Arabian Journal for Science and Engineering (2018) 43:7343–7358
https://doi.org/10.1007/s13369-018-3079-4

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Machine Learning-Based Energy Optimization for Parallel Program Execution on Multicore Chips

Mwaffaq Otoom[1] · Pedro Trancoso[2] · Mohammad A. Alzubaidi[1] · Hisham Almasaeid[1]

Received: 21 June 2017 / Accepted: 4 January 2018 / Published online: 29 January 2018

Abstract
Energy is increasingly becoming the major constraint in designing multicore chips. Power and performance are the main components of energy and are inversely correlated. In this paper, we study the energy optimization of multicore chips that process parallel workloads using either power or performance optimization. To do so, we propose novel machine learning-based global and dynamic power management controller. The controller is used either to maximize performance within a fixed power budget or to minimize the consumed power to achieve the same baseline performance. The controller is also scalable, as it does not incur significant overhead as the number of cores or demands increases. The technique was evaluated using the PARSEC benchmark suite on a full-system simulator. The experimental results show that our global power controller outperforms, in terms of the EDP metric, the non-DVFS baseline by 28 and 35.5%, when optimized for performance and power, respectively. This suggests that optimizing power is more related to energy efficiency than optimizing performance.

Keywords Dynamic power management · Dynamic voltage/ frequency scaling · Machine learning · Multicore · Scalability

1 Introduction

Power consumption was the major reason leading to the architecture shift toward multicore chips, as a way to manage the demand for increasing frequency. But, to keep up with the performance demands, more and more cores are added to the processors. As a result, power is again a key design issue; in particular, the challenge now is to manage a large number of cores as to deliver the most performance at the least power consumption. Thus, improving the performance directly through frequency or indirectly through increasing chip resources such as cores or even cache memories and communication bandwidths would result in more power consumption. Energy is the by-product of performance and power requirements; thus, it is appealing to study this relationship.

The problem of managing the power consumption has been thoroughly studied resulting in breakthrough techniques such as dynamic voltage/frequency scaling (DVFS) and dynamic power management (DPM) and many other algorithms based on these techniques that directly target power reduction such as [1,2]. In contrast, other studies have indirectly targeted the energy efficiency through performance improvement such as [3,4]. Up to our knowledge, none has considered the relationship between power and performance using a comparison between the two optimization approaches. In this paper, we study the relationship between power and performance by optimizing each one individually to improve the energy efficiency of a multicore chip that process parallel workloads. To do so, we extend our work in [5], where we proposed a power management technique based on performance optimization, whereas in this work we propose a power management technique that can be used to do either power or performance optimization.

Recently, machine learning-based DVFS and DPM approaches have been used as to tackle the complex issue of reducing the power consumption of multicore chips [6]. These techniques have been successfully applied at the core level. Nevertheless, such strategies are not able to scale gracefully to the chip level, where the management needs to be applied to all cores in a coordinated way. In particular, the overheads of the technique would be unacceptable with the

✉ Mwaffaq Otoom
mof.otoom@yu.edu.jo

[1] Computer Engineering Department, Yarmouk University, Irbid, Jordan

[2] Computer Science Department, University of Cyprus, Nicosia, Cyprus

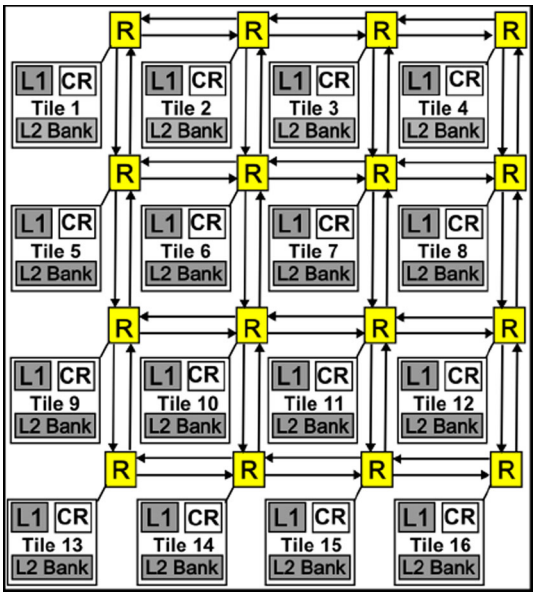


Fig. 1 Multicore chip that consists of 16 tiles connected in an 4×4 MESH NoC architecture, used for the evaluation of our proposed techniques

increasing number of cores [7]. Reinforcement learning (RL) was used in previous work as a technique to help manage the power consumption, as it is adaptive and requires no prior knowledge [8,9]. However, none of this work has considered allocating power among chip tiles or clusters *dynamically* for the purpose of either improving the performance or saving power, while maintaining the scalability of the system.

In our proposed power management technique, an RL-based power agent monitors each multicore chip tile, where tiles are connected via an MESH Network on Chip (NoC) architecture, shown in Fig. 1. The agent dynamically learns which power policy (V/F) achieves either:

- the maximum performance for the tile, in the case of performance optimization, or
- the baseline performance for the tile, in the case of power optimization

The major contribution of this work is to study the relationship between power and performance in the context of multicore chips that process parallel workloads. This paper also makes the following contributions:

- Global adaptation of core power level. Based on local learning, the intelligent controller allows for the exchange of power between cores, thus providing a solution that is more flexible and closer to optimal.
- Dynamic adaptation of the core power level. The learning is done continuously during the execution of the application, and actions are taken at fixed quantum time intervals. This allows for the power to change dynamically as to satisfy changes in workload behavior.
- Scalable algorithm. The proposed technique is based on a local learning agent that incurs low overhead and an intelligent controller that is scalable, with the increase in the number of chip resources.

The rest of this paper is organized as follows. In Sect. 2, we review the literature on the performance and power optimization, where we focus in particular on the usage of RL in DVFS and DPM. In Sect. 3, we describe the methodology of our work. Results and discussion are presented in Sect. 4. Finally, we conclude our paper in Sect. 5.

2 Related Work

In this section we survey existing work on power and performance optimization. We pay more attention at the machine learning-based techniques done for such kind of optimization.

2.1 Power Optimization

There are numerous researches on power optimization. This research can be broadly classified into software and hardware solutions. Software solutions are those that use intelligent scheduling and learning techniques to optimize DVFS and DPM decisions, whereas hardware solutions are mainly concerned with designing hardware components or processors that consume less power. While the power optimization component of our proposed algorithm is a software solution, for the sake of completeness in this section we survey the most recent advances in both classes.

We first survey some existing software solutions. For instance, Kianzad et al. [10] integrate DVFS with genetic-based scheduling strategy, in order to meet deadlines in embedded systems and to save power, whereas our approach is more general in that it uses Q-learning to improve performance in terms of execution time, as well as power consumption. Hua et al. [11] studied the trade-off of quality of service and power consumption in multimedia applications. Our approach, in contrast, considers all application types. Choi et al. [12] proposed a DVFS method that is fed with data about memory accesses and their relation to CPU frequency and voltage levels. Gheorghita et al. [13] proposed a technique for saving power by classifying the modes of execution to guide power adaptation at runtime. Xian et al. [14] presented a technique to save energy by choosing the best scheduling policy for the task type. Yang et al. [15] proposed a technique for scheduling concurrent tasks onto a heterogeneous multiprocessor in a way that saves energy. Kim et

al. [16] presented an offline approach to find the optimal frequency and voltage settings. Hua et al. [17] studied the optimal number of voltage levels, in order to save overhead. In contrast to [12–17], our algorithm learns these information, and more, at runtime.

Other software techniques consider the definition of power management modes at different levels in computing. For instance, Li et al. [18] proposed a method for selecting the power modes for the optimal power management. Hoeller et al. [19] proposed an interface for power management for both hardware and software elements. Huang et al. [1] proposed an energy-saving technique which dynamically controls the power mode of the embedded system according to the history of tasks arrival. Bhatti et al. [20] presented an online framework that uses machine learning approach to dynamically select the best performing policy for any given workload. Niu et al. [21] proposed a technique to save both leakage and dynamic power by using both DVFS and power modes. Kim et al. [22] proposed a technique which partitions the task execution into several intervals and shuts down the unneeded components, in each interval. Similarly, Shin et al. [23] exploited slack for saving energy, by using an offline and online DVFS approaches. Our approach is different than the work in the above-referenced software solutions in that it uses multitier power management that considers the scalability issue. Jha et al. [2] proposed a two-tier hierarchal power management technique that controls power at the tile-level and the last-level cache and supports thread migration. While the work in [2] used hierarchal power management, our approach distributes power allocations dynamically.

