PEPON: Performance-Aware Hierarchical Power Budgeting for NoC Based Multicores*

Akbar Sharifi, Asit K. Mishra, Shekhar Srikantaiah, Mahmut Kandemir and Chita R. Das
Department of CSE
The Pennsylvania State University
University Park, PA 16802, USA
{akbar, amishra, srikanta, kandemir, das}@cse.psu.edu

ABSTRACT

Targeting NoC based multicores, we propose a two-level power budget distribution mechanism, called PEPON, where the first level distributes the overall power budget of the multicore system among various types of on-chip resources like the cores, caches, and NoC, and the second level determines the allocation of power to individual instances of each type of resource. Both these distributions are oriented towards maximizing workload performance without exceeding the specified power budget. Extensive experimental evaluations of the proposed power distribution scheme using a full system simulation and detailed power models emphasize the importance of power budget partitioning at both levels. Specifically, our results show that the proposed scheme can provide up to 29% performance improvement as compared to no power budgeting, and performs 13% better than a competing scheme, under the same chip-wide power cap.

Categories and Subject Descriptors

C.1.2 [**Computer Systems Organization**]: Multiprocessors; Interconnection architectures; C.4 [**Performance of Systems**]: Design studies

General Terms

Design, Experimentation, Management, Performance

Keywords

Power Budgeting, NoC-based Multicores, Performance

1. INTRODUCTION

Power budgeting and performance optimization under a given power budget in multicore systems have received considerable attention recently [19, 20, 24, 32]. A majority of these solutions focus on power budgeting at the processor level and the only *knob* they use to control the power of a multicore processor is the voltage/frequency level of the cores. While modulating the voltage/frequency level of the cores in a multicore processor has the potential to control power consumption of the processor, it provides no direct control on the efficiency (performance per Watt) of the processor. This is because the performance of an application is not controlled by the core frequency alone (although it has a vital influence), but is also influenced by other on-chip components; specifically, the last-level-cache and the network-on-chip (NoC). In fact, the significance of the power consumption of the non-core components is increasing and projected to be quickly on par with the power consumption of cores themselves, if not greater [11, 12, 23]. Our experimental studies also reinforce these observations made in the literature. For example, our experiments on a 16-core machine with 8MB last-level cache and a 4×4 mesh-based NoC show that about 33% of the power is consumed by the non-core components on average (15% and 18% by the last-level cache banks and the NoC, respectively).

A recent network based multicore from Intel [12] reports 31% and 79% non-core power contribution in full power and low power operating modes, respectively. Despite the fact that the power consumptions of caches and NoC are on par with that of the cores, to the best of our knowledge, there has been *no* prior work on distributing the power budget among all these resources to maximize performance, while maintaining the same overall power budget of the system as a whole. We believe significant performance improvements can be achieved by distributing the overall power intelligently among the resources based on the workload's sensitivity to core/NoC frequency and cache allocation.

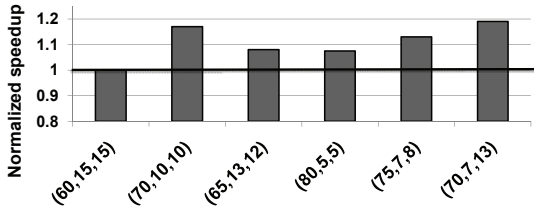


Figure 1: Performance (normalized weighted speedup) of a mix of 16 applications on a 16-core machine under different distributions of the same power budget (90W). All bars are normalized to the first bar. (x,y,z) on the x-axes indicates the specified core, cache and NoC power budgets (in watts), respectively.

To illustrate the key insight leveraged by this paper, that different distributions of the overall system power can lead to different system performances (despite keeping the overall power budget the same), we performed a simple experiment. We allocated different percentages of a fixed chip-wide power cap (90W) to different resources in the system. The core frequency is tuned so that the overall power allocation to the cores is respected. Similarly, the number

*This research is supported in part by NSF grants #1213052, #1152479, #1147388, #1139023, #1017882, #0963839, #0811687 and a grant from Microsoft Corporation.

PACT'12, September 19–23, 2012, Minneapolis, Minnesota, USA.

of active cache ways in a set-associative last-level cache (LLC) are tuned such that the overall cache power remains within the allocated budget, and the NoC frequency is set such that the NoC power budget is also respected. The performance of a multiprogrammed workload mix of 16 applications on a 16-core 4×4 system as measured by the normalized weighted speedup under various distributions of the (same) overall power budget is shown in Figure 1. From this figure, we can see that different power budget distributions (x-axis) result in significantly different system performance, under the same power budget. Note that, the results presented in Figure 1 are only with a *static distribution* of the overall power among different resources, which does not react to changes in the workload behavior. Further, any changes in the power allocation to a specific resource requires tracking the actual (now-limited) resource allocation to applications over time. Therefore, a *dynamic distribution* of the power budget can potentially achieve much higher overall performance and a mechanism to track the allocated power among applications can also ensure that the power budget is always respected despite changing workload behavior.[1]

To this end, in this paper, we propose a power budget distribution mechanism (called *PEPON*) that distributes the chip-wide power budget among various types of critical resources like the cores, caches, and NoC to maximize performance, while respecting the allocated budget in the corresponding resources, thereby respecting the overall power budget. Specifically, the main contributions of the paper can be summarized as follows:

• We propose a novel *two-level* power distribution scheme that distributes a specified power budget among different resources by carefully managing the allocated power to attain best possible performance for a given workload. This scheme employs different models at different power distribution levels and uses feedback controllers to enforce the core and NoC power allocations.

• Extensive evaluation of the proposed technique using a full system simulation and detailed power/performance models shows that carefully distributing the power budget among all the on-chip resources not only ensures that the power cap on the processor is respected, but also optimizes the system performance. Our experimental analysis indicates that overall workload performance can be improved by as much as 29%, as compared to the case where no power distribution is implemented. Further, with respect to a recently-proposed power budgeting scheme that considers redistribution of only core power [20], our scheme achieves, on average, about 13% performance improvement –under the same chip-wide power budget.

2. TARGET MULTICORE SYSTEM

Figure 2 shows an NoC-based multicore architecture. It is a shared-memory based two-dimensional mesh structure, where each node contains a processor core, private L1 data or instruction caches, a shared L2 cache bank, and a router through which it gets connected to the neighboring nodes. Although all nodes share the same logical on-chip L2 cache space, data access latencies depend on the distance between the requesting core and the cache bank that holds the requested data. The set of links used during an access to a data element in some other node's L2 is determined by the routing policy, which is XY-routing [17] in this work.

[1]Note that, while (for a given workload/configuration), one may find a static partitioning that comes close to the dynamic one, one would need an exhaustive search to achieve this (which is not possible in practice). Further, in other cases, one may not even find a comparable static partitioning (since static partitionings do not consider dynamic modulations in workload behavior).

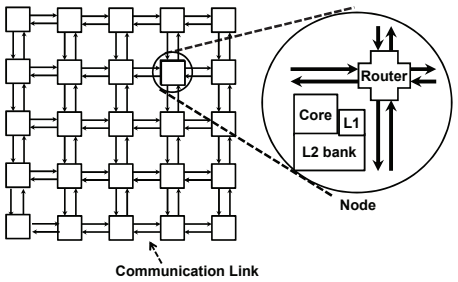


Figure 2: An NoC-based multicore architecture.

