Distributed Reinforcement Learning for Power Limited Many-core System Performance Optimization

Zhuo Chen
Electrical and Computer Engineering
Carnegie Mellon University
Email: zhuoc1@andrew.cmu.edu

Diana Marculescu
Electrical and Computer Engineering
Carnegie Mellon University
Email: dianam@cmu.edu

***Abstract*—As power density emerges as the main constraint for many-core systems, controlling power consumption under the Thermal Design Power (TDP) while maximizing the performance becomes increasingly critical. To dynamically save power, Dynamic Voltage Frequency Scaling (DVFS) techniques have proved to be effective and are widely available commercially. In this paper, we present an On-line Distributed Reinforcement Learning (OD-RL) based DVFS control algorithm for many-core system performance improvement under power constraints. At the finer grain, a per-core Reinforcement Learning (RL) method is used to learn the optimal control policy of the Voltage/Frequency (VF) levels in a system model-free manner. At the coarser grain, an efficient global power budget reallocation algorithm is used to maximize the overall performance. The experiments show that compared to the state-of-the-art algorithms: 1) OD-RL produces up to 98% less budget overshoot, 2) up to 44.3x better throughput per over-the-budget energy and up to 23% higher energy efficiency, and 3) two orders of magnitude speedup over state-of-the-art techniques for systems with hundreds of cores.**

I. Introduction

While historically the major goal of processor designers was to gain better performance by continuously shrinking device size, adding more pipeline stages, and speeding up the clock cycle, the power wall was eventually reached and energy has become the main design constraint.

The ever growing power consumption increases the burden of heat dissipation, lowers the chip reliability, and decreases battery life of mobile devices. As Dennard scaling breaks down, multi-core systems have become the solution to mitigate the high power problem. For highly parallel applications, multiple small cores with lower voltage and frequency can offer similar throughput as one large core at a much higher voltage and frequency. Indeed, the multi-core system consumes less power according to the V^2f Scale Law [1]. While multi-core systems are now mainstream market products, the desire for higher performance is again pushing the envelope toward higher power consumption. In order to ensure the safety and reliability of multi-core systems, a Thermal Design Power (TDP) constraint is imposed which the system power consumption should not exceed. As a result, improving performance under the TDP constraint becomes one of the main directions in power/performance optimization. In addition to the static way of increasing the number of cores, Dynamic Voltage Frequency Scaling (DVFS) is developed to save the power at runtime. By being smart in tuning the Voltage and Frequency (VF) levels of the cores, on-chip computation can be performed in a more energy efficient manner. However, finding the optimal VF level assignment can be formulated as an integer linear programming problem which is NP hard [2]. Therefore, the exact solution cannot be implemented as an on-line algorithm, especially when the number of cores scales up. Many algorithms have been proposed to find near-optimal solutions in polynomial time, however, they did not take the budget overshoot problem into consideration and may only be efficient for small-scale multi-core systems rather than systems with hundreds of cores. By exploiting both spatial and temporal hierarchy, we propose On-line Distributed Reinforcement Learning (OD-RL) that is able to improve the performance with much less TDP overshoot and smaller runtime overhead.

This research was supported in part by NSF CCF Grant No. 1314876 and NSF CNS Grant No. 1128624.

II. Related work and paper contributions

Multi-core system DVFS control for power management has been widely studied [1], [3], [4], [5], [6], [7], [8], [9]. Reinforcement Learning (RL) [10], [11], [12], [13] and supervised learning methods [14], [15], [16], [17] have recently been applied for single core systems or systems with a modest numbers of cores and are proved to be effective, but no scalability study exists. Isci *et al.* propose MaxBIPS which exhaustively searches for the best combination of VF levels that maximizes the performance under power constraint. However, MaxBIPS is not a scalable approach, although it does deliver good quality solutions. Winter *et al.* analyze the complexity of LinOpt [18] to be $O(N^4)$ where N is the number of cores, but the approach is unsuitable for many-core systems [6]. They further propose the Steepest Drop method which has a complexity of $O(\alpha \cdot N \cdot log(N))$ where α is the number of VF levels. Although they are solving the power limited performance improvement problem, none of them takes the budget overshoot into consideration. Authors of [16] coordinate multiple resources, including power constraints. Yet, the approach requires off-line profiling and training for an Artificial Neural Network. Although a RL-based approach has the strength of learning the model of the system and workload, none of the aforementioned contributions addresses the scalability of RL in many-core systems. In order to mitigate the scalability problem, Juan *et al.* [12] propose to group the cores into multiple clusters each managed by supervisors which suppress the actions back to nominal VF level whenever the budget is exceeded. Nevertheless, the recovery mechanism, the under-investigated granularity of the clusters and the equal-distribution of the power budget renders this method not optimal. There are also works on thermal constrained performance optimization [19], [20], [7]. However, they need separate procedures for model learning and action decision while RL learns the optimal actions in a model-free manner. Besides, the thermal constraint is usually local (per-core) which is very different from the global power constraint in our case. Since the problem scope is different, we cannot provide a direct comparison with these approaches. To the best of our knowledge, our work makes the following contributions:

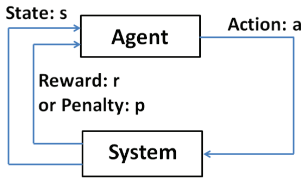

Fig. 1. Agent-system interaction in RL. The systems is characterized by state s. The action a is taken by an agent to change the state, with resulting outcomes r or p and new state s'

1. The RL method is known to be not scalable, since the number of states increases exponentially with the number of cores. In this work, we exploit the *spatial and temporal hierarchical* structure, and propose the OD-RL method that combines RL and an efficient power budget reallocation algorithm. The number of states of OD-RL therefore no longer depends on the number of cores, and the algorithm complexity is reduced dramatically to $O(N \cdot log(N))$ which is also independent of the number of VF levels. The proposed method **overcomes the scalability problem of RL** while maintaining its powerful aspect of learning the optimal action without building a full model of the system.

2. Power constrained performance improvement is usually solved by pushing the average power close to the TDP constraint, regardless of the resulting budget overshoot. Many methods do exceed the budget quite often and may also suffer from the inaccuracy of V^2f Scale Law. However, our approach is able to adapt to the workload, mitigate the inaccuracy problem and hence **suppress the budget overshoot** effectively.

3. We evaluate our proposed method, MaxBIPS and Steepest Drop with a wide spectrum of parallel, multi-threaded applications in systems with as many as 64 cores. The experimental results show up to **98%** less budget overshoot, **44.3x** better Throughput per Over-the-budget Energy (TOE), **23%** energy efficiency improvement and **100x** speedup in a 512-core system when compared with Steepest Drop [6] approach.

The remainder of this paper is organized as follows. Section 3 describes our proposed OD-RL method. Section 4 gives the experimental results and compares them with the state-of-the-art methods. Section 5 concludes our work.

III. Methodology

RL algorithms are developed to find the optimal solution to sequential decision problem, and have been proved effective in a variety of problems from different areas [21]. However, they often suffer from state space explosion and thus are too expensive to use in large scale problems. In this paper, we propose OD-RL method which assigns VF levels to each core to improve the global performance under a given power budget in a scalable manner. To solve the scalability problem of the RL algorithms, our method consists of a distributed RL algorithm working at finer grain and an efficient power reallocation algorithm working at coarser grain.

RL [21] is inspired by the trial-and-error method humans used for making decisions for millions of years. In RL, the agent interacts with the system (Fig. 1), *e.g.*, committing actions based on the state of the system and also observing the new state of the system. The goal of RL is to find the best actions under different states such that by following those best actions, the agent can optimize the long-term reward. The probability of selecting an action a under a state s is called a policy $\pi(s, a)$. RL determines how the agent changes its policy by experiencing the states. As the problem is sequential, the long-term reward starting from time t is defined as $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where r_i is the immediate reward at time i. Discount factor γ determines how important the future reward is. If $\gamma = 1$, future reward is as important as the immediate reward r_t. If $\gamma = 0$, the agent does not care about future rewards. $Q^\pi(s, a)$ is the expected R_t of starting from state s, taking action a and then following the policy π: $Q^\pi(s, a) = \mathbb{E}\{R_t | s_t = s, a_t = a\}$. The optimal Q value is defined as $Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$ which means the optimal Q value is obtained when we choose the policy that can maximize the long-term reward of $Q^\pi(s, a)$. Formally speaking,

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}\{r + \gamma \cdot \max_{a'} Q^*(s', a') | s_t = s, a_t = a\} \\ &= \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \cdot \max_{a'} Q^*(s', a')] \end{aligned} \tag{1}$$

where $P^a_{ss'}$ is the transition probablity (probability of becoming state s' by taking action a at state s), and $R^a_{ss'}$ is the expected reward of reaching state s' by taking action a at state s. This equation is known as Bellman optimality equation for Q^* [21]. It is always looking for the action a' that can maximize the Q^* value of the next state, and averaging over all the possible next states. However, we usually do not know the value of $P^a_{ss'}$ and $R^a_{ss'}$.