On the other hand, hardware solutions focus mainly on designing power-aware memories. For instance, Yang et al. [15] discuss a technique for saving main memory energy. Trajkovic et al. [24] proposed a buffering-based technique for saving energy, by adapting write-combining and pre-fetching schemes for each application. Reddy et al. [25] presented an approach that selects best cache partitioning for different running applications in an offline manner and uses this information at runtime. Tsai et al. [26] proposed a technique for saving energy by using a new memory structure called Trace Reuse Cache (TRC). Hajimiri et al. [27] integrated cache reconfiguration and code compression to improve both performance and energy efficiency. Albonesi [28] proposed selective-way approach in cache memories, where some of the ways are turned off to save energy, while maintaining reasonable performance levels. Zhang et al. [29] proposed highly configurable cache architecture to save energy. Kin et al. [30] proposed another technique by filtering accesses to cache. Steinke et al. [31] proposed another technique by utilizing scratchpad memories. Benini et al. [32] discussed the mapping of the most frequently accessed locations onto a small memory that can be placed on chip, thus save memory. Bournoutian et al. [33] presented another technique by reducing L1 cache misses. Recently approximate computing has emerged to sacrifice performance for the sake of saving energy by designing lower-performance hardware components [34]. Moreover, power optimization has been considered in designing hardware such as GPUs, FPGAs, ASICs and DSPs [35,36]. While these approaches tackled the increasing cost problem of hardware solutions, they sacrificed the system performance. Moreover, hardware solutions are static solutions that cannot adapt to the different execution scenarios, in contrast to machine learning-based software solutions.

2.2 Performance Optimization

Other studies have reversely targeted energy through optimizing performance. These studies can be classified into centralized and decentralized techniques for multicore chips.

Examples of centralized techniques include the following studies. Sharifi et al. [3] proposed a power budget distribution algorithm for NoC-based multicores, PEPON, at both the chip and resource levels in order to maximize workload performance within the specified power budget. At the chip level, specific portions of the total power budget are distributed among cores, networks and memories. Then at each resource level, each portion is distributed among the different instances of the same type of that resource. This distribution is changing every specific amount of time by a feedback controller that uses mathematical equations to adapt frequency levels to the changes in the workload to achieve high performance. Ma et al. [37] designed a scalable three-layer approach to maximize the performance of a multicore chip that processes a mix of single and multithreaded workloads, within a fixed power budget. This approach targets chip cores only, neglecting the power of memories and communication resources. Isci et al. [4] proposed a global power manager to maximize performance that senses the per-core power and performance of the multicore chip at periodic intervals. Based on that, it dynamically sets the power mode of each core to not exceed the assigned chip power budget. Wang et al. [38] proposed a global power control algorithm that uses control theory to maximize performance. Mishra et al. [39] proposed a two-tier power management approach for chip multiprocessors. At the first tier, a global power manager provides power to individual voltage/frequency islands. At the second tier it regulates island power consumption using DVFS to dynamically adapt to workload changes. The proposed approach is illustrated using performance-aware, thermal-aware and variation-aware power provisioning.

On the other hand, examples of decentralized techniques include the following studies. Sartori and Kumar [40] proposed a decentralized power management technique for many core architectures that maximizes performance within a peak power level by placing several more cores on a

die. Winter et al. [41] analyzed different scheduling and power management algorithms for effectiveness and scalability without affecting the power–performance optimization. For example, Hanson et al. [42] proposed a runtime manager, PET, feasible for single processor to optimize for multiple constraints, namely performance, power, energy and temperature, by continuously monitoring the system and choose the appropriate power state that is mainly a v/f setting. They demonstrated how PET maximizes performance within power and thermal budgets by dynamically applying DVFS settings at runtime on an Intel Pentium M system to adapt to the dynamic nature of the SPEC CPU2000 suite workload behavior. Cochran et al. [43] proposed a technique, named Pack & Cap, for datacenters and HPC clusters to maximize the performance within variable power caps running PARSEC parallel, multithreaded workloads. The Pack & Cap manages V/F settings and thread packing in order to maximize performance within a fixed power budget. Etinski et al. [44] proposed a job scheduling policy that maximized overall job performance within a fixed power budget.

The performance optimization component of our proposed algorithm uses a hybrid approach, where a central controller dynamically distributes chip power budget among tiles and a local agent that learns the best power allocation for its tile.

2.3 Machine Learning

RL has been exploited in DPM and DVFS for multicore chips. Compared to our proposed RL-based model, to the best of our knowledge, no one has developed a runtime global solution that is also linearly scalable with the increase of the number of cores and the variability of workloads.

Khan and Rinner [6] developed an RL-based approach for optimal selection of time-out values in the different operational states of a computing system. Das et al. [4] proposed an RL-based, run-time approach for multicore system to adapt to both inter- and intra-application thermal variations. The number of different affinity masks grows exponentially with the number of threads and cores. Ye and Xu [45] developed a Q-learning for multicore processors to learn the system behavior and determine proper processor power state transitions. The solution space increases exponentially with the increase of processor cores; thus, a neural network is used to speed up the training process. Shen et al. [46] present an RL-based approach to minimize the energy dissipation of the peripheral device and the microprocessor under a given performance constraint. Lie et al. [8] improve the convergence speed of the Q-learning for the DPM problem by restricting the search space to the policies that may increase the performance. Juan and Marculescu [47] proposed a semi-supervised RL-based approach for performing DVFS to evaluate and analyze the advantages of managing the processing cores and the on-chip communication fabric in synergy for the purpose of performance increase under power constraints. In our proposed work we dynamically allocate portions of the total power available to the chip, based on the expected performance gain of individual tiles, whereas this proposed technique allocates statically equal portions of the total power budget. Instead of developing new real-time DVFS techniques, Islam and Lin proposed a reinforcement learning approach that learn the best DVFS policy from a set of predefined DVFS policies [48]. Wang et al. [49] proposed an online DVFS based on reinforcement learning at the core level to select the appropriate voltage/frequency values. Biswas et al. [50] proposed a run-time power management approach based on Q-learning to select the appropriate voltage/frequency values for a given workload.

Kumar and Vidyarthi utilize genetic algorithm and DVFS for workload scheduling in multicore systems. The proposed genetic algorithm balanced the power–performance trade-off and the DVFS controls the voltage and frequency [51]. Zhu and Ding proposed a genetic-based scheduling approach for heterogeneous multicore to minimize power consumption [52].

Other works have adopted different learning techniques in power management. However, these techniques suffer from a considerable amount of computation overhead. Jung and Pedram [7] developed a supervised learning-based DVFS for the multicores to predict for each incoming task the system state by using a Bayesian classification technique, provided with collected input features. Dhiman and Rosing [53] apply online learning to select among a set of experts a possible DPM policy or a DVFS setting based on workload characterization that has the best chance to perform well. Kolpe et al. [54] grouped cores in a multicore processor into clusters at design time such that the performance and power dissipation of the clusters is optimized at runtime.

There is also plenary amount of work on maximizing performance under power and thermal constraints using the OS or task schedulers [55]. Such algorithms are centralized and cannot scale well with the increase in the number of cores and variable demands.

2.4 Summary

This section surveyed tens of research contributions in the area of saving energy of multicore chips. These contributions target energy savings using power or performance optimization techniques—as power and performance are the two main components of energy. Power optimization techniques are classified into developing software solutions such as schedulers or hardware solutions such as designing power-aware memories, approximate hardware, GPUs. On the other hand, performance optimization techniques focus on algorithms that intelligently optimize performance for specific power

budget. The section also focused on the machine learning algorithms such as Q-learning and genetic algorithms used to support techniques for both power and performance optimizations. However, none has studied the effects of both optimizations for the same system.

Based on that, our work proposes to study and analyze power and performance optimizations for the system shown in Fig. 1, with a hope to come up with some insights. Moreover, all surveyed work considered power or performance optimization using static power distribution at tile and/or chip levels, whereas this paper proposes an intelligent power management technique implemented at the tile (core/memory) level as well as the chip level that dynamically distributes power to tiles, and pays attention to its scalability when number of tiles and types of workloads increases.

3 Methodology

3.1 Overview

This work addresses power management on multicore chips. We first deploy a reinforcement learning-based agent in each tile to learn the best power needed to achieve the maximum *reward* (i.e., performance) of the corresponding tile. This knowledge is then communicated to a chip-level controller for allocating power budgets to tiles with the objective to optimize either performance or power consumption of the overall system. The chip-level controller utilizes proposed simple heuristic-based algorithms to allocate power. Experiments are then conducted by considering the PARSEC benchmark in a setup with 16 core processor model, shown in Fig. 1, to analyze both optimization cases.

3.2 Reinforcement Learning-Based Agent Model

RL techniques such as Q-learning are used to find an optimal *policy* for a set of states in a given Markov decision process [56]. Q-learning model consists of: an agent, a finite set of states, S, a finite set of actions, A and a reward function, $R_t = R(S_t, A_t)$, where t is a time step. A policy is a set of rules that constructs a mapping between the states and the actions that the agent follows in selecting action(s) in any given state(s). The agent interacts with the environment to learn the optimal policy that maximizes a defined reward function by simply selecting the action that enhances the system performance in any given state at any point in time. Note that the convergence speed to the best power policy can be improved more than conventional Q-learning using other techniques that determine the transition probability [47]. But, since the conventional Q-learning algorithm has experimentally shown at the tile level an acceptable convergence speed, we opt to not proceed with any improvement.

