E_X + E_Y = m + m^2 + m = m^2 + 2m$, and the number of iterations is bounded by the maximum capacity n. Thus, the worst-case time complexity is the number of edges times the number of iterations, which results in

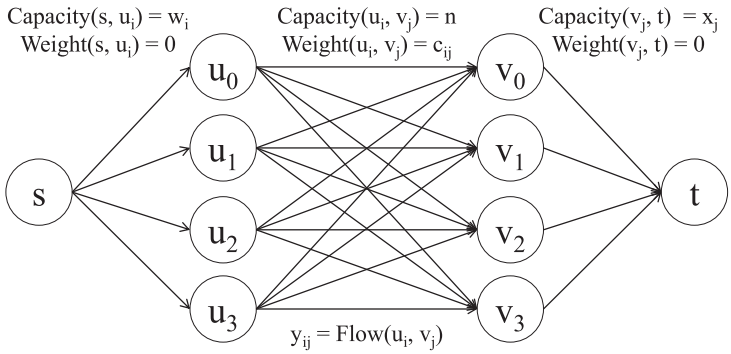

Fig. 3. Model state assignment as min-cost-max-flow problem. The complexity is proportional to the number of states.

ALGORITHM 2: Successive Shortest Path Algorithm

```
1  while the residual network contains a path s ⇝ t do
2  |  path ← DIJKSTRA(G, Capacity, Flow, Weight, s, t);
3  |  aug ← min_{(u,v)∈path} (Flow(v, u) > 0 : Flow(v, u)?Capacity(u, v) − Flow(u, v));
4  |  foreach (u, v) ∈ path do
5  |  |  if Flow(v, u) > 0 then
6  |  |  |  Flow(v, u) ← Flow(v, u) − aug;
7  |  |  end
8  |  |  else
9  |  |  |  Flow(u, v) ← Flow(u, v) + aug;
10 |  |  end
11 |  end
12 end
```

$O(m^2n)$. The complexity of SSPA can be further improved to $O(m^3 \lg(n/m))$ using the scaling technique [Edmonds and Karp 1972], which reduces the number of iterations by approximating the capacities with successively finer precisions.

In implementation, we can cache the mapping of $(w, x) \rightarrow y$ to further speed up our policy. Since the system characteristics (m, n, C) are statically given and the optimum state combinations (w, x) are finite, SSPA usually receives repeated input pairs and produces the same results for state assignment. Thus, we can record some recent results in a small table and look up the table before running SSPA with repeated input pairs. Simulation results show that less than 100 entries are enough, especially for systems with a small number of performance levels m.

4.3. Transitive Condition

The transition costs always satisfy the triangle inequality

$$c_{ij} + c_{jk} \geq c_{ik} \text{ for } 0 \leq i \leq j \leq k < m; \tag{23}$$

otherwise the state transition $i \rightarrow k$ is replaced by $i \rightarrow j \rightarrow k$ with smaller transition cost. When the equality holds, we call this condition `transitive`, which is common in power management. For example, when the transition delay is linearly proportional to the voltage change [Isci et al. 2006], the transition costs become transitive.

Under the `Transitive Condition`, the time complexity of the `State Assignment` problem can be further reduced. By linear scanning through the states, the augmenting path is directly determined, instead of finding by Dijkstra's algorithm.

ALGORITHM 3: State Assignment with Transitive Condition

```
1  i ← 0, j ← 0;
2  while i < m and j < m do
3  |  aug_u ← Capacity(s, u_i) − Flow(s, u_i);
4  |  aug_v ← Capacity(v_j, t) − Flow(v_j, t);
5  |  if aug_u < aug_v then
6  |  |  augment the path s → u_i → v_j → t with aug_u;
7  |  |  i ← i + 1;
8  |  end
9  |  else
10 |  |  augment the path s → u_i → v_j → t with aug_v;
11 |  |  j ← j + 1;
12 |  end
13 end
```

The complete method is shown in Algorithm 3. The iterators i and j scan through states U and V, respectively. The variables aug_u and aug_v record the residual capacity of u_i and v_j, respectively. The augmenting path is fixed as $s \to u_i \to v_j \to t$. During each iteration, at least one of u_i and v_j runs out of its residual capacity, and the corresponding iterator moves to the next state. The number of iteration is $2m$, and the operations in each iteration are done in constant time. Thus, the time complexity is greatly reduced to $O(m)$.

Note that the flows created by Algorithm 3 are cross free because i and j are both nondecreasing. Then we provide the optimality proof for this algorithm under the `Transitive Condition`. In other words, we need to prove that the resultant residual network does not contain any negative-weight cycle; otherwise, we can augment the cycle and lower the total costs.

First we define the forward edges $E_{forward} = \{(u, v) \in U \times V : \text{Flow}(u, v) < n\}$ and the reverse edges $E_{reverse} = \{(v, u) \in V \times U : \text{Flow}(u, v) > 0\}$ in the residual network of G induced by the flow in the preceding algorithm.