Q-learning [22] is one of the most important breakthroughs in RL [21], because (1) it converges to the Bellman optimal solution in an on-line and incremental manner; (2) it does not require the model of system, *e.g.*, prior knowledge of $P^a_{ss'}$ and $R^a_{ss'}$. The following equation gives the updating rule of Q-learning:

$$\begin{aligned} Q(s, a) &= Q(s, a) \\ &\quad + \theta \cdot \{r + \gamma \cdot \max_{a'} [Q(s', a')] - Q(s, a)\} \end{aligned} \tag{2}$$

The agent starts from state s and chooses action a. By observing the reward r and the next state s', $r + \gamma \cdot \max_{a'} [Q(s', a')]$ gives the expected long-term reward of the state-action pair (s, a). Then Q-learning updates the Q value incrementally, which is suitable when the agent experiences the state-action pairs sequentially, by using the difference $\{r + \gamma \cdot \max_{a'} [Q(s', a')] - Q(s, a)\}$. Repeating this procedure, Q-learning is guaranteed to converge to the Bellman optimal solution [22], *i.e.*, solution to equation (1).

Nevertheless, when we define the machine states and actions globally, the state space explodes exponentially. Therefore, if a centralized RL is used, the overall number of states and actions grows exponentially in the number of cores. To prove this exponential growth, we assume that the system has *N* cores in total, K features per core, *e.g.*, IPC, and D different values per feature, *e.g.*, high/low. The k_{th} feature of core i is denoted as f_{ik} and the global state is defined as $S_{global} = \{f_{11}, f_{12}, ..f_{1K} ... f_{N1}, ... f_{NK}\}$. The number of global states of a *N*-core system would be $|S_{global}| \propto D^{KN}$. The action of all *N* cores will be a vector $A_{global} = (a_1, a_2, ... a_N)$ where a_i is the action of the i_{th} core. Without loss of generality, we assume the same number of possible actions α for each core, then the number of possible values of global action vector $|A_{global}| \propto \alpha^N$. Accordingly, in the centralized RL method, the total number of state-action pairs would be $|S_{global}| \cdot |A_{global}| \propto D^{KN} \cdot \alpha^N$ which indeed grows exponentially in the number of cores *N*. One way to mitigate the exponential growth of the number of actions is to decompose the actions of different cores, in which case each core makes its own independent decision under the global state. The number of states is still the same as before while the number of action is reduced to αN from α^N in total, *i.e.*, α for each core. Therefore, for each core, the number of state-action pairs would be $|S_{global}| \cdot |a_i| \propto D^{KN} \cdot \alpha$, which is much smaller than the previous case but still exponential in the number of cores. The training time and memory overhead will render both approaches impractical in real application.

Nonetheless, by following the same path of decomposing the states, we are able to further reduce complexity. A globally defined state space contains more information than necessary, *i.e.*, in such a high dimensional state space, many of them are never reached. Therefore, one way to reduce the complexity is to eliminate the features of the other cores and retain only the features of the current

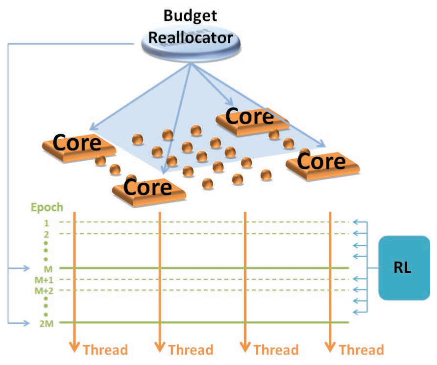

Fig. 2. Structure of the algorithm. Spatial hierarchy: budget reallocation does global coordination while RL is distributed to each core. Temporal hierarchy: budget reallocation works at coarser gain while RL executes at finer grain.

core. By doing so, the state of each core i is $S_i = \{f_{i1}, f_{i2}, ..f_{iK}\}$. The number of state-action pairs of each core is further reduced to $|S_i| \cdot |a_i| \propto D^K \cdot \alpha$, which is no longer a function of N. Consequently, a distributed RL algorithm allows every core to run independently and learn the optimal DVFS control policy on its own. Since the distributed RL method is highly parallel, all the cores can work simultaneously and hence the time complexity of this method is O(1). However, we actually achieve this high scalability at the cost of missing information on the other cores. To solve this problem, we propose to reallocate the power budget at a coarser grain in which all the cores communicate by means of receiving new power budget constraint.