Here, the agent is implemented in each tile, which consists of a core, L1 private cache, L2 shared cache and a router:

- The states are defined using three representative metrics for cores, caches and NoCs, respectively: Instructions Per Cycles (IPC), Misses Per Kilo Instructions (MPKI) and Buffer Utilization (BU),
- The actions are power actions that are defined using two power-dependent parameters: V and F and
- The reward function is defined differently for each of the two investigated cases, specifically:

Case A *the reward function measures the number of instructions per second (IPS) and is defined as:*

$$R = \text{IPS} \tag{1}$$

IPS is a representative metric for multicores because it indicates not just the performance of processing cores but also is a reflect on the performance of memories and NoCs.

Case B *the reward function measures the inverse of the power consumption cost, P:*

$$R = \begin{cases} \frac{1}{P} & \text{if } P \text{ is sufficient for the current workload} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

power is the cost metric of interest in this case. Since RL is designed to maximize the reward function and our target in this case is to minimize the power, we used the inverse power.

The complexity of the algorithm at the tile level is $O(|S|)$, where $|S| = |\text{IPC}| \times |\text{MPKI}| \times |\text{BU}|$. The complexity of the algorithm, if applied at the chip level, is $O(|S|^N)$, where N is the number of tiles. Thus, with the increase in the number of tiles, the Q-learning algorithm complexity becomes exponential. In the next section, we define an alternative, scalable solution, with approximately same performance, as a major contribution of this work.

3.3 Chip-Level Controller Model

For the multicore chips that process parallel, multithreaded applications, efficient power management approaches need to be applied at the chip level. The chip-level view allows for power to be exchanged among cores (e.g., cores that have lower demands can give power to cores with higher

demands), and also it is a better fit to the execution of multi-threaded applications, where different threads are related to each other through application dependencies.

3.3.1 Case A: Performance Optimization

In the first case, we assume in our model that the power budget available to the chip is fixed; thus, the design goal of our global power controller is to maximize the system performance under this power constraint. We find it rational to do so because the typical design goal of multicore chips that run multithreaded programs in parallel is to integrate as much functionality as possible while maintaining the thermal design power (TDP) limit.

Because of the exponential complexity of modeling an Q-learning algorithm at the chip level, we propose a complexity relaxation approach in which we distribute the power model into two hierarchical levels. At the tile level, we deploy a Q-learning agent that learns the best power policy (V/F) for that tile that maximizes the tile performance. At the chip level, we design a power controller that collects from each agent the power-level request needed to apply the best power policy and intelligently allocates to each tile a portion of the power available to the chip that maximizes the overall chip performance. Here, we define a set of notions:

1. N: is the no. of tiles,
2. M: is the no. of clusters,
3. P_{base}: is the baseline power level applied to a tile,
4. $P_{\text{best}}(i)$: is the best power policy learned by the proposed RL model that achieves a high tile performance, when applied to tile i,
5. $G_{\text{base}}(i)$: is the gain (Instructions per Joule in a Second) achieved by allocating P_{base} Watts to tile i,
6. $G_{\text{best}}(i)$: is the gain (Instructions per Joule in a Second) achieved by allocating P_{best} Watts to tile i, and
7. P_{total}: is the total power budget available to the chip.

The gain, G, is defined in Equation 3 as the number of instructions executed per energy unit in a second,

$$G = \frac{\text{IPS}}{P} \tag{3}$$

where P is power. G is representative because it puts all processing cores on the same power scale, regardless of the type of the core or the executed workload.

Algorithm 1 defines our problem. Our proposed global controller should be able to maximize the total gain, TG, while maintaining the power within the total fixed chip power, P_{total}. The problem of defining the best power portion allocations among multiple tiles that maximizes overall performance is an optimization problem, which needs to be solved using an integer linear programming (ILP) optimizer; as such its complexity is exponential, as well.

Algorithm 1: Problem Definition of Case A

Data: Power Level Request j
Result: Total Gain TG

$$X_j = \begin{cases} 0 & \text{Base Power} \\ 1 & \text{Best Power} \end{cases}$$

Maximize:

$$TG = \sum_j G_{best}(j).X_j + \sum_j G_{\text{base}}(j).(1 - X_j)$$

Subject to:

$$\sum_j P_{best}.X_j + \sum_j P_{\text{base}}.(1 - X_j) \leq P_{total}$$

To reduce the complexity of the ILP model, we develop a *heuristic* model that has approximately the same performance as the ILP, but with polynomial complexity. Algorithm 2 describes the heuristic model implemented in our controller, in which at the start of each control period the model receives from each agent the best power action learned by its Q-learning algorithm, P_{best}. The controller estimates the required power using the power model described later in the experiments section. The model also receives the expected reward and calculates the associated gain, G_{best}, that can be achieved by the best power policy, and the gain that can be received when the baseline power is applied, G_{base}, using prior execution knowledge. The model then sorts the power-level requests in descending order according to the gain, both G_{best} and G_{base}—because it may happen that the G_{base} is greater than the G_{best} depending on the type of the workload processed on the cores. The model after that starts with the request with the highest gain. If the difference between the available power budget, P_{total}, and the requested power, P_{best}, is enough to provide the baseline power to each of the other

Algorithm 2: Heuristic Model for Case A

Data: Power Level Request j, Number of Agents N
Result: Total Gain TG
Sort j based on (G_{best}, G_{base}) in descending order
for *each j in the sorted list* **do**
  if $(P_{total} - P_{best}) > \sum_j^N P_{base}$ **then**
    $P_{total} = P_{total} - P_{best}$
    $TG += G_{best}$
  else
    $P_{total} = P_{total} - P_{base}$
    $TG += G_{base}$
  end
end

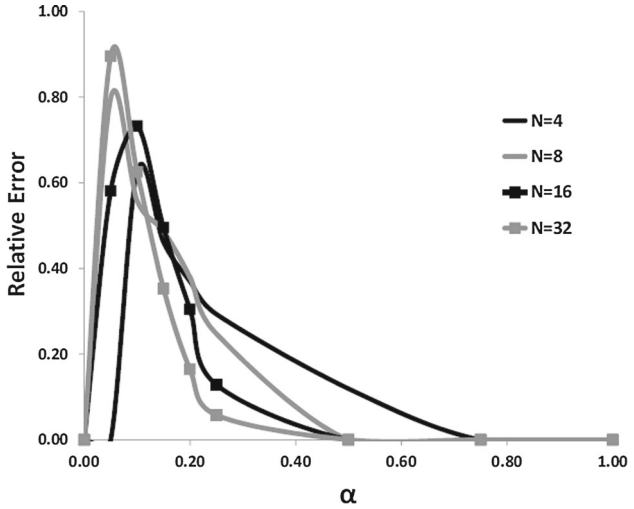


Fig. 2 The relative error, defined as the ratio of the difference between our heuristics and the optimal to the optimal, for different number of tiles N

remaining requests, that best power-level request is granted. Otherwise, the baseline power-level request is granted. Note here that we assume that the chip power budget, P_{total}, is at least enough to provide each tile with the baseline power, P_{base}.

Figure 2 shows the performance of our heuristics, compared to the ILP optimal solution computed using the IBM CPLEX Optimizer [57], for different number of tiles, N. Note that the relative error, defined as the ratio of the difference to the optimal, is less than 1%, at worst. Also, note that there is no difference between our heuristic and the ILP model as the total power budget, P_{total}, is close to the summation of the baseline or the maximum power levels because the optimal solution would be to provide each tile with the baseline or the best power, respectively, where α defines how much the power budget is above the baseline power of the N cores. Note that even when N increases from four to 32 cores the relative error stays within an acceptable margin of maximum 0.25%. The data used to validate our heuristic, 700 in total where each include P_{best}, P_{base}, G_{best}, G_{base} and P_{total}, were generated using random functions fed with measured data from our experiments. We have chosen five power levels for each tile that range from 16 to 32 W, where the TDP of the Intel Xeon processor used in our experiments is 32.5 W.

The control period chosen is 1 ms, while the controller timing overhead found experimentally is 29 µs, which is included in the execution time. The complexity of our heuristics model is the complexity of the sort procedure. We used the *simple* bubble sort, which has a quadratic complexity, at worst. This is due to the fact that the complexity of the Q-learning algorithm is at least quadratic as well. So even if $O(n \log n)$ sorting algorithm is used, the overall complexity will remain quadratic.

3.3.2 Case B: Power Optimization

In the second case, we aim at minimizing the total chip power consumption, while maintaining constant performance, which is similar to the objective of conventional DVFS techniques. Algorithm 3 formally defines the design objective of Case B as to minimize the consumed power, P_{consumed}, while maintaining a constant chip total gain, G_{const}. Similar to the problem definition in Case A, the complexity of this algorithm has been relaxed using a heuristic model, described in Algorithm 4. Note that in this case we employ the same system-level model, described in the previous section of Case A. However, the heuristic model that controls the function of the global controller has been changed.