One of the critical issues in designing such multicore architectures is *power consumption,* which mainly comes from three sources: cores, on-chip cache components (shared L2 banks), and the NoC. To reduce the power consumption of cores, prior *dynamic voltage/frequency scaling* (DVFS) based methods [4, 12, 14] proposed in the literature can be employed. The idea of DVFS is to exploit the variance in processor utilization by lowering the voltage/frequency when the processor is lightly loaded and increasing the voltage-/frequency when the processor is highly utilized. Like processor cores, there is also a wide variance in cache line utilization in on-chip memories as well as link utilization in NoC-based multicore architectures, depending on applications' communication/data sharing patterns. In fact, one can obtain up to $10\times$ power savings, by accurately shutting down unused cache ways or lowering frequency and voltage of links when link usage is decreased [16]. We exploit DVFS for varying the power allocations of cores and the NoC.

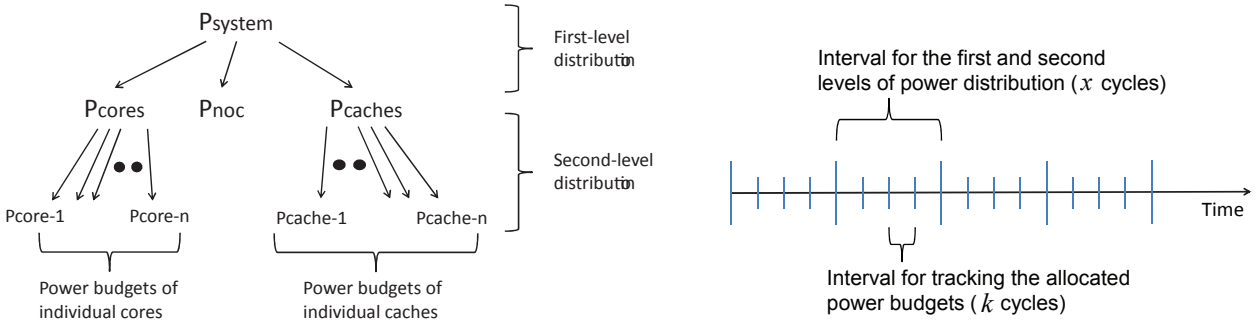
3. OUR PROPOSED APPROACH

3.1 High-Level View of PEPON

Targeting the multicore architecture shown in Figure 2, our goal in this work is to maximize workload performance under a chip-wide power budget constraint. More specifically, given a multicore-wide power budget, we want to distribute/redistribute it across different hardware components such that (i) the given power budget is not exceeded and (ii) performance is maximized. Consequently, we are not interested in solutions that maximize workload performance but violate the specified power budget. Further, among all solutions that satisfy the power budget constraint, we are interested in finding the one that maximizes performance.

Figure 3(a) illustrates the high level operation of our proposed two-level power budget distribution strategy (PEPON). In the first level of this strategy, the overall power budget is partitioned among cores, NoC and L2 caches. In the second level, the power budget assigned to cores is further partitioned among individual cores and, similarly, the power budget assigned to L2 caches is further partitioned across individual L2 caches. While it is also possible to divide the NoC into voltage islands [26] and distribute the power allocated to the NoC across these islands, we did not pursue that direction in this work because such a partitioning can have a non-deterministic impact on different applications. Instead, in this work, we treated the NoC as a single monolithic entity, all components of which have the same voltage/frequency level at a given time (i.e., when voltage/frequency scaling is applied, it is applied to the entire NoC).

At each level of this power distribution strategy, a different approach is employed. Specifically, for the first-level distribution, we adopt a *regression-based* performance model that guides us to decide the most performance-efficient distribution of power. For the second-level distribution, on the other hand, we use different strate-



(a) Power budget distribution. (b) Control intervals.

Figure 3: High level view of PEPON.

gies for caches and cores. The power budget assigned to cores is distributed using a control theoretic approach. On the other hand, for power budget distribution across different L2 banks, we employ a *utility-based* model. The technical details of these models as well as the details of our two-level power distribution strategy and our power control knobs are described next.

3.2 Design Details of PEPON

For the first level power distribution, the system power budget (P_{system}) is partitioned among different types of components, namely, cores (P_{cores}), cache banks (P_{caches}), and NoC (P_{noc}). That is, we have: $P_{system} = P_{cores} + P_{caches} + P_{noc}$. At the second level, P_{cores} is distributed among the cores and P_{caches} is distributed among the cache banks. Both the first and the second-level distributions are performed at every x million cycles. Further, each component of interest (each core, cache bank and NoC) has a *controller* associated with it, which tracks the allocated budget. As illustrated in Figure 3(b), the longer epochs (x million cycles) are divided into several sub-epochs, and the controller associated with each component is invoked every k million cycles (sub-epochs) to track its power budget, which is determined at every x million cycles. While we performed experiments with different values of x and k parameters, the default values used in most of our experiments are 100 and 20, respectively.

3.2.1 First-Level Power Distribution

At this level, we employ a *regression-based* estimator:

$$Perf = \epsilon + a_1 \times P_{cores} + a_2 \times P_{caches} + a_3 \times P_{noc}. \quad (1)$$

In this expression, $Perf$ is the overall workload performance, a_i ($1 \leq i \leq 3$) represent model coefficients, and ϵ captures the model error value. This model gives the overall performance ($Perf$) expected to be achieved over an epoch when the total power budget is partitioned into P_{cores}, P_{caches} and P_{noc} values in that epoch. In our model, we define the overall workload performance ($Perf$) over each time interval as $\sum_i \frac{IPS_i^k}{Avg\_IPS_i}$, where IPS_i^k is the *instruction per second* (IPS) of the ith application over the kth time interval and Avg_IPS_i is the average measured IPS of that application when it is executed without any power constraint. Avg_IPS_i can be obtained for each application by running that application on a core with the highest possible frequency and without imposing any power budget constraint.

At each time interval, first, the coefficients are updated based on the measured performance and allocated power budgets in the previous epoch and then our scheme increases the power budget of the component with the largest coefficient and reduces the power budget of the component with the smallest coefficient. As an example, if $a_1 > a_2 > a_3$, then P_{cores} and P_{noc} are increased by δ and reduced by $\delta - \epsilon_1$, respectively (δ is an increment/decrement unit and ϵ_1 is a small number and helps the regression to be computable). To update the model, we keep a *window* (*table*) with *four* entries that store the power budget allocations and corresponding measured performance numbers for the four recent epochs. Each entry in this table is in the form of ($Perf$, P_{cores}, P_{caches}, P_{noc}); an example table is given in Table 1. At each budget distribution epoch (the longer of the intervals shown in Figure 3(b)), first, the oldest entry in the table is replaced with the new entry ($Perf^k$, P_{cores}^k, P_{caches}^k, P_{noc}^k), where $Perf^k$ is the measured overall system performance over the last epoch and P_{cores}^k, P_{caches}^k and P_{noc}^k are the recent power budgets assigned to the cores, caches and NoC, respectively. The data stored in Table 1 and the model coefficients can be stacked together and written as follows:

$Perf^k$	P_{cores}^k	P_{caches}^k	P_{noc}^k
$Perf^{k-1}$	P_{cores}^{k-1}	P_{caches}^{k-1}	P_{noc}^{k-1}
$Perf^{k-2}$	P_{cores}^{k-2}	P_{caches}^{k-2}	P_{noc}^{k-2}
$Perf^{k-3}$	P_{cores}^{k-3}	P_{caches}^{k-3}	P_{noc}^{k-3}

Table 1: A sample window (table) of the power budget allocations and corresponding performance numbers (first column).

$$PF = P \times A + \epsilon, \quad \text{where}$$

$$PF = \begin{pmatrix} Perf^k \\ Perf^{k-1} \\ Perf^{k-2} \\ Perf^{k-3} \end{pmatrix}, \quad P = \begin{pmatrix} P_{cores}^k & P_{caches}^k & P_{noc}^k \\ P_{cores}^{k-1} & P_{caches}^{k-1} & P_{noc}^{k-1} \\ P_{cores}^{k-2} & P_{caches}^{k-2} & P_{noc}^{k-2} \\ P_{cores}^{k-3} & P_{caches}^{k-3} & P_{noc}^{k-3} \end{pmatrix},$$

$$A = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix}.$$

After updating the regression history, we use the *least square method* to estimate a_1, a_2 and a_3 values. In the least square method, regression coefficients are computed as: $A = (P^T \times P)^{-1} P^T \times PF$. PEPON then increases the power budget with the largest coefficient and reduces the power budget with the smallest coefficient, since these values reflect the impact of varying the power budget of the corresponding components on the overall performance improvement.

3.2.2 Second-Level Power Distribution

At the second level, we employ different power distribution strategies for cores and caches. To distribute P_{cores} across the different cores in the multicore system, we use the approach proposed in [20]. Specifically, we allocate a power budget of P_i to core i

based on its $\frac{IPS}{watt}$ (performance per watt) over the last epoch ($\frac{IPS}{watt}$ = $\frac{Instruction\ per\ second}{power\ consumption}$). The allocated budgets are proportional to this parameter. In mathematical terms, if IPS_i and Pow_i are the measured IPS and the estimated power consumption of the application running on core i over the last epoch, and Avg_IPS_i is the average IPS of that application when it is running without any power constraint, then the power budget allocated to core i would be computed as:

$$Power\_Budget_{core}(i) = \frac{\frac{IPS_i}{Avg\_IPS_i}/Pow_i}{\sum_{core\ j}(\frac{IPS_j}{Avg\_IPS_j}/Pow_j)} \times P_{cores}.$$

Note that, in this equation, $\frac{IPS_i}{Avg\_IPS_i}$ is used instead of IPS_i since some applications intrinsically has low IPS numbers even when they are executed without any power budget constraint. We can obtain Avg_IPS for each application by running that application on a core with the frequency set to the highest possible value and without imposing any power budget constraint. Note that, this profiling is performed once for each application (not for workloads). Previous work such as [31], also uses the average IPS value as a baseline. In Section 4.2, we present experimental results to show that partitioning power only at the core level (as in the case of [20]) may not be sufficient to maximize performance. Once the power budget of each core is determined, the controller associated with each core attempts to track the budget by adjusting frequency/voltage of the core. As stated before, we use DVFS as our knob to control power consumption of the cores. More details are given later in this section.

To distribute P_{caches} among different cache banks, we employ a *utility-based* strategy. More specifically, the power share of each bank is proportional to its utility over the last epoch. Our goal is to allocate more power budget to the cache banks which are likely to bring more benefits towards the overall system performance. Different strategies can be used to compute utility of cache banks. As an example, the utility of a bank can be computed as the total number of accesses to that bank over the last epoch. However, this would not be a good metric for power budget partitioning since those accesses may have a small cache footprint and it is not probably beneficial to allocate more power budget to the bank with higher utility (note that, more budget means larger size for the banks). The approach adopted in this work is based on *cache footprints*. In this approach, we consider the number of unique cache lines that are hits in a cache bank over the last epoch as the *utility* of that cache bank. To implement this scheme, a small bit vector is associated with each cache bank. This bit vector is used as a hash table. Once a miss occurs in a cache bank, a bit in its bit vector is set to '1'. This bit is indexed using the tag of the new data as the key (the key is hashed using an efficient hardware function). At the same time, the corresponding bit of the evicted data is set to '0'. Consequently, within a period of time, the '1's in the bit vector of a cache bank indicate the number of unique active cache lines that have been accessed. Since a cache line may not be accessed over a long time after the first access, we reset the bit vectors at the beginning of each time interval. In PEPON, the power budget share of each cache bank is proportional to its utility (i.e., the number of '1's in the bit vector) over the last epoch.

Note that, in our system, each core runs a single-application, whereas NoC and cache-banks are shared. Therefore, a pure application-centric budget-partitioning is not possible for an NoC or caches. This is why we take a component-centric partitioning strategy instead.

3.2.3 Design of the Power Controllers

In this subsection, we explain how the power budgets assigned by PEPON are tracked by the controllers associated with different hardware components.

{0.25 , 7}	{0.41 , 0.8}	{0.60 , 0.9}	{0.65 , 0.94}	{0.76 , 1}
{0.84 , 1.05}	{0.93 , 1.1}	{1.0 , 1.14}	{1.23 , 1.28}	{1.33 , 1.34}

Table 2: {frequency (GHz) , voltage (V)} levels considered for the cores.

{0.06 , 0.55}	{0.55 , 0.7}	{0.88 , 0.8}	{1.24 , 0.91}	{1.34 , 0.94}
{1.54 , 1}	{1.7 , 1.05}	{2.0 , 1.14}	{2.23 , 1.21}	{2.46 , 1.28}

Table 3: {frequency (GHz) , voltage (V)} levels considered for the NoC.

Core and NoC Controllers. As stated before, we use the frequency/voltage of the cores and the NoC as our knob to control power consumption of these modules (we employ DVFS). Tables 2 and 3 give the frequency/voltage pairs we consider in our DVFS scheme. A simple *P* (*proportional*) *feedback controller* is configured and associated with each core as well as the NoC to modulate the frequency levels, with the goal of tracking the allocated power budgets. Figure 4 illustrates the high level view of how our controllers operate.[2] We employ *P* controllers in PEPON due to two main reasons: (i) they are relatively easy to design and implement with low overhead since the control output is the control error multiplied by a constant value, and (ii) they are fast to take the system output to the desired value (we need this property in PEPON, since the controllers have only limited number of time intervals to reach the desired power budgets). In this section, based on our system model, we show that employing only the *P* component of a *PID* controller provides a fast and stable controller with zero *steady error*[3].

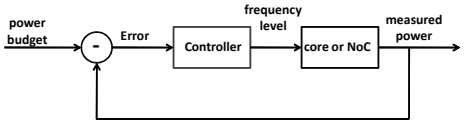


Figure 4: Feedback control loop for regulating the core/NoC power.