Therefore, the problem of RL-based power constrained performance improvement is decomposed into two sub-problems.

1. At finer grain, given a fraction of total budget, each core learns the best policy that maximizes the performance under that budget, locally.

2. At coarser grain, reallocate the power budget to ensure a better utilization of the total budget, globally.

Fig. 2 depicts the overall structure of our algorithm. Spatially, distributed RL works on each core locally and independently, while the budget reallocator reallocates the budget among all the cores. Temporally, distributed RL operates at every control epoch, while the budget reallocator executes every M epochs. By doing experiments, we find $M = 15$ is the best value for most benchmarks.

A. *Finer Grain: Reinforcement learning algorithm*

At finer grain, each core will use Q-learning to learn the best policy. We define the state of machine as *S*={Instruction Per Cycle (*IPC*), Million L2-cache-misses Per Kilo-Instructions (*MPKI*), current power value (*Power*), current VF level (*VF level*)}. Traditionally, the features are discretized equally. Instead, we discretize feature IPC by its statistical distribution to minimize inter-interval oscillation due to poor discretization threshold. We define the reward r as the core throughput, which stands for the preference of the action that leads to higher performance. The budget-overshoot penalty is $-PF \cdot |power\ value - power\ budget|$ where PF is the penalty factor. The penalty is proportional to the value of budget overshoot, which follows the intuition that it is more desirable to eliminate a higher overshoot. We will decide PF value in section IV.B.

To accelerate the convergence of Q-learning, we propose a *batch-update method*. For any state s_0 and s_1, we define $s_1 > s_0$, when all the features of s_1 are larger than those of s_0. By analyzing the program trace, we observe that, if the core exceeds the budget at state-action pair (s, a), it will also exceed at $(S, a), \forall S > s$. Therefore, we penalize all the state-action pairs $(S, a), \forall S > s$, when the budget is exceeded at (s, a). In such a way, unnecessary budget overshoots are eliminated by taking advantage of the correlation between the states.

RL algorithm is able to suppress the budget overshoot by learning the transition probability of the workload. However, these actions can sometimes be conservative due to the abnormality in the data experienced. We observe in the experiments that high frequency selection is sometimes penalized due to abnormal spikes in power. We note that it would be possible to overshoot the budget for a short period, therefore we develop a memory mechanism to accommodate these spikes in a similar way as how branch predictor works. We maintain an m-bit memory queue for all the state-action pairs. If a state-action results in under the budget, the memory receives a 1. Whenever the same state-action pairs receives a 0, which means a budget overshoot, it will check the m-bit memory: (1) it will not penalize the pair when all the bits in memory are 1s. (2) it will penalize the pair when one of the bits is 0. We determine the best value of $m = 3$ by experiments. This memory mechanism can accommodate the abnormality and mitigate the performance degradation due to the workload idiosyncrasies while still being able to learn the periodic power spikes.

B. *Coaser Grain: Power budget reallocation*

As distributed RL maximizes the performance under the given budget at a finer grain, power budget reallocation algorithm will serve as a global coordinator to further maximize the performance as well as the power utilization at a coarser grain. Following the same intuition as MaxBIPS which favors the CPU-intensive threads, we propose the Maximize-the-Max (MM) method which always maximizes the VF level of the busiest cores. Algorithm 1 gives the pseudocode of MM method. Let's associate a tuple (core number, IPC) = (i, IPC_i) to each core i. Then we have a "core number-IPC" profile: $\mathbb{CORE} - \mathbb{IPC} = \{(0, IPC_0), (1, IPC_1), \ldots, (n-1, IPC_{n-1})\}$. Power of core i at VF level VF_i is $Pow(i, VF_i)$. At every power reallocation epoch, MM first estimates the power consumption of all the cores at their lowest VF level, and subtracts this value from the total budget. Then MM builds a max-heap out of $\mathbb{CORE} - \mathbb{IPC}$ based on the IPC value, and pops out the core with the highest utilization. The required budget to boost it to the highest VF level is calculated and then given to the core. If there is still residual budget after the first allocation, MM will pick the core with the second highest utilization value and will again allocate the budget to allow the highest VF level selection. MM repeats this process until no budget value is left. To implement this algorithm, we used the max-heap to sort the cores in the order of their utilization, with a time complexity of $N \cdot log(N)$. To estimate the power consumption of core i at a different VF level $VF_i = q$, we use the $V^2 f$ Scale Law [1] which states that dynamic power is proportional to $V^2 f$ where V is voltage and f is frequency. Assuming core i is originally at $VF_i = p$, we estimate