The model shown in Algorithm 4 receives at each control epoch best power actions from each tile-based agent, learned using the Q-learning algorithm, P_{best}, as well as the

Algorithm 3: Problem Definition of Case B

Data: Power Level Request j, Total Constant Gain G_{const}
Result: Total Consumed Power $P_{consumed}$

$$X_j = \begin{cases} 0 & \text{Constant Power} \\ 1 & \text{Best Power} \end{cases}$$

Minimize:

$$P_{consumed} = \sum_j P_{best}(j).X_j + \sum_j P_{const}(j).(1 - X_j)$$

Subject to:

$$\sum_j G_{best}.X_j + \sum_j G_{const}.(1 - X_j) \geq G_{const}$$

Algorithm 4: Heuristic Model for Case B

Data: Power Level Request j, Number of Agents N, Constant Gain G_{const}
Result: Total Consumed Power $P_{consumed}$
if $(\sum_1^N G_{best}) \geq G_{const}$ **then**
  Grant requested power levels $P_{best\,i}$ $\forall\, i$ from 1 to N
else
  $G_{remaining} = G_{const}$
  Sort j based on $diff(G_{best}, G_{const})$ in descending order
  for *each j in the sorted list* **do**
    if $(\sum_j^N G_{best}) \geq G_{remaining}$ **then**
      Grant requested power levels $P_{best\,i}$ $\forall\, i$ from j to N
      $P_{consumed} = P_{consumed} + \sum_j^N P_{best}$
    else
      Grant request j its $P_{const\,j}$
      $G_{remaining} = G_{remaining} - G_{const\,j}$
      $P_{consumed} = P_{consumed} + P_{const\,j}$
    end
  end
end

associated IPS result from applying this power policy. The model uses this information to calculate the expected gain, G_{best}, according to Eq. 3. Note that the model also defines a constant power level, P_{const}, and accordingly calculates the associated gain that may be achieved from applying this power level, G_{const}. Using this information, the model first checks whether the total gain, G_{best}, expected from applying the requested power levels, P_{best}, would reach the total gain the chip needs to maintain, G_{const}. If so, the controller would grant all agents the requested power levels; otherwise, the controller would sort the requests in descending order according to the difference between the gain expected from applying the requested power, G_{best}, and that of the constant power level, G_{const}. Then, the controller starts granting the constant gain, G_{const}, so long as the remaining gain expected from applying the P_{best} is not sufficient to reach the total constant gain, G_{const}.

Note that our definition of the gain works for both of Case A and Case B as our goal is to maximize the gain either through maximizing the performance, IPS, (Case A) or minimizing the consumed power, P_{consumed}, (Case B). For the purpose of this paper, G_{const}, used in Case B, is assigned to different values generated from Case A, as discussed in the next section, as a common ground to compare the two optimization techniques.

Remember that the RL agent developed at the tile level has also been changed in this case (Case B), as described in Sect. 3, as its reward function is to minimize power consumption, in contrast to that of the first case (Case A) where the goal is to maximize the performance.

3.4 Experimental Setup

We evaluate the power–performance efficiency of the two proposed techniques, and thus the associated proposed algorithms, for a 16-core processor as shown in Fig. 1. This chip consists of 16 tiles connected with a 4 × 4 MESH NoC architecture. Each tile includes a processing core, a L1 cache, a portion of the shared L2 cache, and a router.

The applications used in the evaluation belong to the PARSEC 2.1 benchmark suite [58]. PARSEC targets multicore chips with shared memory and includes programs from emerging workloads and diverse domains [59]. For this work we used the medium input dataset for the execution of the applications. We run the workloads on the gem5 full-system mode. gem5 is widely used in the design of multicore systems because of its flexible simulation framework [60]. The applications were simulated until completion. For real implementation, there are many APIs/tools that can be used to monitor hardware performance counters including time, cache hits and misses and power. Since our proposed algorithms use Intel-based implementation, Intel offers many APIs/tools to monitor hardware performance counters such as Intel PCM (performance counter monitor) [61].

Table 1 Architectural parameters

Parameter	Value
Number of tiles	16
Core type	Intel Xeon
L1-I/D caches	Private 32KB, 8-way SA
L2 caches	Shared 4MB, 32-way SA
DRAM	2 GB
NoC	GARNET model
Technology	45 nm
Power L1 (V, F)	0.6, 2.0 GHz
Power L2 (V, F)	0.8, 2.5 GHz
Power L3 (V, F)	1.0, 3.0 GHz
Power L4 (V, F)	1.2, 3.5 GHz
Power L5 (V, F)	1.4, 4.0 GHz

Table 1 summarizes the architectural parameters for the target multicore chip. For the NoC, we used GARNET's fixed pipeline model [62]. GARNET is a detailed cycle-accurate interconnection network model that can be integrated with gem5 simulator. It supports flit size of 16 bytes, buffer size of four bytes, five pipeline stages, five virtual networks and four virtual channels per virtual network. For the power model of NoC, we use Orion [63] incorporated into GARNET. Orion is a power area simulator to obtain the power consumption of NoCs including routers, FIFO buffers, arbiters and physical links.

In order to control the power levels of the processing cores we used five V/F pairs (L1 to L5) as shown in Table 1. To estimate the power consumption of the cores, we use the power model of Bartolini et al. [64]. This model was verified to be 90% accurate, compared to the actual power consumption.

RUBY [60] is a built-in component in gem5. It is used to model memories. To calculate the power dissipated by the L1 and L2 caches, we integrate CACTI [65] with RUBY.

The Q-learning agent was implemented on each processing core as a kernel thread, whereas the global-level power controller was implemented at the OS level in gem5's FS mode. We used linux 2.6.28.4 for x86-64 bits multicore chip. The same implementation of the Q-learning and the global controller can be used on a real machine.

We design three sets of experiments for each optimization case to demonstrate our contribution, described in the following subsections. As in other similar studies, the baseline to compare the performance of our proposed algorithm is the non-DVFS static power assignment.

3.4.1 Setup of Case A

In this case we configure different power allocations as follows:

1. *Tile Level* Each tile is assigned statically a specific portion of the total power budget. We developed a recursive model, Algorithm 5, to generate all possible combinations of three fractions of the total power (M): $\frac{1}{2M}$, $\frac{1}{M}$, and $\frac{3}{2M}$. This results into nine configurations ($T1$ to $T9$) for $N = 16$, where $M = N$, as shown in Table 2.
2. *Cluster Level* If tiles are grouped into clusters, then we can statically assign a specific portion of the total budget to the cluster. We choose three cluster sizes: *2-way*: eight clusters, *4-way*: four clusters and *8-way*: two clusters. Again, we use the recursive algorithm to generate all possible allocation combinations ($C1$ to $C10$ in Table 2). For instance, C6 is similar to the model proposed by [47]. Inside each cluster, we dynamically allocate power among tiles using Algorithm 2.
3. *Global Level* The total power budget is assigned to the whole chip. The core power levels are assigned dynamically using our Algorithm 2.

Algorithm 5: Recursive Model

Let $W = \{\beta_1 P_{total}, \beta_2 P_{total}, \cdots, \beta_M P_{total}\}$ be a power portion allocation for the M clusters such that:

1. $\beta_j \in \{\frac{1}{2M}, \frac{1}{M}, \frac{3}{2M}\}$
2. $\sum_{j=1}^{M} \beta_j = 1$

Then, $\mathcal{W}$ is defined as the set of all possible power portion allocations that satisfy the two conditions mentioned above.

3.4.2 Setup of Case B

In this case we configure different gain assignments that need to be maintained at the tile/chip level as follows:

1. *Tile Level* Each tile is assigned statically a specific portion of the total chip constant gain, G_{const}. We use the same recursive model, Algorithm 5, to generate all possible combinations of three fractions of the total constant gain (M): $\frac{1}{2M}$, $\frac{1}{M}$ and $\frac{3}{2M}$. This results into nine configurations ($T1$ to $T9$) for $N = 16$, where $M = N$, similar to the power portion distribution shown in Table 2.
2. *Cluster Level* We statically assign a specific portion of the chip constant gain, G_{const}, to each cluster. We choose three cluster sizes: *2-way*: eight clusters, *4-way*: four clusters and *8-way*: two clusters. Again, we use the recursive algorithm to generate all possible allocation combinations ($C1$–$C10$ in Table 2). Inside each cluster, we dynamically allocate power among tiles using Algorithm 4, in a way to maintain the assigned gain for that cluster.
3. *Global Level* The total chip constant gain, G_{const}, is assigned to the whole chip. The core power levels that achieve this gain are assigned dynamically using our Algorithm 4.