At each epoch, the controller compares the current power budget and the measured power over the last epoch and, based on the result of that comparison, adjusts (increases or decreases) the frequency level to *minimize* the measured error value (which is the difference between the power budget and the actual power). Each controller in PEPON implements the following rule to modulate the frequency level:

$$f(T+1) = f(T) + K \times (P_{budget} - P_{current}), \qquad (2)$$

where $f(T)$ is the frequency at T^{th} time interval, K is a constant value, and P_{budget} and $P_{current}$ are the specified power budget and the current power, respectively. The controller first computes $f(T+1)$ and then sets the current frequency level to the level which is the closest one to $f(T+1)$. The controller ensures that its output be one of the available frequency levels in Tables 2 and 3. As can be observed from this control rule, if the measured power is greater than the reference power, the controller reduces the frequency level to reduce power consumption over the next epoch.

The next step to configure the controller is to determine the value of K. This value is computed based on the system power behavior

[2]The *P* controller [10] is a type of controller in which the output of the controller is *proportional* to the error value, which is the difference between the target value and the current system output.

[3]Although most our experiments are performed using a *P* controller, we later report results with a *PID* controller as well.

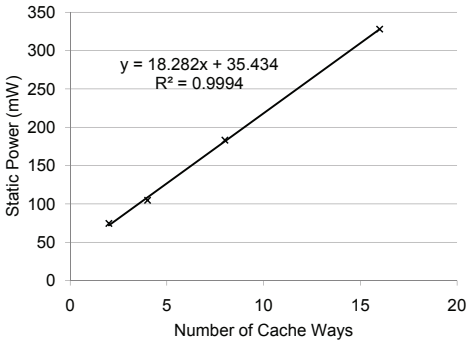


Figure 5: Static power versus the number of active (L2) cache ways.

(core/NoC) when the frequency level is varied. As shown in [24], the impact of varying the frequency on power consumption can be estimated linearly as follows:

$$P_2 = P_1 + a \times \Delta F, \tag{3}$$

where P_1 is the power being consumed at the old frequency and P_2 is the power consumption after the frequency is increased by ΔF. The value of a can be estimated offline by performing experiments with different applications and frequencies and averaging over the obtained values for a. By combining Equations (2) and (3), we observe that, if we set the K value to $\frac{1}{a}$ in the controller, the frequency is changed towards reaching the reference (target) power budget (the value of K is 60M in our scheme).

Control System Analysis. One of the main methods to characterize a control system is to analyze it in the frequency-domain [10]. In this method, the system is represented by a *transfer function* that shows the correlation between its input and output in frequency-domain (*z-transform* is used for discrete time systems). Our system, which is modeled by Equation(3), can be represented as $\frac{Y(z)}{X(z)} = \frac{a}{z-1}$, where $Y(z)$ is the power consumption of the system and $X(z)$ is the variation in the operating frequency in *z-domain*. Since we employ a *P* controller, our closed loop transfer function can be obtained as $H(z) = \frac{Y(z)}{R(z)} = \frac{K \times a}{z-1+K \times a}$, where $R(z)$ is the power budget target.

In formal control theory, a control system is stable and tracks the target over the steady state if the *poles* of the closed loop transfer function is placed inside the unit circle in *z-plane*. The poles of the transfer function are obtained by solving the characteristic equation (setting the nominator of the transfer function to zero). Therefore, as can be observed, our frequency control system has a pole at $z = 1 - K \times a$. Further, as mentioned before, the value of the K parameter is set to $\frac{1}{a}$, which ensures the stability of the system.

Steady state error is another important metric for a control system. This error is defined as the difference between the input reference and the actual system output when the controller reaches its steady state. It can be shown that the steady state error equals $\lim_{z \to 0} \frac{1}{1+H(z)}$ for step reference inputs, where $H(z)$ is the closed loop transfer function. In our control system, the reference inputs are step signals since they vary between two different frequency levels instantly. If we replace $H(z)$ with our system transfer function, one can observe that the steady error would be zero based on the mathematical analysis. Note that, although our analysis shows an accurate control system, there may still be some inaccuracies in tracking the reference targets as will be discussed in our experimental evaluations. These imprecisions can occur because we employ a linear estimation for modeling the system (see Equation(3)) and it does not ideally capture the dynamic behavior of co-runner applications.

Cache Bank Controller. As technology scales down, static (leakage) power of caches becomes an important part of the overall power consumption. The static power highly depends on the size of the caches. As an example, Figure 5 plots the leakage power of a 512KB, 16-way set associative cache manufactured with the 45nm technology, when the number of "active" cache ways varies. This figure clearly indicates that there is a *linear correlation* between the number of active cache ways and the static power consumption. Motivated by this observation, in PEPON, power is controlled in cache banks by turning 'ON' or 'OFF' the cache ways. Specifically, at each epoch, the controller associated with each cache bank compares the current power and the reference power budget and, using the data in Figure 5, decides to turn on or off a number of cache ways, to minimize the difference between the actual and reference power.

3.2.4 Implementation Overheads

PEPON is implemented in software at the OS level with support from hardware. One of the overheads imposed by PEPON is the process of computing the individual power budgets. This includes simple mathematical and matrix operations to partition power at the first and second levels. These overheads are minimal, since the scheme is implemented in software and executed only in every 100-million cycles. In comparison, the controllers are invoked with a higher frequency to track the power budgets. It should be noted, however, that our controllers perform a simple operation (subtraction and multiplication) since they are implemented as *P* controllers. Therefore, the additional overheads caused by our controllers are also not very high. *The power and performance numbers presented in the next section include all the overheads incurred by PEPON*. In addition to these software-level overheads, PEPON assumes that the underlying system has support for DVFS, which is a standard in almost all processors today. We make the same assumption for the DVFS overheads as in [20] (20us DVFS actuation overhead). If the voltage/frequency values are enforced every 20-million cycles (our default value) and if the system operates at 500MHz, the resulting DVFS overhead (20us) would be about 0.05%. In addition, PEPON also assumes that the processor has support for software level control for reconfiguring (turning on/off) cache ways in the L2 cache and tuning the operating frequency of the NoC.

4. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate PEPON, and compare it against alternate power budgeting strategies quantitatively. We use SIMICS [21], which is a full-system simulator for multicores, as our simulation framework. For our experiments, SIMICS is enhanced with detailed latency models as well as power models from CACTI [29], Orion [30] and Wattch [3], to estimate the power consumption of different components in our target multicore architecture.

4.1 Power Estimation Models

Power model for cores. We use the same model as in [24] to estimate power consumption of the cores. The power consumed by a core depends almost linearly on the utility of the core. In other words, once an application starts its execution, the average power consumption of the core can be estimated based on this equation: $P_i = k_0 \times U_i + k_1$, where P_i is the power consumption of core i, U_i is its utilization, and k_0 and k_1 are constant values. Note that the values of k_0 and k_1 may be different for different appli-

cations. In this work, we estimated values of these constants by running each application independently and using Wattch [3] for monitoring power consumption over execution.

Power model for caches. We use numbers obtained from CACTI [29] to estimate power consumption of the caches. Power of each cache bank can be estimated as:

$$Power = Leakage\ Power + (read\ power \times \#\ of\ reads) + (write\ power \times \#\ of\ writes),$$

where the leakage power linearly depends on the cache size. Since, in our scheme, we use the number of active cache ways as the knob to control the power consumption of a cache bank, the leakage power can be estimated considering the active cache size at each time interval over the execution.