$$Pow(i, q) = Pow(i, p) \cdot \frac{Volt^2(q) \cdot freq(q)}{Volt^2(p) \cdot freq(p)} \cdot \beta(i) \tag{3}$$

where $\beta(i)$ is the discount factor of core i accounting for the transition cost of VF level [1], and $Volt(p)$ and $freq(p)$ are the voltage and frequency of VF level p, respectively. We determine $\beta(i) = 0.9\ for\ \forall i$ by experiments. Since subthreshold leakage power usually remains stable for a specific voltage, to further increase the power estimation accuracy, we subtract it from the total power first, then do the $V^2 f$ scale on dynamic power, and finally add the leakage power of the new voltage back to the total power. The subthreshold leakage power of different voltages are measured during machine idle period.

IV. Experimental Results

A. *Experiment Set-up*

We verify the effectiveness of the proposed method by using Sniper simulator [23] with Parsec and Splash-2 multi-threaded bench-

Algorithm 1 Pseudocode of MM method

```
1: Input: $\mathbb{CORE}-\mathbb{IPC}$, $Global\ Budget$, $N$
2: Output: $Budget_i$ for a core $i, 1 \le i \le N$
3: Variable: $Residual\ Budget$.
4: for $i = 1$; $i \le N$; $i++$ do
5:     $Budget_i \leftarrow Pow(i, VF_i = lowest)$
6: end for
7: $Residual\ Budget \leftarrow Global\ Budget - \sum_{i=1}^{N} Pow(i, VF_i = lowest)$
8: Built a max-heap of $\mathbb{CORE}-\mathbb{IPC}$ based on the IPC value
9: while $Residual\ Budget > 0$ do
10:     Pop the max-heap and get the tuple $(i, IPC_i)$
11:     $\Delta \leftarrow Pow(i, VF_i = highest) - Pow(i, VF_i = lowest)$
12:     if $\Delta \le Residual\ Budget$ then
13:         $Budget_i \leftarrow Budget_i + \Delta$
14:         $Residual\ Budget \leftarrow Residual\ Budget - \Delta$
15:     else
16:         $Budget_i \leftarrow Budget_i + Residual\ Budget$
17:         break
18:     end if
19: end while
```

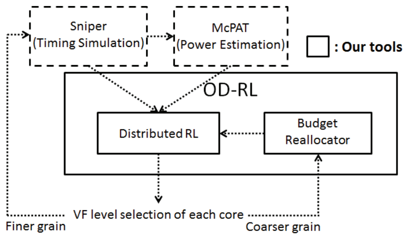

Fig. 3. Simulation infrastructure: Sniper for timing simulation and McPAT for power estimation. Distributed RL and budget reallocator are implemented as Python scripts to make the VF level decisions and interact with Sniper.

mark suites. The simulation infrastructure is shown in Fig. 3. Sniper is able to perform timing simulations for multi-threaded, shared-memory applications with tens to 100+ cores, and has been validated against Intel Core2 and Nehalem systems. Sniper uses McPAT [24] for system power estimation. The system configuration is shown in Table I. In Sniper simulator, our algorithm is implemented as a Python script. In every control epoch, this script will be invoked by Sniper and executed to select the VF level of all the cores. The epoch size varies from $500\mu s$ to $10ms$ across different benchmarks, because 1) we explore the effectiveness of our approach across different epoch lengths; 2) we are using the large input sets for the benchmarks and therefore, we need hundreds of epochs to demonstrate the effectiveness of the algorithm in a statistically significant manner.