Table 2 Power portion distribution

Tile based	Cluster based		
	2-Way	4-Way	8-Way
$\{\frac{1}{32}, \frac{1}{16}, \frac{3}{32}\}$	$\{\frac{1}{16}, \frac{1}{8}, \frac{3}{16}\}$	$\{\frac{1}{8}, \frac{1}{4}, \frac{3}{8}\}$	$\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$
T1. {0, 16, 0}	C1. {0, 8, 0}	C6. {0, 4, 0)	C9. {0, 2, 0}
T2. {1, 14, 1}	C2. {1, 6, 1}	C7. {1, 2, 1}	C10. {1, 0, 1}
T3. {2, 12, 2}	C3. {2, 4, 2}	C8. {2, 0, 2}	
T4. {3, 10, 3}	C4. {3, 2, 3}		
T5. {4, 8, 4}	C5. {4, 0, 4}		
T6. {5, 6, 5}			
T7. {6, 4, 6}			
T8. {7, 2, 7}			
T9. {8, 0, 8}			

In order to compare the two proposed optimization methodologies, we set the values of chip constant gain, G_{const}, of the tile-level experiment to the best gain resulted from the nine tile-level power configurations of Case A. Similarly, G_{const} of each cluster-level experiment is set to the best gain resulted from the corresponding cluster-level power configurations of Case A. Finally, in the case of the global-level experiment, we set G_{const} to the global-level gain of Case A. The gain is obtained from the execution time using Equation 4.

$$G = \frac{I_{\text{C}}}{E_{\text{T}} P} \tag{4}$$

where I_{C}, E_{T} and P are the instruction count, the execution time and the consumed power, respectively, of specific benchmark application.

4 Results and Discussion

4.1 Results

4.1.1 Results of Case A

We present the experimental results where we compare the performance of the tile- and cluster-level power controllers to our proposed dynamic global power controller.

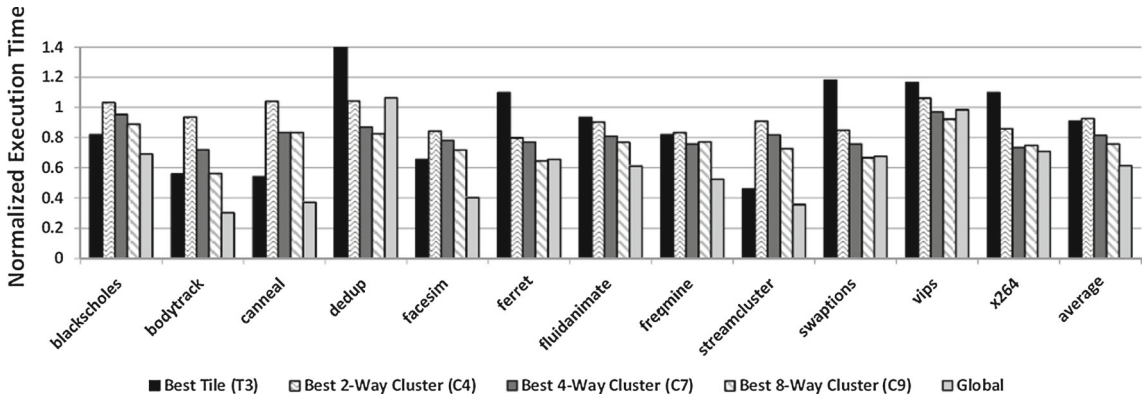


Fig. 3 Case A: the execution time of the best performing tile and cluster, compared to our global controller, normalized to the baseline

Figure 3 depicts the execution time of each PARSEC benchmark, normalized to the execution time of the baseline. To focus the discussion onto the main findings, the figure shows only the results of the best tile-, cluster- and global-level configurations.

The results in Fig. 3 show that the best tile configuration ($T3$) outperforms the baseline by 9%, on average. Configuration $T3$ statically allocates the chip power budget unevenly ($2 \times \frac{1}{32}$, $12 \times \frac{1}{16}$, and $2 \times \frac{3}{32}$)—this allows some benchmarks to take advantage of this diversity to match their different runtime requirements. For instance, the pipeline-parallel benchmarks, *dedup* and *ferret*, behave worse than the baseline because their execution phases are short, so runtime DVFS does not help. Others like *swaptions*, *vips* and *x264* show also worse performance than the baseline, because the fixed tile budget cannot grant the best power policy, especially for the phases that include high data transfer. Benchmarks with high Clock per Instruction (CPI) variations such as *bodytrack*, *canneal*, *facesim* and *streamcluster* perform better because they exhibit frequent, long phases that can take advantage of DVFS.

Figure 3 also shows the best performing configurations for each cluster size: configuration $C4$ in the 2-way cluster, configuration $C7$ in the 4-way cluster and configuration $C9$ in the 8-way cluster. The figure shows that configuration $C9$ is the best configuration with 25% performance improvement on average, relative to the baseline. It is again observed from the results that the more flexibility of allocating power among tiles (i.e., larger cluster sizes) is better because it provides more flexibility and thus is able to match better the different program needs at runtime. Interestingly, benchmarks such as *canneal* and *streamcluster* become worse, in contrast to tile level, especially in configurations that have smaller cluster sizes. This is because the implementation of the DVFS struggles with the competition of the different tiles within each cluster, for the fixed cluster power budget.

Notice the results show that it is relevant to study a large set of configurations since in most cases the best configuration is one with uneven power distribution among the tiles or clusters. For example, for the tile based, the applications achieve better performance with $T3$ as compared to $T1$.

Figure 3 shows that our dynamic global power controller outperforms the baseline on average by 39%. Compared to [47], *C6*, our system outperforms by 30%. This demonstrates that global-level power adaptation responds more properly to the changes in the workloads. Most of the benchmarks took advantage of the adaptability provided by our proposed controller except *vips* because of its high network contention that result from frequent access to the memory and *dedup* because of the unpredictability due to the frequent phase changes with short periods.

In addition to improving the performance by reducing the execution time of the applications, the proposed algorithm can also achieve benefits in terms of power consumption since it adapts better to the specific needs of the application. In order to show this we depict in Fig. 4 the Energy Delay Product (EDP), which is a metric that is used to evaluate the power–performance of the execution. The results in Fig. 4 show that our dynamic global controller outperforms the baseline by 28%, on average. This is the best performance for all three techniques studied since the best cluster-level controller achieves only 21% and the best tile-level controller only 6%, compared with the baseline. Compared to [47], *C6*, our system outperforms by 16%.

Overall, the results show that the more flexibility there is for allocating the power budget at runtime, the more performance improvement is achieved. This is justified by the fact that there are conditions for the better utilization of the power budget in order to maximize the performance. The gains observed for the dynamic global algorithm are in terms of both performance and energy.

4.1.2 Results of Case B

Here, we show the experimental results from which we compare the power consumption of the tile- and cluster-level

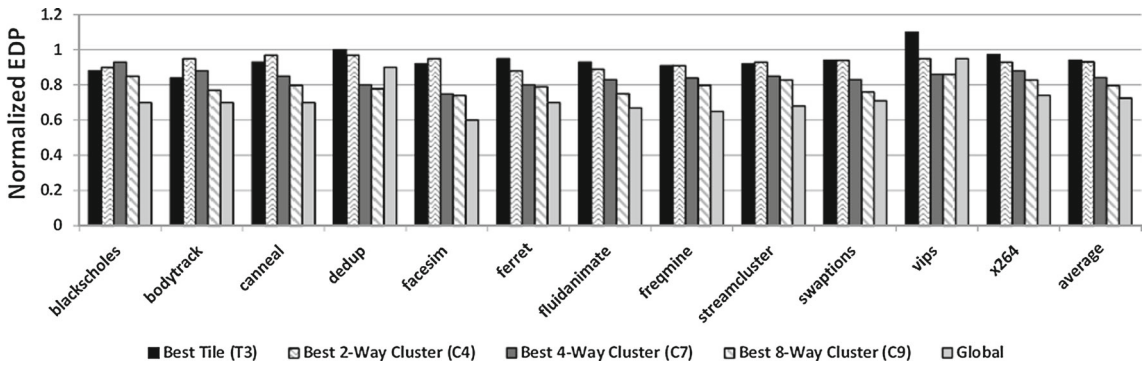


Fig. 4 Case A: the EDP of the best performing tile and cluster, compared to our controller, normalized to the baseline

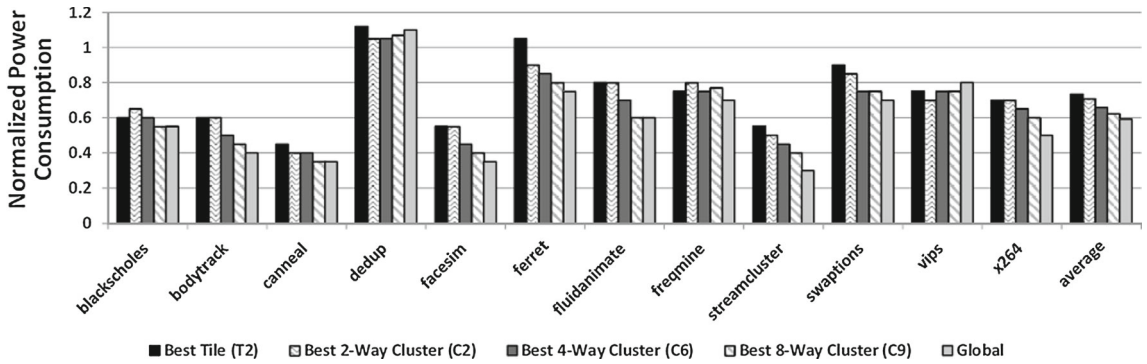


Fig. 5 Case B: the power consumption of the best performing tile and cluster, compared to our global controller, normalized to the baseline

power controllers to our proposed dynamic global power controller, normalized to the non-DVFS baseline.