Then we define the weight of a path as the summation over the weight of each edge on the path. We need to prove the weight of any cycle (with equal number of forward and reverse edges) is nonnegative; otherwise, better solutions can be obtained by augmenting those negative-weight cycles.

LEMMA 4.1. *Let l denote the number of forward (reverse) edges of a cycle on the residual network, then the forward edges of a cycle form at least a cross for $l \geq 2$.*

PROOF. The path is closed to form a cycle. Since the reverse edges contain no cross, those forward edges must contain at least a cross if a cycle contains more than one forward edge. □

THEOREM 4.2. *The weight of any cycle on the residual network is nonnegative.*

PROOF. Since the flow on a cycle is the same, we add the weight of forward edges and subtract the weight of reverse edges to calculate the weight of a cycle.

For $l = 1$, $\text{Weight}(u, v) - \text{Weight}(u, v) = 0 : (u, v) \in E_{forward}, (v, u) \in E_{reverse}$.

For $l = 2$, let $u_\alpha, v_\beta, u_\gamma, v_\delta$ be the four vertexes on the cycle where $(u_\alpha, v_\beta), (u_\gamma, v_\delta) \in E_{forward}, (v_\beta, u_\gamma), (v_\delta, u_\alpha) \in E_{reverse}$. According to Lemma 4.1, $\gamma < \alpha \cap \beta < \delta \cup \alpha < \gamma \cap \delta < \beta$. Without loss of generality, we assume $\alpha \leq \beta$ and then there are six possibilities, as shown in Figure 4. In either case, $c_{\alpha\beta} - c_{\beta\gamma} + c_{\gamma\delta} - c_{\delta\alpha} \geq 0$.

Suppose that the weight of any cycle with $l = k$ ($k \geq 2$) is nonnegative, and without loss of generality, let the cycle with $l = k + 1$ be $u_\alpha \rightsquigarrow v_\beta \to u_\gamma \to v_\delta \to u_\alpha$; an example

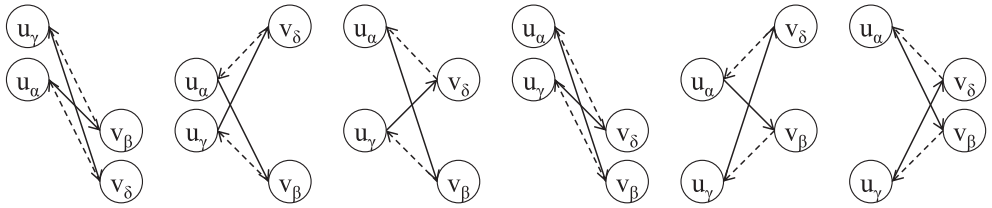

Fig. 4. The six possibilities for the weight of any cycle with two forward (reverse) edges on the residual network.

(a) $c_{\alpha\beta} + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = c_{\alpha\beta} + (c_{\gamma\alpha} + c_{\alpha\beta} + c_{\beta\delta}) - (c_{\gamma\alpha} + c_{\alpha\beta}) - (c_{\alpha\beta} + c_{\beta\delta}) = 0.$
(b) $c_{\alpha\beta} + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = (c_{\alpha\gamma} + c_{\gamma\beta}) + (c_{\gamma\alpha} + c_{\alpha\delta}) - c_{\gamma\beta} - c_{\alpha\delta} = c_{\alpha\gamma} + c_{\gamma\alpha} \geq 0.$
(c) $c_{\alpha\beta} + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = (c_{\alpha\delta} + c_{\delta\gamma} + c_{\gamma\beta}) + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = c_{\delta\gamma} + c_{\gamma\delta} \geq 0.$
(d) $c_{\alpha\beta} + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = (c_{\alpha\gamma} + c_{\gamma\delta} + c_{\delta\beta}) + c_{\gamma\delta} - (c_{\gamma\delta} + c_{\delta\beta}) - (c_{\alpha\gamma} + c_{\gamma\delta}) = 0.$
(e) $c_{\alpha\beta} + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = c_{\alpha\beta} + (c_{\gamma\beta} + c_{\beta\alpha} + c_{\alpha\delta}) - c_{\gamma\beta} - c_{\alpha\delta} = c_{\alpha\beta} + c_{\beta\alpha} \geq 0.$
(f) $c_{\alpha\beta} + c_{\gamma\delta} - c_{\gamma\beta} - c_{\alpha\delta} = (c_{\alpha\delta} + c_{\delta\beta}) + (c_{\gamma\beta} + c_{\beta\delta}) - c_{\gamma\beta} - c_{\alpha\delta} = c_{\delta\beta} + c_{\beta\delta} \geq 0.$