TABLE I. ARCHITECTURAL PARAMETERS

Number of cores	8, 16, 32, 64
Architecture	Intel Gainestown
L1-I/D cache	32KB, 4-way, LRU
L2 cache	512KB, 8-way, LRU
L3 cache	8MB, 16-way, LRU
VF levels (GHz/V)	2.7/0.9, 3.0/1.0, 3.3/1.1, 3.6/1.2
Nominal VF level	3.0GHz/1.0V
Control epoch	$500\mu s$ to $10ms$
Reallocation period	every 15 epochs
Technology node	45 nm

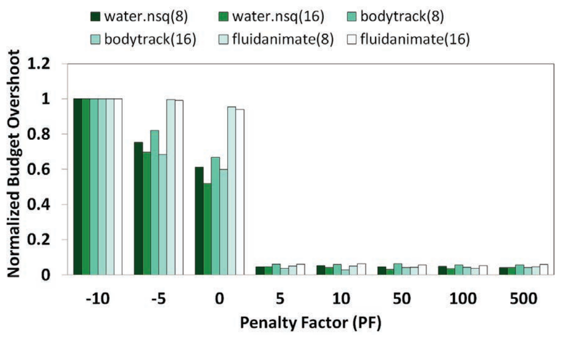

Fig. 4. Budget overshoot quickly satures when Penalty Factor (PF) is larger than 5.

B. Results and Analysis

The penalty in RL plays an important role in determining the algorithm behavior [21]. Higher penalty offers better overshoot suppression, but can also negatively impact the performance. Lower penalty is able to give better performance at the cost of more budget overshoot. In order to find the best Penalty Factor (PF) value, we studied the penalty impact on budget overshoot under different benchmarks and numbers of cores. The results in Fig. 4 show that the budget overshoot quickly saturates when $PF \ge 5$, therefore we select $PF = 5$ as it offers the best trade-off between penalty and performance.

We evaluate our algorithm together with two state-of-the-art algorithms: MaxBIPS [1] and Steepest Drop [6] in terms of their budget overshoot control, relative performance improvement, and runtime overhead. Budget overshoot is defined as the total energy consumption over the TDP. This extra energy increases the risk of thermal emergencies and subsequent throttling, negatively impacts the chip reliability, and puts heavier burden on the cooling system. Fig. 5 shows the budget overshoot control of three algorithms. Our algorithm is able to better suppress the budget overshoot in all but one case, by a margin of several orders of magnitude. In *swaptions*, we see that our approach does not perform as well as Steepest Drop and is close to MaxBIPS. The reason for this behavior is that *swaptions* is a very balanced multi-threaded application in which all cores have the same utilization across cores and in time. The exception is core 0 which does nothing with $IPC_0 = 0$. In this case, the inevitable on-line learning overhead of OD-RL increases the relative budget overshoot. In *Radiosity*, OD-RL saves **98%** budget overshoot due to its capability of adapting to the workload change and hence reducing the over-the-budget energy to a much lower value 0.06J compared to the other approaches. OD-RL outperforms the state-of-the-art approaches because it is able to learn the workload transition probability and therefore suppress the budget overshoot. We point out that less budget overshoot may lead to lower power consumption and lower performance. To make a fair comparison, rather than simply comparing the performance without taking budget overshoot into account, we have to use metrics that can show the relative values of performance, power and budget overshoot. To evaluate the relative performance under different budget overshoot, we propose the metric Throughput per Over-the-budget Energy (TOE). TOE measures how much performance we could gain by taking the risk of exceeding the budget by one unit of energy. Higher performance and lower over-the-budget energy will both lead to higher TOE value. As MaxBIPS and Steepest Drop are actually pushing the average power close to the budget line to achieve high performance regardless of the budget overshoot, we will also calculate $\frac{Throughput^2}{Average\ Power}(\propto \frac{1}{Energy-Delay\ Product})$ to evaluate the energy efficiency of all three methods. Fig. 6 and Fig. 7 compare the TOE value and energy efficiency of all three approaches under different benchmarks, respectively. An 8-core system is used here because

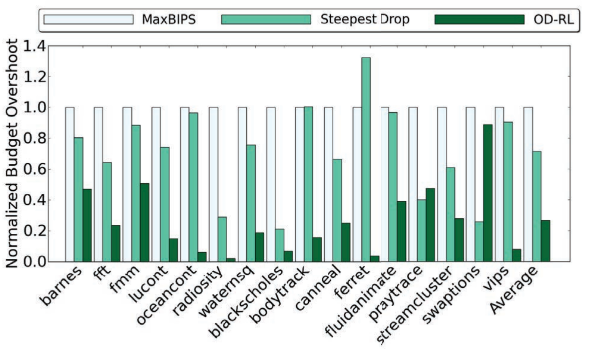

Fig. 5. Comparison among three algorithms on budget overshoot control (Lower is better). Results normalized with respect to MaxBIPS.