Figure 5 depicts the power consumption of each PARSEC benchmark, normalized to the power consumption of the baseline. Again, we show only the results of the best tile-, best cluster- and global-level configurations.

The results in Fig. 5 show that the best tile configuration ($T2$) outperforms the baseline by saving 26% of the power, on average. Configuration $T2$ statically distributes the chip constant gain that needs to be maintained. Remember that, in this case, the chip constant gain is the one obtained from the best tile-level configuration of Case A ($T3$). For instance, *dedup* and *ferret* benchmarks consume power a little more than the baseline because their execution phases are short—so runtime dynamic power actions do not help, but incur more power overhead. Similarly, other benchmarks like *swaptions*, *vips* and *x264* show a little more power saving than the baseline because these benchmarks by default need power especially for the phases that include high data transfer, so dynamic power actions do not work very well. Benchmarks such as *bodytrack*, *canneal*, *facesim* and *streamcluster*, characterized with high CPI variations, saved more power because they exhibit frequent, long phases that can take advantage of dynamic power actions. *blackscholes* already do not require too much power; thus, dynamic power actions do not help that much. Benchmarks like *fluidanimate* and *freqmine* showed a tangible amount of power saving as they by default consume considerable amount of power.

Figure 5 also shows the best performing configurations for each cluster size: configuration $C2$ in the 2-way cluster, configuration $C6$ in the 4-way cluster and configuration $C9$ in the 8-way cluster. The figure shows that configuration $C9$ is the best configuration with 37.5% power saving on average, relative to the baseline. Interestingly, the results trend of intra-cluster configurations (not shown) is that the more the gain distribution is even, the more power is saved. This is due to the fact that cores used in this experiment are homogeneous; thus, balanced distribution of the gain (which is interpreted as load) would result in better performance. Further, some benchmarks have slightly affected negatively in cluster configurations relative to the baseline, such as *blackscholes* and *freqmine*.

Figure 5 shows that our dynamic global power controller outperforms the baseline in terms of power saving by 41%, on average. This demonstrates that the global controller was able to monitor the gain received from tiles and *intelligently* distribute the power, accordingly.

Similar to the experiment of case A, Fig. 6 depicts the Energy Delay Product (EDP) of Case B to evaluate the power–performance trade-off of the execution. The results in Fig. 6 show that our dynamic global controller outperforms the baseline by 35.5%, on average. This is the best

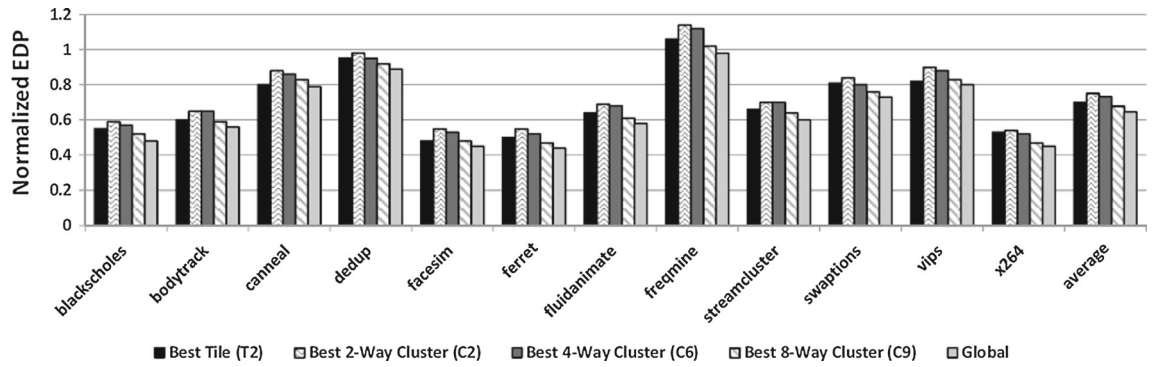


Fig. 6 Case B: the EDP of the best performing tile and cluster, compared to our controller, normalized to the baseline

performance for all three techniques studied since the best cluster-level controller achieves 32% and the best tile-level controller achieves 30%, compared with the baseline.

Overall, the results suggest that when using the global controller to balance load distribution, better performance improvement was achieved.

4.2 Discussion and Analysis

In the previous section we presented the results for our dynamic global algorithm for both optimization cases. The goal of the algorithm in the first case (Case A) is to achieve the best performance within a fixed power budget. Some energy savings in the execution (as seen in the benefits in terms of EDP that are shown in Fig. 4) were achieved as a side effect from the algorithm trying to optimize performance and improve the power utilization (as depicted in Equation 5, where P and $T(t)$ are the constant power and the time-varying execution time, respectively, for each benchmark). Similarly, the algorithm of the second case (Case B) achieves the power–performance trade-off through optimizing power consumption while maintaining a constant performance of the chip, as described in Eq. 6, where $P(t)$ and T are the time-varying power and the constant reciprocal performance or execution time, respectively, for each benchmark. This raises the question of which optimization technique provides better power–performance benefits (i.e., lower EDP metric).

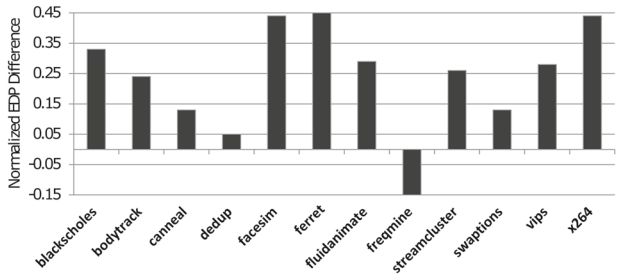
$$\mathrm{EDP}_A = P * T(t)^2 \tag{5}$$

$$\mathrm{EDP}_B = P(t) * T^2 \tag{6}$$

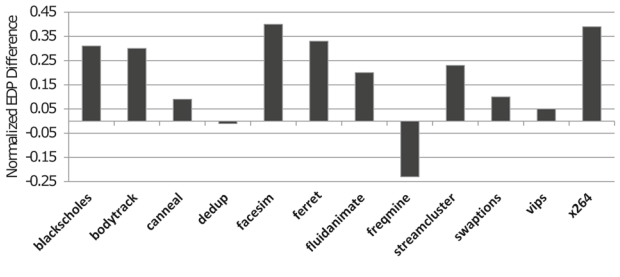
Figure 7 depicts the differences between the two different energy optimizations. It shows for each benchmark the difference between Case A and Case B in terms of the EDP, normalized to the non-DVFS baseline. The positive difference is in favor of Case B, whereas the negative difference is in favor of Case A. The results show the following:

- Energy is reduced more effectively in most benchmarks when Case B is used. For instance, Fig. 7a shows that 11 out of the 12 benchmarks have positive difference, i.e., the normalized EDP of Case B prevails. This ratio changes over different experiments, but preserved same trend.
- The previous point is demonstrated in Fig. 7f which clearly shows, overall, optimizing energy through power was more effective. While this contradicts the math in Eqs. 5 and 6 as execution time is squared and thus should have more impact, power is more direct to saving energy. Also, the figure shows that optimizing the power distribution favors the more flexibility, whereas optimizing the load distribution favors the more balancing, especially in homogeneous chip multiprocessors.
- Figure 7a–e shows that different benchmarks behave differently from one configuration to another and relative to each other in the same configuration. Specifically, the figures show that the only benchmark that favored Case A in all configurations is the *freqmine*. This benchmark does not consume too much power; thus, performance optimization is more effective. The *Canneal* benchmark also favored Case A in the more flexible configurations only. The high CPI variations and long, frequent execution phases in the *Canneal* benchmark took more advantage of performance optimization, especially when the portion distribution of the total power budget was dynamic. Another exception is the *dedup* benchmark which showed energy saving slightly below and above the bar, where both optimizations show the same performance.