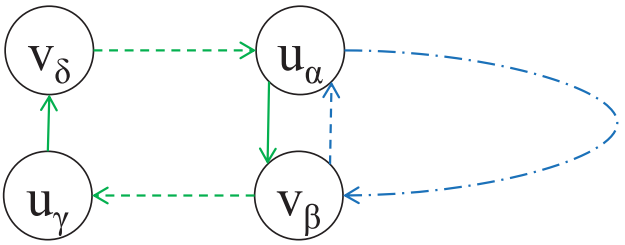

Fig. 5. Decompose a cycle $u_\alpha \rightsquigarrow v_\beta \to u_\gamma \to v_\delta \to u_\alpha$ with $l = k + 1$ into two cycles: $u_\alpha \rightsquigarrow v_\beta \to u_\alpha$ with $l = k$ and $u_\alpha \to v_\beta \to u_\gamma \to v_\delta \to u_\alpha$ with $l = 2$.

diagram is drawn in Figure 5. This cycle can be decomposed to $u_\alpha \rightsquigarrow v_\beta \to u_\alpha$ with $l = k$ and $u_\alpha \to v_\beta \to u_\gamma \to v_\delta \to u_\alpha$ with $l = 2$. Because either of the decomposed cycles has nonnegative weight, the total weight of the cycle with $l = k + 1$ is also nonnegative.

Then the proof is completed by mathematical induction. □

5. SIMULATION RESULTS

In this section, we first run simulations to compare the power and performance among different algorithms, then analyze the runtime (policy overhead) of them. The simulations and runtime analysis are both conducted on Intel Xeon CPU E5420 running at 2.5GHz.

Our algorithms based on CO are compared to SD [Winter et al. 2010], which is claimed to be more effective than other approaches in the work of Winter et al.

5.1. Environment Settings

We combine the instruction-set performance simulator Multi2Sim [Ubal et al. 2007] and power simulator McPAT [Li et al. 2009] to obtain statistics for multiprocessor systems. Both simulators are adapted to support per-core DVFS, where the frequency and voltage of each core can be scaled dynamically. The global power manager with the preceding policies are triggered periodically. In our simulations, the power management period is set to 1ms, which is between 0.5ms in Isci et al. [2006] and 10ms in Winter et al. [2010]. During each period, the performance statistics (the number of accesses of each component) of the last period are collected from Multi2Sim and sent to McPAT to obtain the corresponding power statistics (the dynamic and leakage power of each

Table V. SPEC CPU2006 Benchmarks [Henning 2006]

Group	Context 0	Context 1	Context 2	Context 3
group1	gcc (I)	perlbench (I)	bwaves (F)	bzip2 (I)
group2	milc (F)	gamess (F)	zeusmp (F)	mcf (I)
group3	cactusADM (F)	leslie3d (F)	gromacs (F)	namd (F)
group4	soplex (F)	povray (F)	dealII (F)	gobmk (I)
group5	sjeng (I)	GemsFDTD (F)	hmmer (I)	calculix (F)
group6	h264ref (I)	lbm (F)	tonto (F)	libquantum (I)
group7	aster (I)	xalancbmk (I)	sphinx3 (F)	omnetpp (I)

Table VI. Configuration Parameters of ARM Cortex A9 [ARM 2008]

Parameter Name	Parameter Value
Number of cores	4
Number of threads per core	1
Technology node	40nm
Operating frequency	2000MHz
Supply voltage	0.66V
Threshold voltage	0.23V
Decode width	2
Issue width	4
Commit width	4
Number of ALUs per core	3
Number of MULs per core	1
Number of FPUs per core	1
Branch predictor	Two-level, 1024-set, 2-way BTB
L1 data cache	32KB, 4 way, 10-cycle latency
L1 instruction cache	32KB, 4 way, 10-cycle latency
L2 unified cache	1MB, 8 way, 23-cycle latency

component). Based on the feedback information, the global power manager determines the power mode of each core and sets corresponding frequency and voltage for the next period.

We use the benchmark set SPEC CPU2006 V1.2 [Henning 2006], whereas its previous (retired) version CPU2000 is widely used to evaluate DVFS policies in previous works [Isci et al. 2006; Teodorescu and Torrellas 2008; Meng et al. 2008; Winter et al. 2010]. The benchmarks are randomly partitioned into seven groups, as listed in Table V. Each group contains a random number of integer (denoted as I) or floating-point (denoted as F) benchmarks. The benchmarks (contexts) are scheduled using the built-in dynamic scheduler in Multi2Sim [Ubal et al. 2007], where the thread-binding overheads are inherently modeled in the simulator. Each run of simulation is terminated after 1 billion instructions in total are committed.

The target multiprocessor architecture is based on ARM Cortex-A9 MPCore [ARM 2008], which consists of up to four processors in a cluster and a Snoop Control Unit (SCU) ensuring coherency. Up to 14 power domains are supported, where the CPU, data engine, and cache(s) of each processor can reside in different power domains. With the IEM support, per-core DVFS policies can be implemented in software. The configuration parameters obtained from the samples of McPAT [Li et al. 2009] and the official manual [ARM 2008] are listed in Table VI. The technology parameters are supported by McPAT [Li et al. 2009] as well. The simulations can be extended to systems with more cores when the technology parameters are released by McPAT.