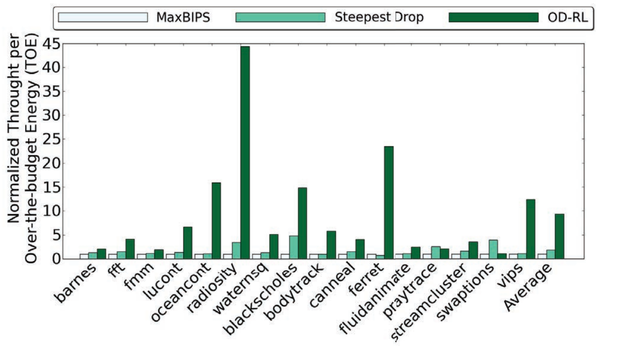

Fig. 6. Comparison among three algorithms on Throughput per Over-the-budget Energy (TOE) (Higher is better). Results normalized with respect to MaxBIPS

MaxBIPS (which uses exhaustive search) is not scalable and thus requires too much time in a system with more cores. *Radiosity* shows a high TOE improvement around **45x** because OD-RL almost never exceeds the budget (0.06J over-the-budget energy). OD-RL adapts to the workload change and gives moderate performance improvement with much less over-the-budget energy. Therefore, it gives better TOE and energy efficiency values across all the benchmarks. Fig. 7 gives the comparison of energy efficiency. OD-RL also provides up to **23%** higher energy efficiency, and is slightly better than the other approaches on average because MaxBIPS is already near optimal on average [1]. To show how OD-RL performs when the number of cores scales up, we compare Steepest Drop and OD-RL for a system

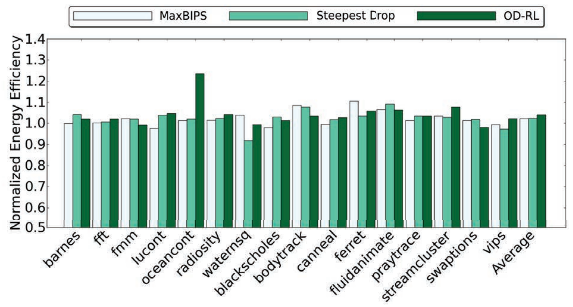

Fig. 7. Comparison among three algorithms on energy efficiency (Higher is better). Results normalized with respect to MaxBIPS.

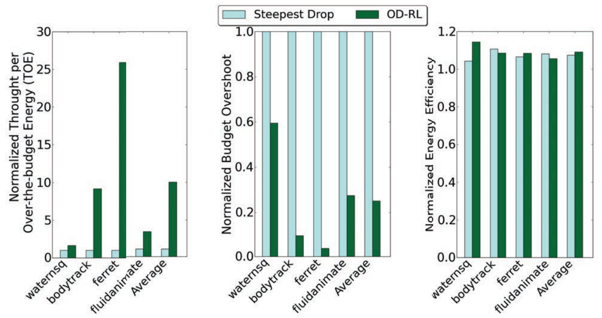

Fig. 8. Comparison of TOE (left), budget overshoot (middle) and energy efficiency (right) in a system of 16 cores.

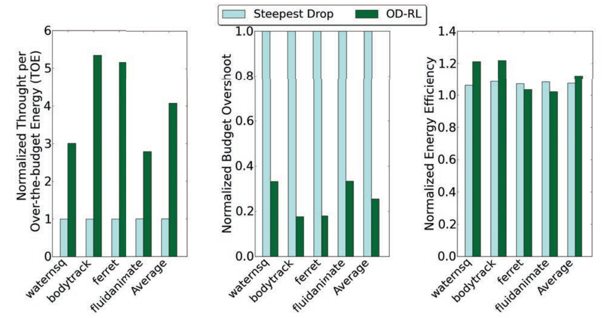

Fig. 9. Comparison of TOE (left), budget overshoot (middle) and energy efficiency (right) in a system of 32 cores.