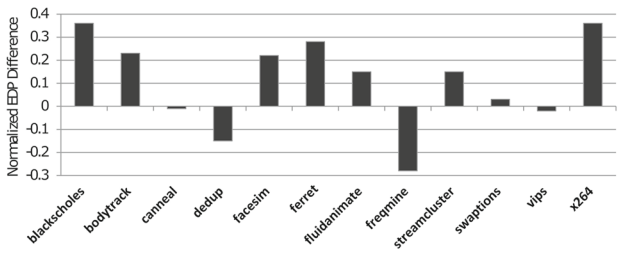
Since the global controller in Case A showed best performance than any other configuration, this means that the global controller can satisfy the performance of the other configurations at a smaller power budget. Figure 8 depicts these savings relative to the baseline, tile and cluster configurations. The curves of the tile- and cluster-based techniques, as well as the baseline, shown in the figure, demonstrate the percentage increase in the total power budget and the speedup achieved,



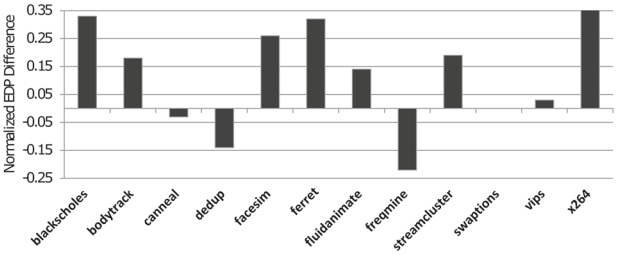
(a) Best Tile Config., per benchmark



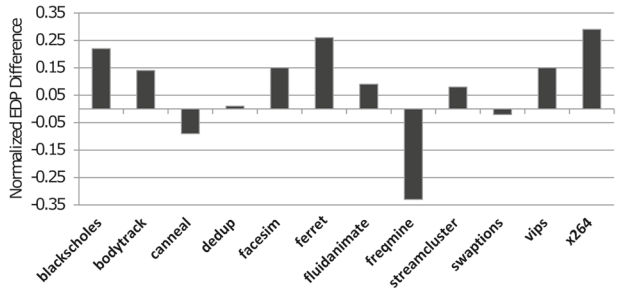
(b) Best 2-Way Cluster Config., per benchmark



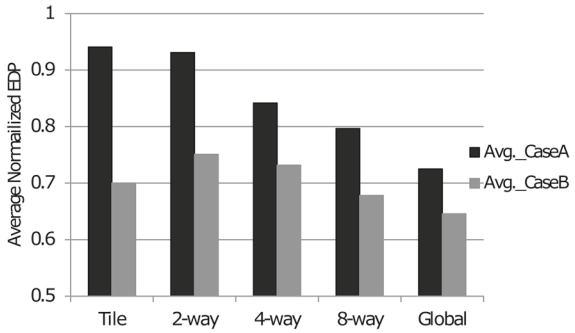
(c) Best 4-Way Cluster Config., per benchmark



(d) Best 8-Way Cluster Config., per benchmark



(e) Global Config., per benchmark



(f) Average EDP of all benchmarks for each Config.

Fig. 7 Comparison between the performance optimization (Case A) and power optimization (Case B) in trading off the power–performance metrics for chip multiprocessors executing parallel workloads. **a**–**e** The difference between the EDP metric, normalized to the baseline, of Case B and that of Case A for best tile-, 2-way, 4-way and 8-way cluster and global configurations, respectively. **f** The average EDP metric, normalized to the baseline, for all benchmarks in the best configuration of each type

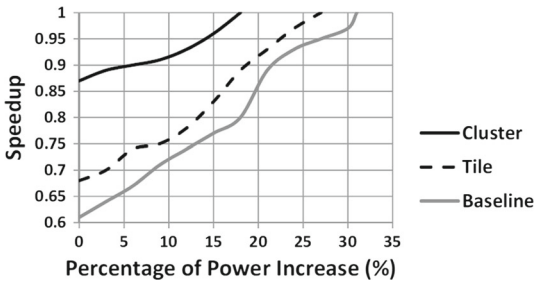


Fig. 8 The percentage increase in the total power budget (for both tile- and cluster-level controllers) to achieve same performance as of our global controller

relative to the global controller. The points where the curves cross the vertical axis indicate the speedup achieved by the other techniques, compared to global, when the power budget is the same as the one for global. On the other hand, the points where the curves cross the horizontal axis indicate the extra power budget that is necessary for the different techniques to achieve the same performance as global.

Consequently, from Fig. 8 it is possible to observe that the best cluster, best tile and non-DVFS techniques required 18, 27 and 31% more power than the global controller, respectively, to achieve the same performance of the global. Therefore, the global controller can also be used as a way to satisfy the performance of the other techniques but at a lower power budget.

Table 3 Power consumption, normalized to the baseline

Chip configuration	Optimization case	
	Case A	Case B
Tile	0.95	0.74
2-Way cluster	0.91	0.71
4-Way cluster	0.87	0.66
8-Way cluster	0.84	0.63
Global	0.69	0.59

Table 3 compares these power savings to the power consumed by the different techniques in Case B. Note that comparison between the two optimization cases is valid as we use the same generated gain from Case A, having fixed power budget, to be achieved in Case B, where no power limitation exists. The results demonstrate that targeting power optimization leads to better power–performance trade-off than optimizing performance. This confirms the findings of the previous analysis.

5 Conclusions and Future Work

In this paper, we proposed a novel machine learning-based energy optimization technique that we used to either maximize the performance within a fixed power budget or minimize the consumed power to achieve the same baseline performance. We showed experimentally that using our proposed technique outperformed, in terms of the EDP metric, the non-DVFS baseline by 28 and 35.5%, when optimized for performance and power, respectively. This suggests that optimizing power is more direct to energy efficiency than optimizing performance.

For future work, we plan to test our proposed algorithm along with other state-of-the-art algorithms using real workloads on different real platforms to provide a more comprehensive cause and effect analysis.

References

1. Huang, K.; Santinelli, L.; Chen, J.-J.; Thiele, L.; Buttazzo, G.C.: Adaptive power management for real-time event streams. In: Proceedings of the 2010 Asia and South Pacific Design Automation Conference, pp. 7–12. IEEE Press (2010)
2. Jha, S.S.; Heirman, W.; Falcón, A.; Tubella, J.; González, A.; Eeckhout, L.: Shared resource aware scheduling on power-constrained tiled many-core processors. J. Parallel Distrib. Comput. **100**, 30–41 (2017)
3. Sharifi, A.; Mishra, A.K.; Srikantaiah, S.; Kandemir, M.; Das, C.R.: Pepon: performance-aware hierarchical power budgeting for noc based multicores. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, pp. 65–74. ACM (2012)
4. Das, A.; Shafik, R.A.; Merrett, G.V.; Al-Hashimi, B.M.; Kumar, A.; Veeravalli, B.: Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM (2014)
5. Otoom, M.; Trancoso, P.; Almasaeid, H.; Alzubaidi, M.: Scalable and dynamic global power management for multicore chips. In: Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures, pp. 25–30. ACM (2015)
6. Khan, U.A.; Rinner, B.: Online learning of timeout policies for dynamic power management. ACM Trans. Embed. Comput. Syst. (TECS) **13**(4), 96 (2014)
7. Jung, H.; Pedram, M.: Supervised learning based power management for multicore processors. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(9), 1395–1408 (2010)
8. Liu, W.; Tan, Y.; Qiu, Q.: Enhanced q-learning algorithm for dynamic power management with performance constraint. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 602–605. European Design and Automation Association (2010)
9. Chen, Z.; Marculescu, D.: Distributed reinforcement learning for power limited many-core system performance optimization. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 1521–1526. EDA Consortium (2015)
10. Kianzad, V.; Bhattacharyya, S.S.; Qu, G.: Casper: an integrated energy-driven approach for task graph scheduling on distributed embedded systems. In: 16th IEEE International Conference on Application-Specific Systems, Architecture Processors, 2005. ASAP 2005, pp. 191–197. IEEE (2005)
11. Hua, S.; Qu, G.; Bhattacharyya, S.S.: Energy reduction techniques for multimedia applications with tolerance to deadline misses. In: Proceedings of the 40th Annual Design Automation Conference, pp. 131–136. ACM (2003)
12. Choi, K.; Soma, R.; Pedram, M.: Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **24**(1), 18–28 (2005)
13. Gheorghita, S.V.; Basten, T.; Corporaal, H.: Application scenarios in streaming-oriented embedded-system design. IEEE Des. Test Comput. **25**(6) (2008)
14. Xian, C.; Lu, Y.-H.; Li, Z.: Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In: Proceedings of the 44th Annual Design Automation Conference, pp. 664–669. ACM (2007)
15. Yang, P.; Marchal, P.; Wong, C.; Himpe, S.; Catthoor, F.; David, P.; Vounckx, J.; Lauwereins, R.: Managing dynamic concurrent tasks in embedded real-time multimedia systems. In: Proceedings of the 15th International Symposium on System Synthesis, pp. 112–119. ACM (2002)
16. Kim, W.; Gupta, M.S.; Wei, G.-Y.; Brooks, D.: System level analysis of fast, per-core dvfs using on-chip switching regulators. In: IEEE 14th International Symposium on High Performance Computer Architecture, 2008. HPCA 2008, pp. 123–134. IEEE (2008)
17. Hua, S.; Qu, G.: Approaching the maximum energy saving on embedded systems with multiple voltages. In: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design, p. 26. IEEE Computer Society (2003)
18. Li, D.; Chou, P.H.; Bagherzadeh, N.: Mode selection and mode-dependency modeling for power-aware embedded systems. In: Proceedings of the 2002 Asia and South Pacific Design Automation Conference, p. 697. IEEE Computer Society (2002)
19. Hoeller Jr., A.S.; Wanner, L.F.; Fröhlich, A.A.: A hierarchical approach for power management on mobile embedded systems. In: Kleinjohann, B., Kleinjohann, L., Machado, R.J., Pereira, C.E.,