The settings of power modes are based on the ARM IEC [ARM 2005], which controls the dynamic clock generator (DCG) and the dynamic voltage controller (DVC) to realize DVFS-based intelligent energy management (IEM). Power modes can be selected from up to 129 performance levels, where 128 and 0 are the highest and lowest levels,

Table VII. Power Modes [ARM 2005]

Index Value	Binary Value	Performance Level	Operating Frequency	Core Voltage
0	10000000	100%	2000.0MHz	0.66V
1	01011100	72%	1437.5MHz	0.54V
2	01001000	56%	1125.0MHz	0.47V
3	00100100	28%	562.5MHz	0.35V

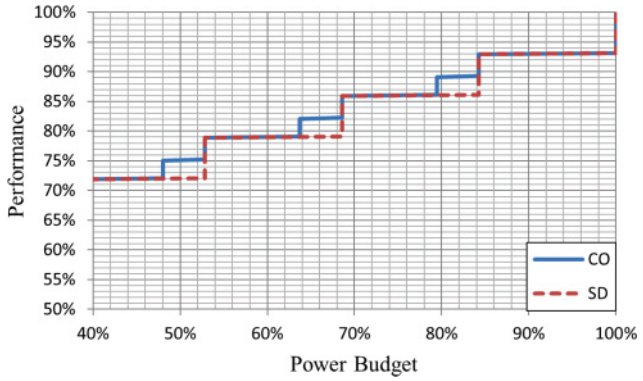

Fig. 6. Steady-state comparison for the two policies. The performance of CO is higher than or equal to SD under any given power budget.

respectively. The power modes are obtained from the IEC manual [ARM 2005], whereas the architecture and technology parameters are obtained from McPAT [Li et al. 2009]. The transition delay follows the $10mV/1\mu$s setting as in Isci et al. [2006]. The detailed values are listed in Table VII for simulation.

The relative power budget $np_{m-1} \leq P \leq np_0$ is generated using the same method as previous works [Isci et al. 2006] for comparison. Its value is updated every power management epoch, according to the real power budget P_{budget} (in Watts), the measured power $P_{measured}$ (in Watts), and the current power state combination. The equation is

$$P = \sum_{i=0}^{m-1} p_i x_i * \frac{P_{budget}}{P_{measured}}. \tag{24}$$

Note that our policy is independent of the method of generating P; implementing this method is only for fair comparisons.

5.2. Performance Analysis

First we show the steady-state performance under different values of power budget for CO and SD in Figure 6. The x-axis is the power budget, representing the percentage of maximum aggregate power. The y-axis is the resultant performance, representing the percentage of maximum aggregate frequency. Compared to SD, CO achieves higher or equal performance in any power budget, with the maximum improvement being 3.12%. For example, when the power budget is 68%, CO is able to find the power states (0, 0, 1, 2) for the four cores with 82.03% aggregate frequency, whereas SD can only find (0, 1, 1, 1) with 78.91% aggregate frequency at best.

Then we show the simulation results of performance curve for different benchmark sets of CPU2006 in Figure 7, where the y-axis is the throughput (1 billion instructions over the execution cycles) normalized to full-speed performance. In each of the seven groups, CO achieves higher or equal throughput under the same power budget against

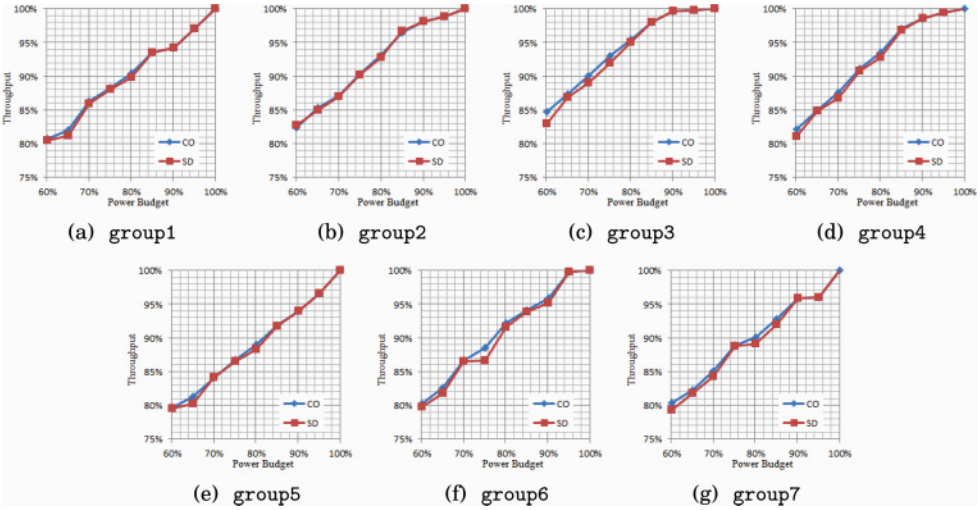

Fig. 7. Performance curve under different values of power budget. For any group, CO achieves higher or equal throughput with the same budget against SD [Winter et al. 2010].