Power model for the NoC. The average power consumption of the NoC, over a time interval T, is estimated as follows: $P_{noc} = P_{dynamic} + P_{static} = \frac{NH_{noc} \times E_{noc}}{T} + P_{static}$, where NH_{noc} is the total number of hops that all the packets in the network traverse over T and E_{noc} is the energy per hop. In this expression, "energy per hop" is the amount of energy consumed when a packet passes through a router. We used the energy values obtained from Orion [30] to estimate the total power consumption of the NoC.

Processors	16 cores with private L1 data and instruction caches
NoC Architecture	4×4
Processor Model	4-way issue superscalar
Private L1 Data and Instr Caches	Direct mapped, 32KB, 64 bytes block size, 3 cycle access latency
Number of L2 Cache Banks	16 (distributed over the network)
L2 Cache	64 bytes block size, 10 cycle access latency
L2 Cache Bank Size	512KB per core
Memory	4GB, 200 cycle off–chip access latency
Power Budget Control Enforcement Interval	100 Million cycles
Tracking Control Enforcement Interval	20 Million cycles
δ	1

Table 4: Baseline configuration.

Table 4 gives the baseline configuration we use in our experimental evaluation. In our baseline configuration, the power budget allocated to each component by PEPON is computed and enforced every 100-million cycles. Further, the controller assigned to each component has the opportunity to track its power target over five intervals with 20-million cycles duration. We form six application mixes (workloads) using the SPEC CPU 2006 benchmark suite [1] for our experimental evaluation. Each of our application mixes consists of 16 applications. We consider one-to-one mapping, that is, in each experiment, 16 applications of a mix are mapped and executed on 16 different cores in our system (one application per core). These six workloads are given in Table 5. In forming these workloads, we tried to have diversity in terms of memory access behavior. For each application, we skip the first 1-billion cycles for warm-up, and then the simulation continues with the next 10-billion cycles (100 power budget enforcement intervals).

4.1.1 Schemes for Performance Evaluation

Table 6 lists the five different schemes we consider in our experimental evaluation to compare with PEPON. In this table, "*dynamic*" indicates that the corresponding power budget is varied over the execution. As can be observed from Table 6, all the entries are shown as "*dynamic*" for PEPON, indicating that the power budgets are varied over the execution in *both* the first and second levels of the budget partitioning. However, in other schemes (Schemes 1 through 4), the power budgets are fixed at the first level of partitioning during runtime (i.e., P_{cores}, P_{caches} and P_{noc} are fixed as given in Table 6). This table also gives the actual power budget values (for the high and low power modes) assigned when using the different partitioning schemes. However, in Scheme 1, the total power budget of the cores and the cache banks are still partitioned among the individual components dynamically. All the power budgets of different components are fixed in Scheme 2, and the total power budget of the cores and the caches are evenly distributed among them. In Schemes 3 and 4, on the other hand, only one of the power budget distribution mechanisms is enabled at the second level (for cores in Scheme 3 and for caches in Scheme 4). Note that, Scheme 3 mimics a recently-proposed approach to power budget partitioning in multicores [20]. In addition to these schemes, we also implemented another scheme in which at the first level, the total power budget is distributed dynamically and at the second one, the power budgets of the cores and cache banks – from the first level – are partitioned statistically (evenly) among the individual components (cores and caches). We observed that PEPON improves the performance over this last scheme by about 8% on average. The detailed results for this scheme is omitted due to space concerns.

4.2 Results

4.2.1 Performance Results

We performed our experiments under two power modes: (i) *Low power mode.* In this case, the total power budget of the chip is set to be 40 watts, and (ii) *High power mode.* The total power budget is set to be 90 watts in this case. These power budgets are partitioned in two levels by PEPON over the course of execution. For the other schemes (Schemes 1-4) in Table 6, the total power budgets at the first level of the power budget partitioning which are fixed during the runtime, to be about 80% and 40% of the maximum power value in each case as given in Table 6. Figure 6(a-b) plots the normalized weighted speedup values achieved for different mixes when employing the schemes listed in Table 6. The weighted speedup for n applications is computed as follows:

$$Weighted\ Speedup = \sum_{i=0}^{i=n-1} \frac{IPS^i_{scheme}}{IPS^i_{alone}},$$

where IPS^i_{scheme} is the achieved instruction per second (IPS) by the employed scheme for application i and IPS^i_{alone} is the measured IPS achieved when the application is executed alone without any power constraints. The weighted speedup values in Figure 6 are normalized to *Scheme 2* (assumed to be the *base scheme*) in Table 6, where power budgets are not modulated over the execution (i.e., all power distributions are fixed throughout the execution). Note that, we consider IPS instead of instruction per cycle (IPC) to compute the speedup values since we use the operating frequency as a knob to control the power consumption of the cores and the IPC value does not change by varying the core frequency.

Figures 6(a) and (b) show that PEPON achieves significant performance (workload speedup) improvements in both the low and high power modes (more than 25% in the low power case and more than 20% in the high power case). Further, the total power consumptions track the power budgets quite closely as shown in Figures 6(c) and 6(d). These plots reveal that partitioning the total power budget at the first level by PEPON plays a significant role in achieving better performance under a certain power budget. As an example, as can be seen from Figure 6(a), although performance is improved by partitioning the power budget at the second level among the cores and cache banks in Mix-2 (Schemes 1 , 3, and 4),

Workload	Applications
Mix-1	*mcf, soplex, libquantum, GemsFDTD, lbm, leslie3d, milc, sphinx3, xalancbmk, cactusADM, mcf, soplex, libquantum, GemsFDTD, lbm, leslie3d*
Mix-2	*gobmk, sjeng, gcc, wrf, dealII, namd, perlbench, calculix, tonto, povray, povray, tonto, calculix, perlbench, namd, dealII*
Mix-3	*mcf, soplex, libquantum, GemsFDTD, lbm, leslie3d, milc, sphinx3, xalancbmk, cactusADM, omnetpp, astar, hmmer, bzip2, h264ref, gromacs*
Mix-4	*gobmk, sjeng, gcc, wrf, dealII, namd, perlbench, calculix, tonto, povray, omnetpp, astar, hmmer, bzip2, h264ref, gromacs,*
Mix-5	*mcf, soplex, libquantum, GemsFDTD, lbm, leslie3d, milc, sphinx3, xalancbmk, cactusADM, gobmk, sjeng, gcc, wrf, dealII, namd,*
Mix-6	*gobmk, sjeng, gcc, wrf, dealII, namd, perlbench, calculix, tonto, povray, cactusADM, xalancbmk, sphinx3, milc, leslie3d, mcf,*

Table 5: Various mixes of applications considered for our multiprogrammed workloads.

Scheme	$\sum CorePowerBudget(i)$	$CorePowerBudget(i)$	$\sum BankPowerBudget(i)$	$BankPowerBudget(i)$	$NoCPowerBudget$
PEPON	dynamic	dynamic	dynamic	dynamic	dynamic
Scheme1	static (29 watt – 68 watt)	dynamic	static (5 watt – 10 watt)	dynamic	static (6 watt – 12 watt)
Scheme2	static (29 watt – 68 watt)	static (29/16 watt – 68/16 watt)	static (5 watt – 10 watt)	static (5/16 watt – 10/16 watt)	static(6 watt – 12 watt)
Scheme3	static (29 watt – 68 watt)	dynamic	static (5 watt – 10 watt)	static (5/16 watt – 10/16 watt)	static (6 watt – 12 watt)
Scheme4	static (29 watt – 68 watt)	static (29/16 watt – 68/16 watt)	static (5 watt – 10 watt)	dynamic	static (6 watt – 12 watt)

Table 6: Various experimental schemes with the fixed power budget values.

it is less than the improvement achieved by PEPON since the budgets are fixed at the first level in all those schemes. We also note that there is a significant difference between PEPON and Scheme 3 (which mimics a recently proposed scheme [20]), further emphasizing power budgeting at *both* the levels.