Thiagarajan, P.S. (eds.) From Model-Driven Design to Resource Management for Distributed Embedded Systems, pp. 265–274. Springer, Berlin (2006)
20. Bhatti, K.; Belleudy, C.; Auguin, M.: Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In: 2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC), pp. 184–191. IEEE (2010)
21. Niu, L.; Quan, G.: Reducing both dynamic and leakage energy consumption for hard real-time systems. In: Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pp. 140–148. ACM (2004)
22. Kim, M.; Ha, S.: Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. ACM SIGPLAN Not. **36**(8), 11–19 (2001)
23. Shin, Y.; Choi, K.; Sakurai, T.: Power optimization of real-time embedded systems on variable speed processors. In: Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, pp. 365–368. IEEE Press (2000)
24. Trajkovic, J.; Veidenbaum, A.V.; Kejariwal, A.: Improving sdram access energy efficiency for low-power embedded systems. ACM Trans. Embed. Comput. Syst. (TECS) **7**(3), 24 (2008)
25. Reddy, R.; Petrov, P.: Cache partitioning for energy-efficient and interference-free embedded multitasking. ACM Trans. Embed. Comput. Syst. (TECS) **9**(3), 16 (2010)
26. Tsai, Y.-Y.; Chen, C.-H.: Energy-efficient trace reuse cache for embedded processors. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **19**(9), 1681–1694 (2011)
27. Hajimiri, H.; Rahmani, K.; Mishra, P.: Synergistic integration of dynamic cache reconfiguration and code compression in embedded systems. In: Green Computing Conference and Workshops (IGCC), 2011 International, pp. 1–8. IEEE (2011)
28. Albonesi, D.H.: Selective cache ways: on-demand cache resource allocation. In: 32nd Annual International Symposium on Microarchitecture, 1999. MICRO-32. Proceedings, pp. 248–259. IEEE (1999)
29. Zhang, C.; Vahid, F.; Najjar, W.: A highly configurable cache architecture for embedded systems. In: 30th Annual International Symposium on Computer Architecture, 2003. Proceedings, pp. 136–146. IEEE (2003)
30. Kin, J.; Gupta, M.; Mangione-Smith, W.H.: The filter cache: an energy efficient memory structure. In: Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, pp. 184–193. IEEE Computer Society (1997)
31. Steinke, S.; Grunwald, N.; Wehmeyer, L.; Banakar, R.; Balakrishnan, M.; Marwedel, P.: Reducing energy consumption by dynamic copying of instructions onto onchip memory. In: Proceedings of the 15th International Symposium on System Synthesis, pp. 213–218. ACM (2002)
32. Benini, L.; Macii, A.; Macii, E.; Poncino, M.: Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. IEEE Des. Test Comput. **17**(2), 74–85 (2000)
33. Bournoutian, G.; Orailoglu, A.: Miss reduction in embedded processors through dynamic, power-friendly cache design. In: Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, pp. 304–309. IEEE (2008)
34. Han, J.; Orshansky, M.: Approximate computing: an emerging paradigm for energy-efficient design. In: Test Symposium (ETS), 2013 18th IEEE European, pp. 1–6. IEEE (2013)
35. Mittal, S.; Gupta, S.; Dasgupta, S.: Fpga: an efficient and promising platform for real-time image processing applications. In: National Conference on Research and Development in Hardware Systems (CSI-RDHS) (2008)
36. Keckler, S.W.; Dally, W.J.; Khailany, B.; Garland, M.; Glasco, D.: GPUs and the future of parallel computing. IEEE Micro **31**(5), 7–17 (2011)
37. Ma, K.; Li, X.; Chen, M.; Wang, X.: Scalable power control for many-core architectures running multi-threaded applications. In: ACM SIGARCH Computer Architecture News, vol. 39, pp. 449–460. ACM (2011)
38. Wang, Y.; Ma, K.; Wang, X.: Temperature-constrained power control for chip multiprocessors with online model estimation. In: ACM SIGARCH Computer Architecture News, vol. 37, pp. 314–324. ACM (2009)
39. Mishra, A.K.; Srikantaiah, S.; Kandemir, M.; Das,C.R.: Cpm in cmps: coordinated power management in chip-multiprocessors. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12. IEEE Computer Society (2010)
40. Sartori, J.; Kumar, R.: Distributed peak power management for many-core architectures. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1556–1559. European Design and Automation Association (2009)
41. Winter, J.A.; Albonesi, D.H.; Shoemaker, C.A.: 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, pp. 29–40. ACM (2010)
42. Hanson, H.; Keckler, S.W.; Rajamani, K.; Ghiasi, S.; Rawson, F.; Rubio, J.: Power, performance, and thermal management for high-performance systems. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pp. 1–8. IEEE (2007)
43. Cochran, R.; Hankendi, C.; Coskun, A.K.; Reda, S.: Pack & cap: adaptive dvfs and thread packing under power caps. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 175–185. ACM (2011)
44. Etinski, M.; Corbalan, J.; Labarta, J.; Valero, M.: Linear programming based parallel job scheduling for power constrained systems. In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 72–80. IEEE (2011)
45. Ye, R.; Xu, Q.: Learning-based power management for multicore processors via idle period manipulation. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(7), 1043–1055 (2014)
46. Shen, H.; Tan, Y.; Lu, J.; Wu, Q.; Qiu, Q.: Achieving autonomous power management using reinforcement learning. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **18**(2), 24 (2013)
47. Juan, D.-C.; Marculescu, D.: Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 97–102. ACM (2012)
48. ul Islam, F .M .M.; Lin, M.: Hybrid dvfs scheduling for real-time systems based on reinforcement learning. IEEE Syst. J. **11**(2), 931–940 (2017)
49. Wang, Z.; Tian, Z.; Xu, J.; Maeda, R.K.; Li, H.; Yang, P.; Wang, Z.; Duong, L.H.;. Wang, Z; Chen, X.: Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system. In: Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific, pp. 684–689. IEEE (2017)
50. Biswas, D.; Balagopal, V.; Shafik, R; Al-Hashimi, B.M.; Merrett, G.V.: Machine learning for run-time energy optimisation in many-core systems. In: 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1588–1592. IEEE (2017)
51. Kumar, N.; Vidyarthi, D.P.: A GA based energy aware scheduler for DVFS enabled multicore systems. Computing 1–23 (2017)
52. Zhu, K.; Ding, Y.: Research on low power scheduling of heterogeneous multi core mission based on genetic algorithm. In: 2017 9th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), pp. 219–223. IEEE (2017)
53. Dhiman, G.; et al.: System-level power management using online learning. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **28**(5), 676–689 (2009)

54. Kolpe, T.; Zhai, A.; Sapatnekar, S.S.: Enabling improved power management in multicore processors through clustered DVFS. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1–6. IEEE (2011)
55. Zhang, Z.; Chang, J.M.: A cool scheduler for multi-core systems exploiting program phases. IEEE Trans. Comput. **63**(5), 1061–1073 (2014)
56. Mitchell, T.M.: Machine learning. 1997, vol. 45, pp. 870–877. McGraw Hill, Burr Ridge (1997)
57. IBM: IBM CPLEX Optimizer (2017)
58. P. University: Parsec Benchmark (2017)
59. Bienia, C.; Kumar, S.; Singh, J.P.; Li, K.: The parsec benchmark suite: characterization and architectural implications. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 72–81. ACM (2008)
60. Binkert, N.; Beckmann, B.; Black, G.; Reinhardt, S.K.; Saidi, A.; Basu, A.; Hestness, J.; Hower, D.R.; Krishna, T.; Sardashti, S.; et al.: The gem5 simulator. ACM SIGARCH Comput. Archit. News **39**(2), 1–7 (2011)
61. I. Corp.: Intel Performance Counter Monitor–A Better Way to Measure CPU Utilization (2017)
62. P. University: Princeton's garnet network simulator (2017)
63. Kahng, A.B.; Li, B.; Peh, L.-S.; Samadi, K.: Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 423–428. European Design and Automation Association (2009)
64. Bartolini, A.; Cacciari, M.; Tilli, A.; Benini, L.; Gries, M.: A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In: Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI, pp. 311–316. ACM (2010)
65. Thoziyoor, S.; Ahn, J.H.; Monchiero, M.; Brockman, J.B.; Jouppi, N.P.: A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In: 35th International Symposium on Computer Architecture, 2008. ISCA'08, pp. 51–62. IEEE (2008)