SD. The curves are smoother than the steady-state analysis in Figure 6 because the runtime power varying induces the state changes across several power modes.

In the steady-state comparison in Figure 6, we only count the dynamic power, which is cubic proportional to the performance. However, in the simulation, the performance improvement is diminished due to the inclusion of static power, which is only linearly proportional to the performance. If the static power dominates the total power, then CO and SD will find equivalent state combinations under any power budget. Besides, the performance degradation is smaller for memory-bound benchmarks because the memory subsystem is not slowed down as the cores. (Similar results are demonstrated in Isci et al. [2006] as well.)

At last, we verify if the actual power consumption violates the given power budget or not. In Figure 8, the x-axis is the power budget and the y-axis is the actual power consumption, including dynamic and static power. We also draw the auxiliary green lines and show that the power consumption of both CO and SD are under the given power budget constraints.

5.3. Policy Overhead Analysis

As stated earlier, power management overhead contains the transition delay between power states and the time to run the management policy itself. The former part is implicitly included in the performance simulation shown in the previous section, whereas the policy runtime is excluded because our instruction-set simulator lacks for the full-system operating system. Therefore, we measure the runtime overhead of the two policies and show the results in this section. To analyze the scalability of the two policies through simulation, we scale (m, n) up to 16 performance levels (according to ACPI) and 512 cores (256 cores in Winter et al. [2010]), using the power statistics from the previous section. The highest m performance levels from IEC [ARM 2005] are selected for the power states.

The dynamic runtime of both policies are shown in Table VIII, which is averaged by running 1 million epochs of GPM. Less than 100 table entries are used in CO, where the caching effect of accelerating state assignment is analyzed in the next paragraph. We have verified that our implementation of SD performs consistent simulation results

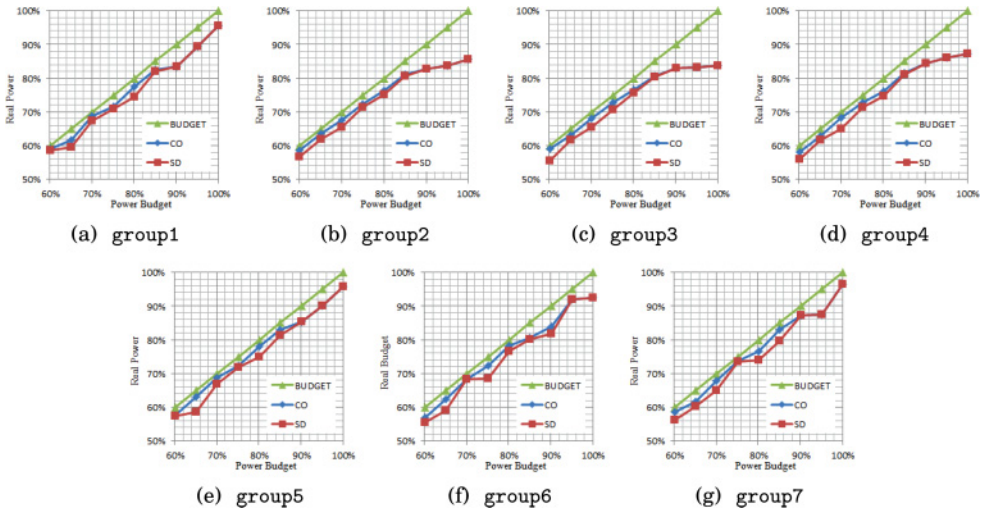

Fig. 8. Real power curve under different values of power budget. For any group, both CO and SD [Winter et al. 2010] are under any given power budget.

Table VIII. Runtime Comparison for Each Power Management Epoch

m	n	2	4	8	16	32	64	128	256	512
4	SD (μs)	0.4	0.9	1.7	3.5	7.2	15.4	31.7	66.1	138.0
	CO (μs)	0.0	0.0	0.1	0.3	0.7	0.9	1.0	1.0	1.1
	Speedup (X)	11.2	21.1	30.8	10.6	10.9	17.6	32.3	64.1	125.6
8	SD (μs)	0.6	1.4	3.1	6.7	14.4	30.2	63.9	134.1	282.8
	CO (μs)	0.1	0.1	0.2	2.2	4.3	5.4	5.9	6.0	6.1
	Speedup (X)	12.9	26.2	13.1	3.0	3.3	5.6	10.9	22.2	46.1
16	SD (μs)	0.9	2.1	4.9	10.7	23.1	50.0	106.2	229.0	473.4
	CO (μs)	0.1	0.1	0.5	6.1	11.9	14.8	16.0	16.5	16.7
	Speedup (X)	13.7	29.8	9.7	1.8	1.9	3.4	6.7	13.9	28.3

with Winter et al. [2010]. The runtime of CO is faster than SD for any combination of (m, n); our policy achieves up to 125.6 times speedup against the state-of-the-art previous work. The speedup is greater for less number of states ($m = 4$) because the (general-case) time complexity is $O(m^3 \lg(n/m))$ for CO and $O(mn \lg n)$ for SD; we emphasize the scalability on the number of cores n while the number of states m is limited. With $(m, n) = (16, 512)$, CO consumes only 16.7μs on average, which is light for per-core GPM. Note that the preceding analysis is for general cases; our policy can be faster when any of the arithmetic sequence or the transitive conditions occurs.