Figures 6(a) and (b) also show the average speedup values achieved by different schemes across all the workloads. One can observe that, PEPON improves the overall workload performance significantly as compared to the other schemes tested in both high and low power modes (about 21% in the low power mode and 17% in the high power mode). Also, as can be seen, the performance improvement achieved by Scheme 1 is higher than the other schemes (Schemes 2, 3 and 4). This is because this scheme takes advantage of partitioning the power budgets among both the cores and cache banks at the second level, whereas in all other schemes tested (except ours), power budgets (among the cores or caches or both) are fixed at the second level.

4.2.2 Dynamics of PEPON

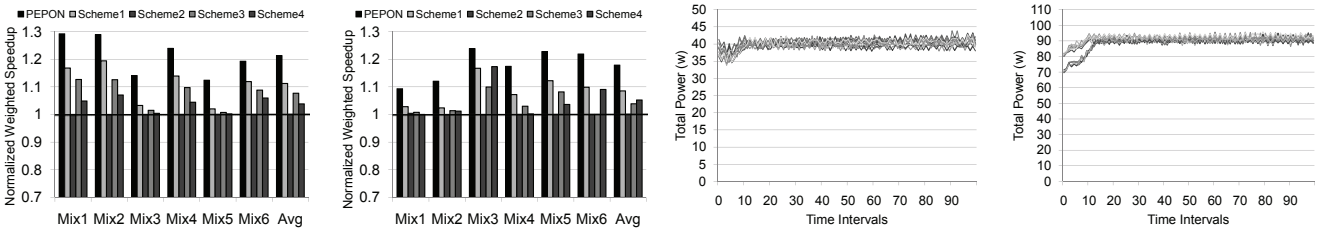
We next focus on how the total power budget is dynamically partitioned by PEPON over the course of execution to achieve the performance enhancements shown in Figure 6. Here, we present the results for one of our workloads (Mix1); the remaining workloads exhibit similar behavior.

Figures 7(a)-(c) plot the power budgets assigned to cores, caches and NoC by PEPON during runtime for Mix-1, when the total power budget is set to 40 watts. These figures also plot the measured power consumption for different components. As can be observed, PEPON first distributes the total budget at the first level and then, the budget targets are successfully tracked at each group. The accumulated power consumptions of the cores, caches and NoC are plotted in Figure 7(d) (this chart shows the total power consumption and *not* the total power budget). It is important to note that the power budget violation experienced when using PEPON is very low (less than 2%); the power budget violations will be further discussed later. There are two other important points in Figures 7(a)-(c) that should be noted: (i) the power consumption of the cores and NoC (Figures 7(a) and (c)) oscillate around the target power budgets and the budgets are slightly violated over the execution, whereas the power consumption of the caches is always below the budget, and (ii) the power budget of the caches is tracked faster. As an example, at time '0', as shown in Figures 7(a) and (c), the power budgets of the cores and NoC are not the same as their power consumptions, and the power budgets are tracked after some intervals. The main reason for these two differences is that the type of the controllers employed for the caches are different from the ones associated with the cores and NoC. As stated before, in the caches, power consumption is dominated by the static power and the static power exact model is known (the correlation between the number of active ways and power consumption). Therefore, the controllers associated with the caches are able to select the correct number of active cache ways based on the model in a short time, such that the power budget is not violated. On the other hand, we employ *P* controllers for the cores and NoC in which the exact model is not known and the frequencies are determined based on the *error* values, leading to small oscillations.

Figure 8 plots the power budget distribution at the first level for Mix-1 (similar to Figure 7) but when our total budget is 90 watts (high power mode). As one can observe, the power budget is successfully partitioned and tracked when we are operating at high power mode, too. We observed the same dynamic behavior at the first level of power budget partitioning from our other mixes as well. Note that, the NoC power in these experiments is controlled directly by modulating the NoC frequeny/voltage level. However, the power budgets of the cores and caches are not directly controlled and are distributed further among the individual components. Therefore, tracking the total power budgets of the cores and caches at the first level (as shown in this set of figures) implies that each individual component (cores and caches) has been successful in tracking its own budget.

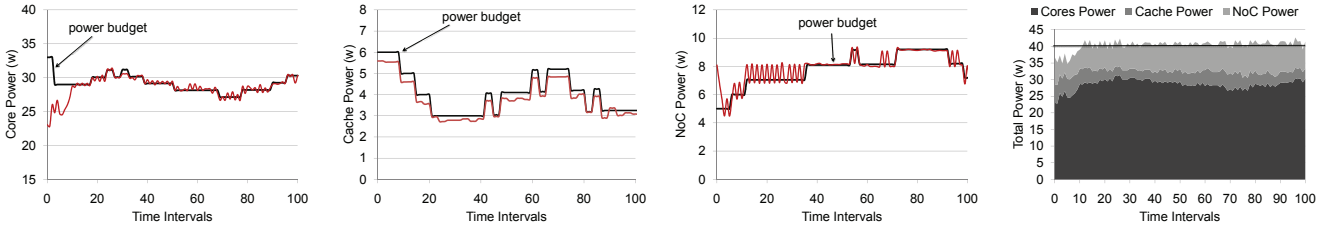
Figures 9 illustrates how the power budgets determined at the first level are partitioned among the individual components (cores and caches) at the second level over the execution for Mix-1. As can be observed from each of these plots, the total power budget is partitioned into 16 pieces, and the size of each piece indicates the amount of power budget allocated to each component. Different components receive different shares from the power budget since the total budgets are distributed based on the utility of the caches or the performance power ratio of the cores over the recent time interval (the schemes that we employ at the second level). The controller configured for each hardware component tracks the target power budget by adjusting the voltage-frequency and the number of active cache ways of the cores and caches, respectively. To better understand these power distribution curves, we plot in Figure 10 the selected frequencies for the cores and the number of cache ways for Mix-1. We see that the values of these knobs are also different for different components at a specific time. As an example, at time 10, the frequencies assigned to some of the cores are low as compared to the others. This is because, the power budgets are not the same for different cores. Note also that even if the budgets were the same, the different cores would consume different amounts of energy when they operate at the same frequency since they execute different applications.