Then we analyze the runtime improvement using the state assignment table (caching), as shown in Figure 9 with different number of states m. In each figure, the x-axis represents the number of cores n in logarithmic scale, and the y-axis shows the runtime in microseconds. Different lines correspond to different policies with the maximum table size inside the parentheses (with empty table at the beginning of simulation). The solid lines show the same data as in Table VIII with less than 100 entries in CO. The dot blue lines represent CO without state assignment table, showing that the upper bound of the proposed policy is still scalable. The caching of state assignment is effective for a small number of cores; using more table entries results in better runtime for larger m or n. According to the system architecture (number of states and cores)

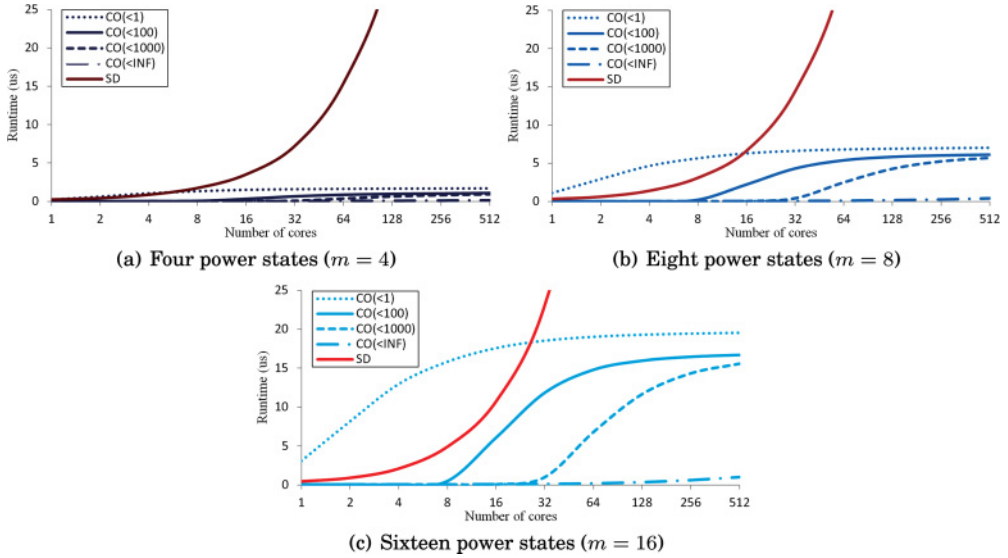

Fig. 9. Runtime for online state assignment with different table sizes. Although CO is scalable without table, it can be further improved using state assignment tables for small number of cores.

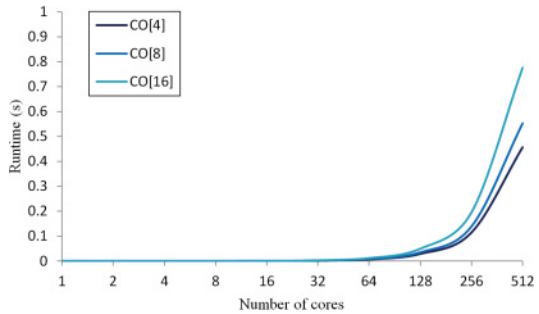

Fig. 10. Runtime for offline table generation. The one-time-only preprocessing of CO requires less than 1 second for up to 16 power states and 512 cores.

and requirement (averaged policy runtime), designers can trade off between runtime and table size.

Finally, we show the overhead for table construction in Figure 10, where the unit of the y-axis is in seconds. Different lines correspond to a different number of power states with m in the brackets, matching the $O(mn^2)$ time complexity as analyzed. It consumes only 0.76s for 16 states and 512 cores, which is acceptable for one-time-only offline preprocessing.

6. RELATED WORKS

In recent years, there are more and more papers discussing DVFS-based power management on multiprocessors. They can be classified by the considerations (performance, power, thermal, and process variation), the target workloads (single- or multithreaded, single- or multiprogrammed), the system architectures (chip-multiprocessors and many-core architectures), and the scope of algorithms (global, distributed, and hierarchical power managers).

Performance and power are the main considerations; each policy either minimizes power consumption with performance constraint or maximizes throughput with power budget [Li and Martinez 2005; Rajamani et al. 2006]. Temperature is another constraint to multiprocessors [Zanini et al. 2012], where dynamic thermal management (DTM) combines DVFS and task migration to remove hotspots [Hanumaiah et al. 2011; Cochran and Reda 2012]. Core-to-core process variation is considered in some works [Lee and Kim 2009]; the power managers are implemented in either hardware [Herbert and Marculescu 2009] or software [Teodorescu and Torrellas 2008]. For the preceding considerations, some information should be provided by dedicated hardware, such as the performance counters and power sensors [Cochran and Reda 2012].

The per-core DVFS technique is shown better than chip-wide DVFS [Kim et al. 2008] for the global asynchronous local synchronous architecture. Several distributed policies with one power manager per island or core are proposed [Herbert and Marculescu 2007], including threshold-based policy [Talpes and Marculescu 2005], controller-based policy [Juang et al. 2005], greedy-search policy [Magklis et al. 2006], and learning-based policy [Shen et al. 2013]. Distributed policies are scalable but suboptimal due to the lack of the global view.

Although per-core power management performs better than chip-wide control [Kim et al. 2008], the overhead induced in centralized managers become overwhelming for many-core architectures. Therefore, hierarchical approaches are introduced, using DVFS to control the power and maximize throughput [Hanumaiah et al. 2011], either for static islands [Mishra et al. 2010] or for dynamic groups of cores [Ma et al. 2011a].

Power management with multithreaded workloads are considered in some works [Mishra et al. 2010]. Cebrian et al. [2011] proposed the power token balancing mechanism to migrate the power tokens to the critical thread for multithreaded workloads. On the other hand, GPM policies aim at overall throughput maximization for multiprogrammed workloads, as stated in the first section [Isci et al. 2006; Teodorescu and Torrellas 2008; Meng et al. 2008; Winter et al. 2010].

Note that Hanumaiah et al. [2011] and Ma et al. [2011a] and the following heuristics optimize throughput by maximizing the aggregate frequency as well. Several works are based on GPM policies, extending to other power management techniques and architectures. Sartori and Kumar [2009] extend MaxBIPS to become hierarchical for many-core architectures. The thread motion technique [Rangan et al. 2009] also exploits MaxBIPS and migrate applications between cores with different voltage and frequency levels. Besides, the DVFS-based policies can be combined with DPM to form hybrid policies that can save more power in different environments [Bhatti et al. 2010; Srivastav et al. 2012]. The power budget or other constraints can be determined by control- or learning-based mechanisms in specific applications [Maggio et al. 2012].

7. CONCLUSIONS

In this article, we provide a scalable GPM policy for per-core DVFS; we first maximize the steady-state performance in the state combination phase and then minimize the transition cost in the state assignment phase.

We propose an optimum offline algorithm to construct the state combination table by transforming the original problem and solving with dynamic programming in pseudo–polynomial time. We obtain optimum solutions in linear time for state assignment by modeling it as a minimum cost maximum flow problem and solving with the SSPA. We also observe the arithmetic sequence and transitive conditions in GPM to further speed up table construction and state assignment, respectively. The optimality of either general or special solutions are mathematically proved; besides, the proposed approach is shown to be better and faster than the state-of-the-art algorithm using commercial benchmarks.

Since many advanced policies are based on GPM, our algorithms form an efficient engine for them. For example, it can be exploited in hybrid (DPM and DVFS) policies to enhance the scalability. It could be wrapped into learning- or controller-based policies for long-term optimization of the power budget. Process variation, thermal issues, and workload variation may be further modeled and considered in the future.

REFERENCES

ACPI. 2011. ACPI—Advanced Configuration and Power Interface Specification. Available at http://www.acpi.info/

AMD. 2013. AMD PowerNow! Technology. Available at http://www.amd.com/us/products/technologies/amd-powernow-technology/Pages/amd-powernow-technology.aspx

ARM. 2005. ARM Intelligent Energy Controller Technical Overview. http://www.arm.com/

ARM. 2008. *Cortex-A9 MPCore Technical Reference Manual*. ARM.

Intel. 2013. Enhanced Intel SpeedStep Technology. Retrieved August 13, 2015, from http://www3.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm

Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8, 3, 299–316.

Khurram Bhatti, Cecile Belleudy, and Michel Auguin. 2010. Power management in real time embedded systems through online and adaptive interplay of DPM and DVFS policies. In *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. 184–191.

Juan M. Cebrian, Juan L. Aragon, and Stefanos Kaxiras. 2011. Power token balancing: Adapting CMPs to power constraints for parallel multithreaded workloads. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*.

Ryan Cochran and Sherief Reda. 2012. Thermal prediction and adaptive control through workload phase detection. *ACM Transactions on Design Automation of Electronic Systems* 18, 1, Article No. 7.

Jack Edmonds and Richard M. Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery* 19, 2, 248–264.

Yang Ge, Parth Malani, and Qinru Qiu. 2010. Distributed task migration for thermal management in many-core systems. In *Proceedings of the 47th ACM/IEEE Design Automation Conference*. 579–584.

Vinay Hanumaiah, Sarma Vrudhula, and Karam S. Chatha. 2011. Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 11, 1677–1690.

Jim Held, Jerry Bautista, and Sean Koehi. 2006. *From a Few Cores to Many: A Tera-Scale Computing Research Overview*. Technical Report. Intel Corporation.

John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News* 34, 4, 1–17.