Controller Selection. As discussed before, we employ simple and fast *P* controllers to modulate the frequency of the cores and the



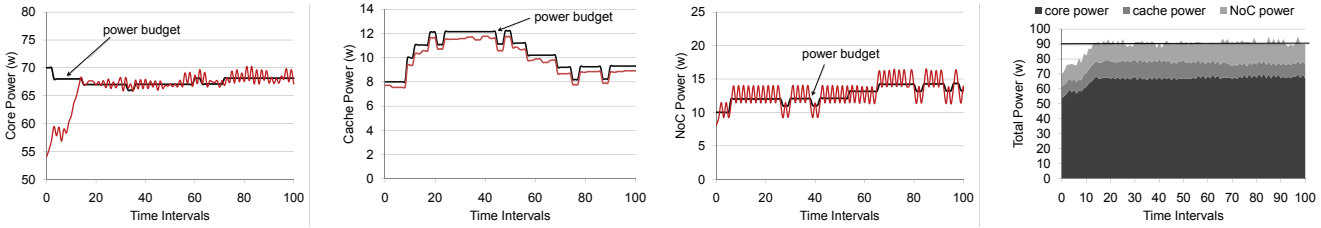
(a) Low power mode (b) High power mode (c) Low power mode (d) High power mode

Figure 6: (a-b) Normalized weighted speedup values achieved by different schemes under certain power budget, and (c-d) Dynamic tracking of the total power budget. Note that both (c) and (d) plot all the mixes together. The point here is that, in the low power case, chip-wide power consumptions of our mixes approach 40W, and similarly, in the high power case, chip-wide power consumptions of our mixes approach 90W.



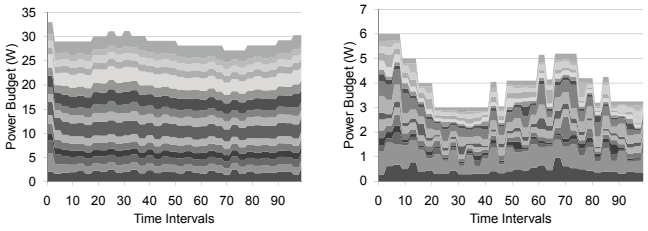
(a) Core power. (b) Cache power. (c) NoC power. (d) Total power consumption (not power budget).

Figure 7: Power budget partitioning and tracking at the first level of PEPON for Mix-1 (low power mode).



(a) Core power. (b) Cache power. (c) NoC power. (d) Total power consumption (not power budget).

Figure 8: Power budget partitioning and tracking at the first level of PEPON for Mix-1 (high power mode).



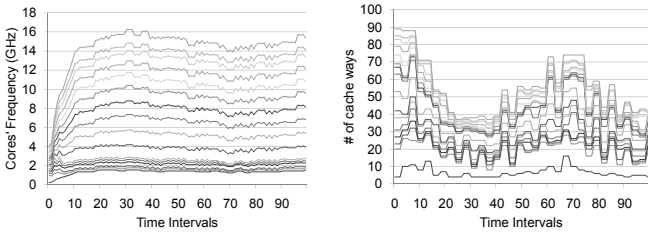
(a) Cores' power budgets (low power mode). (b) Caches' power budgets (low power mode).

Figure 9: Power budget partitioning at the second level (Mix-1). Note that each graph plots all 16 core (or cache) components.

NoC to track the specified power budgets. Instead of a *P* controller, one may suggest to employ a more complicated *PID* controller for frequency modulation. One of the main advantages of employing *PID* controllers is that they can drive steady-state errors to zero. As discussed before, in our system, this error is already zero when employing *P* controllers. Figure 11 plots how the power budgets (which is assigned to one of the cores in Mix-1 experiment) are tracked by *P* and *PID* controllers over the course of execution. As can be observed from this figure, the *P* controller is able to track the targets as well as the *PID* controller. However, the *PID* controller slightly reduces the number of oscillations at some time intervals (e.g., at 30th time interval). On the flip side, the *P* controller has the capability to respond more quickly to the target variation (e.g., at 70th time interval). In addition, as compared to the *PID* controller, the *P* controller incurs less overhead.

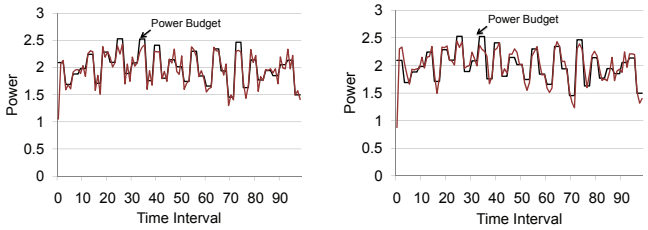
4.2.3 Sensitivity Experiments

We also measured sensitivity of Mix-1 (under the low power mode) to different parameters. As stated before, at each epoch, the power budgets are increased and decreased by δ and $\delta - \epsilon_1$, respectively at the first level of power budget partitioning tree. Recall from Table 4 that the default value for δ is 1 in our experimental evaluations. Figure 12 plots the speedups achieved for Mix-1, when the δ value is set to 2 (the results with a value of 1 are reproduced for ease of comparison). We see that increasing the δ value has a negative impact on the achieved speedup by PEPON. This is because, the power budgets may oscillate around the optimal values and not reach the optimal ones by having a larger δ.



(a) Cores' frequencies (low power mode). (b) Number of caches' ways (low power mode).

Figure 10: Modulation of core frequencies and the number of active cache ways over execution (Mix-1). Note that, the curves are stacked on top of each other for better visibility. Note also that, time intervals where the cumulative number of (cache) ways activated is low correspond to periods where caches are not allocated much power.



(a) P controller (b) PID controller

Figure 11: Employing P and PID controllers.

Our experiments with different cache sizes show that our speedups are not very sensitive to the cache bank sizes, since in this case, due to the low budget allocated to the cache banks, even when more cache space is available, only a portion of them is used during runtime. One of the parameters that affects the performance improvements achieved by PEPON is the frequency at which the control decisions are made and enforced (our epoch sizes). Our experiments show that the speedup achieved for small epoch sizes is generally higher. This is because, in this case PEPON is invoked more frequently, and the behaviors of the running applications are monitored with a higher frequency; as a result, we may end up achieving better performance improvements. However, this also means more overhead.

4.3 Power Budget Violations

As can be observed from Figures 7(d), 8(d), the overall power budget could not always be tracked with 100% accuracy and is violated over the course of execution (although this violation is in general very low). Also, in some cases, the budget is not fully utilized. This is because the power consumption of the individual components at the bottom of the partitioning tree (see Figure 3(a)), is not always exactly the same as their power budgets. There are two main reasons for this: (i) the knobs used by the controllers can only take values from a set of discrete levels and cannot be varied continuously. As an example, the frequency/voltage of a core can only be set to the one of the discrete values selected from Table 2. Consequently, the frequency chosen by the controller may not be the exact one required to reach the power budget; and (ii) when the target power budget of a controller is modulated, it takes time for the controller to track the new target and, during that time, the output power may fluctuate. Even if the budget is fixed, the controller needs to compensate for the potential change in the behavior of the

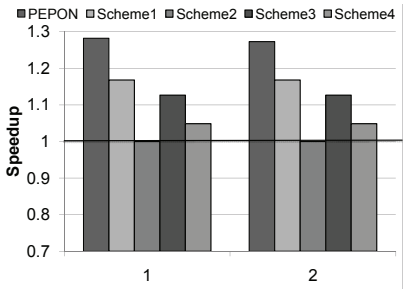


Figure 12: Performance improvements achieved for Mix-1 by varying the default values of configuration parameters.

running application. One of the methods using which these over- and under-utilizations of the available power budget can be handled is through dynamically changing the reference budgets. For example, if we under-utilize the available budget in the previous interval by $x\%$, we could increase the reference budget value by $x\%$. Investigating this and similar compensation schemes is in our future research agenda.