Sebastian Herbert and Diana Marculescu. 2007. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 38–43.

Sebastian Herbert and Diana Marculescu. 2009. Variation-aware dynamic voltage/frequency scaling. In *Proceedings of the International Symposium on High Performance Computer Architecture*. 1–12.

Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. 347–358.

Niraj K. Jha. 2001. Low power system scheduling and synthesis. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer Aided Design*. 259–263.

Philo Juang, Qiang Wu, Li-Shiuan Peh, Margaret Martonosi, and Douglas W. Clark. 2005. Coordinated, distributed, formal energy management of chip multiprocessors. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*. 127–130.

Wonyoung Kim, Meeta S. Gupta, Gu-Yeon Wei, and David Brooks. 2008. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture*. 123–134.

Matthias Knoth. 2009. Power management in an embedded multiprocessor cluster. In *Proceedings of the Embedded World Conference*.

Harold W. Kuhn. 2010. The Hungarian method for the assignment problem. In *50 Years of Integer Programming 1958–2008*, J. Junger, Th. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pullyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey (Eds.). Springer, 29–47.

Jungseob Lee and Nam Sung Kim. 2009. Optimizing total power of many-core processors considering voltage scaling limit and process variations. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*. 201–206.

Jian Li and Jose F. Martinez. 2005. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Transactions on Architecture and Code Optimization* 2, 4, 397–422.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 469–480.

Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. 2011a. Scalable power control for many-core architectures running multi-threaded applications. *ACM SIGARCH Computer Architecture News* 39, 3, 449–460.

Kai Ma, Xiaorui Wang, and Yefu Wang. 2011b. DPPC: Dynamic power partitioning and capping in chip multiprocessors. In *Proceedings of the 2011 IEEE 29th International Conference on Computer Design*. 39–44.

Martina Maggio, Henry Hoffmann, Alessandro V. Papadopoulos, Jacopo Panerati, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. 2012. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Transactions on Autonomous and Adaptive Systems* 7, 4, 36:1–36:32.

Grigorios Magklis, Pedro Chaparro, Jose Gonzalez, and Antonio Gonzalez. 2006. Independent front-end and back-end dynamic voltage scaling for a GALS microarchitecture. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*. 49–54.

Ke Meng, Russ Joseph, Robert P. Dick, and Li Shang. 2008. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. 177–186.

Asit K. Mishra, Shekhar Srikantaiah, Mahmut Kandemir, and Chita R. Das. 2010. CPM in CMPs: Coordinated power management in chip-multiprocessors. In *Proceedings of the 2010 International Conference for High Performance Computing, Networking, Storage, and Analysis*. 1–12.

Massoud Pedram. 1996. Power minimization in IC design: Principles and applications. *ACM Transactions on Design Automation of Electronic Systems* 1, 1, 3–56.

Karthick Rajamani, Heather Hanson, Juan Rubio, Soraya Ghiasi, and Freeman Rawson. 2006. Application-aware power management. In *Proceedings of the 2006 IEEE International Symposium on Workload Characterization*. 39–48.

Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. 2009. Thread motion: Fine-grained power management for multi-core systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*. 302–313.

John Sartori and Rakesh Kumar. 2009. Distributed peak power management for many-core architectures. In *Proceedings of the Conference on Design, Automation, and Test in Europe*. 1–4.

Joseph Sharkey, Alper Buyuktosunoglu, and Pradip Bose. 2007. Evaluating design tradeoffs in on-chip power management for CMPs. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*. 44–49.

Hao Shen, Ying Tan, Jun Lu, Qing Wu, and Qinru Qiu. 2013. Achieving autonomous power management using reinforcement learning. *ACM Transactions on Design Automation of Electronic Systems* 18, 2, Article No. 24.

Meeta Srivastav, Michael B. Henry, and Leyla Nazhandali. 2012. Design of energy-efficient, adaptable throughput systems at near/sub-threshold voltage. *ACM Transactions on Design Automation of Electronic Systems* 18, 1, Article No. 3.

Emil Talpes and Diana Marculescu. 2005. Toward a multiple clock/voltage island design style for power-aware processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, 5, 591–603.

Radu Teodorescu and Josep Torrellas. 2008. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*. 363–374.

Rafael Ubal, Julio Sahuquillo, Salvadore Petit, and Pedro López. 2007. Multi2Sim: A simulation framework to evaluate multicore-multithreaded processors. In *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*. 62–68.

Xiaorui Wang, Kai Ma, and Yefu Wang. 2011. Adaptive power control with online model estimation for chip multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 22, 10, 1681–1696.

Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*. 29–40.

Frances Yao, Alan Demers, and Scott Shenker. 1995. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. 374–382.

Francesco Zanini, David Atienza, Colin N. Jones, Luca Benini, and Giovanni De Micheli. 2012. Online thermal control methods for multiprocessor systems. *ACM Transactions on Design Automation of Electronic Systems* 18, 1, Article No. 6.

Received June 2013; revised May 2014; accepted August 2014