5. RELATED WORK

Control theory as a tool for power management has recently gained popularity but most prior control centric works do not account for chip-wide power management that takes into account cores, on-chip caches, and the underlying NoC. We have already quantitatively compared our work with [20] and show that our proposal fares better since it is a more holistic approach that considers the on-chip cores, caches and the NoC. In this section, we discuss the most relevant prior works to our proposal.

5.1 Feedback Control Based Power Management

[24] proposes a control theoretic framework for power management in multi-clock domain architectures, where each domain is controlled by a separate clock. In this work, a two-tier power management architecture is proposed, where the first tier allocates a target budget to each local island and the second tier controls the power. The solution becomes simpler in this case since it is implicitly assumed that all cores (and their caches) in a domain would be running *similar* applications, i.e., all cores in a particular domain would either host CPU intensive or memory intensive applications. This allows for DVFS of an entire domain using the per domain clock. Compared to this architecture, our solution targets a realistic CMP with cores, shared caches and an NoC, where domain level clock management is prohibitive. To tackle this, our proposal has a two level power control: The first level allocates power budget among the cores, shared caches and the NoC, and the second level redistributes the core power among different cores, and the cache power among different caches. Other related works advocating control theory for power and thermal management include [6, 15, 31, 32, 33]. In [6], a PI-based core thermal controller is proposed and an outer control loop is used to decide process migrations on thermal emergencies. The proposal in [15] extended [33] and proposes a distributed version of the scheme, applying the basic mechanics of the formal approach to CMPs. In [31], a hierarchical control architecture is proposed that controls the total power consumption of a large-scale data center to stay within a constraint imposed by its power distribution capacity. This work, however, only models cores in a processor when it comes to power management. In [32], Wang et al. propose a multi-input multi-output solution to CMP temperature management. One major difference between our proposal and the work in [32] is that, while our approach is simple

and flexible (in fact flexibility policy was one of the prime design time consideration behind our proposed approach), [32] use sophisticated controllers whose complexity is proportional to the number of cores in the chip. Consequently, it is hard to apply the same to really large CMPs. Further, when compared to the work in [20], our proposal targets all major components in a CMP (cores, caches and NoC), whereas [20] only targets power control in cores.

5.2 Ad-hoc Power and Thermal Management

There are many prior proposals focussing on the design of power and thermal management schemes [5, 9, 13, 18, 22, 25, 28]. These works highlight the importance of adapting to workload behavior under power and thermal constraints. However, most of these works adopt a combination of independent local schemes in the architecture or scheduling decisions in the operating system. Meng et al. [22] proposed an adaptive, multi-optimization strategy for multicore power management and addressed the problem of meeting a global chip-wide power budget through run-time adaptation of configurable processor cores. Isci et al. [13] also used a global power manager, similar to ours, that provides a mechanism to sense the per-core power and performance state of the chip at periodic intervals; it then sets the operating power level or mode of each core to enforce adherence to known chip-level power budgets. However, the local monitoring used to enforce these power budgets were based on open loop control and does not provide the same robustness as the formal feedback control in our work. Furthermore, the constraints on the achievable performance were less stringent as they perform per-core DVFS as opposed to core, cache and NoC as we do in our work. Prior proposals like [2, 8, 27] have looked at energy saving mechanisms for cache alone by dynamically reconfiguring the cache hierarchy or partitioning a shared cache among multiple applications. Works such as [7] have investigated memory cell sizing for cache energy efficiency. Our work also uses cache reconfiguration for energy management, but we do so as part of managing the energy consumption of an entire CMP rather than the on-chip caches alone.

6. CONCLUDING REMARKS

The main contribution of this paper is a two-level power distribution strategy oriented towards maximizing system performance under a specified power budget. The results collected from our experiments with 6 workloads formed using 16 applications selected from a set of 30 applications clearly demonstrate the importance of power distribution at both the levels. Specifically, our experimental results indicate that, when using PEPON, overall performance can be improved by as much as 29% over the case when no power distribution is adopted. Further, our results also show that distributing power only at a core level (as done in a recent study [20]) may not be sufficient for maximizing performance.

7. REFERENCES

[1] `http://www.spec.org/cpu2006/`.

[2] R. Balasubramonian, et al. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *MICRO'00*.

[3] D. Brooks, et al. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA'00*.

[4] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *ISLPED'00*.

[5] J. D. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In *PACT'05*.

[6] J. Donald and M. Martonosi. Techniques for multicore thermal management: classification and new exploration. In *ISCA'06*.

[7] R. G. Dreslinski, et al. Reconfigurable energy efficient near threshold cache architectures. In *MICRO'08*.

[8] S. Dropsho, et al. Integrating adaptive on-chip storage structures for reduced dynamic power. In *PACT'02*.

[9] M. Gomaa, et al. Heat-and-run: leveraging smt and CMP to manage power density through the operating system. In *ASPLOS'04*.

[10] J. L. Hellerstein, et al. *Feedback Control of Computing Systems*. 2004.

[11] Y. Hoskote, et al. A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro*, 27:51–61, September 2007.

[12] J. Howard, et al. A 48-Core IA-32 Message-passing processor with DVFS in 45nm CMOS. In *ISSCC'10*.

[13] C. Isci, et al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO'06*.

[14] N. K. Jha. Low power system scheduling and synthesis. In *ICCAD'01*.

[15] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark. Coordinated, distributed, formal energy management of chip multiprocessors. In *ISLPED '05*.

[16] J. Kim and M. Horowitz. Adaptive supply serial links with sub-1v operation and per-pin clock recovery. In *ISSCC'02*.

[17] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley, 2003.

[18] J. Li and J. F. Martínez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *HPCA'06*.

[19] S.-H. Lim, et al. Migration, assignment, and scheduling of jobs in virtualized environment. In *ATC'11*.

[20] K. Ma, et al. Scalable power control for many-core architectures running multi-threaded applications. In *ISCA'11*.

[21] P. S. Magnusson, et al. Simics: A full system simulation platform. *Computer*, 2002.

[22] K. Meng, et al. Multi-optimization power management for chip multiprocessors. In *PACT'08*.

[23] Y. Meng, et al. On the limits of leakage power reduction in caches. In *HPCA'05*.

[24] A. K. Mishra, et al. CPM in CMPs: Coordinated power management in chip-multiprocessors. In *SC'10*.

[25] M. Monchiero, et al. Power/performance/thermal design-space exploration for multicore architectures. *IEEE Trans. Parallel Distrib. Syst.*, 19:666–681, May 2008.

[26] U. Y. Ogras, et al. Voltage-frequency island partitioning for GALS-based networks-on-chip. In *DAC'07*.

[27] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO'06*.

[28] J. Sharkey, et al. Evaluating design tradeoffs in on-chip power management for cmps. In *ISLPED '07*.

[29] S. Thoziyoor, et al. CACTI 5.1. *Technical Report, Hewlett-Packard Company*, 2009.

[30] H. Wang, et al. Orion: A power-performance simulator for interconnection networks. In *MICRO'02*.

[31] X. Wang, et al. SHIP: Scalable hierarchical power control for large-scale data centers. In *PACT'09*.

[32] Y. Wang, et al. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA'09*.

[33] Q. Wu, et al. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *SIGARCH Comput. Archit. News*, 32:248–259, October 2004.