of 16, 32 and 64 cores. We only compare Steepest Drop and OD-RL here because MaxBIPS is unable to run for more than 8 cores. *Waternsq*, *bodytrack*, *ferret* and *fluidanimate* are used here because they are the most representative for our work. Figure 8, 9 and 10 show that OD-RL is constantly better when the number of cores scales up. OD-RL, on average, achieves **5-10x** better TOE value, around **70%** less budget overshoot and **3%** better energy efficiency. The average improvement of three metrics over different number of cores is relatively stable, respectively, except TOE in a 16-core system. That's because Steepest Drop exceeds the budget more than 40% of the time with relatively less performance improvement. The consistent improvement across different number of cores demonstrates that OD-RL method is highly scalable and largely independent on the number of cores. Based on the analysis of the algorithm trace, we have the following observations: 1) MaxBIPS and Steepest Drop heavily rely on the accuracy of the V^2f Scale Law. Inaccurate power estimation will make them choose actions that result in budget overshoot. Although OD-RL also suffers from that inaccuracy only at the budget reallocation epoch, the RL will automatically adapt to the new budget and suppress the budget overshoot. 2) Even with a perfect power estimation, MaxBIPS and Steepest Drop still suffer from making decisions based on the current machine state regardless of the next machine state. For example, let's assume a machine state s_0 at time t_0 and s_1 at time t_1 with $s_1 > s_0$. Here larger s means a higher machine utilization. At time t_0, MaxBIPS and Steepest Drop will calculate the best actions based on s_0 which should not exceeds the budget. Unfortunately, s_0 quickly changes to s_1 under which the chosen actions cause a higher power consumption which actually exceeds the budget. The reason that RL performs better is that it has the capability of learning the transition probability of the machine state, and therefore, is able to predict the next machine state and choose the best action based on both current and future state.

We further analyze and test the overhead of three approaches. MaxBIPS uses exhaustive search and its complexity is $O(N \cdot \alpha^N)$ where α is the number of VF levels. Steepest Drop is much more

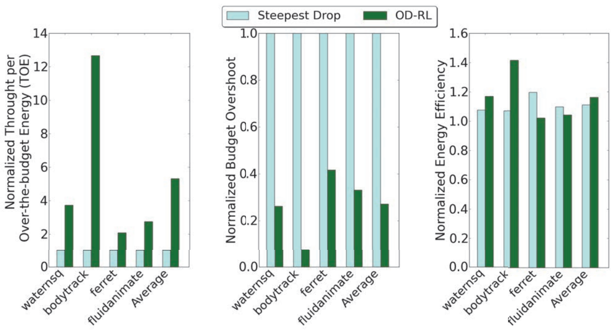

Fig. 10. Comparison of TOE (left), budget overshoot (middle) and energy efficiency (right) in a system of 64 cores.

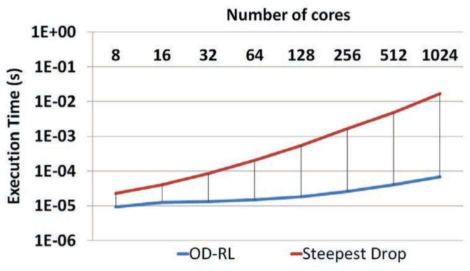

Fig. 11. Log-log plot of approaches overhead in a system up to 1024 cores. Execution time is averaged over 100 runs.

scalable with a complexity of $O(\alpha \cdot N \cdot log(N))$. In OD-RL, the distributed RL method is highly parallel and the only phase requiring global coordination is the power budget reallocation. MM actually does a heapsort with respect to the IPC value, and in worst case its complexity is $O(N \cdot log(N))$ [25]. Therefore, MM has a better scalability in terms of more VF levels. Besides, since OD-RL invokes the power reallocation algorithm in a coarser grain, it further reduces the runtime overhead by a constant factor asymptotically though not changing the complexity. Fig. 11 shows the execution time (log scale) of the Steepest Drop method and OD-RL with number of cores up to 1024. They indeed follow the $O(N \cdot log(N))$ trend. The execution time is averaged over 100 runs. The results show that OD-RL can achieve **100x** speedup in a 512-core system when compared to Steepest Drop.

V. Conclusion

In this work, we propose an On-line Distributed Reinforcement Learning-based approach which decomposes the original power constrained, performance optimization problem into two sub-problems at different spatial and temporal granularities. The RL at finer grain is able to improve the performance while suppressing the budget overshoot by learning and adapting to the workload change. Maximize-the-max (MM) method at the coarser grain is more scalable in terms of the number of VF levels compared to Steepest Drop, and can be asymptotically faster as a result of infrequent execution. It is **100x** faster in a 512-core system. Our approach achieves up to **98%** saving on budget overshoot, **44.3x** TOE and **23%** higher energy efficiency, and is consistently better when the number of cores scales up.

References

[1] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *MICRO*. IEEE Computer Society, 2006, pp. 347–358.

[2] G. Liu, J. Park, and D. Marculescu, "Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems," in *ICCD*. IEEE, 2013, pp. 54–61.

[3] J. Li and J. F. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *HPCA*. IEEE, 2006, pp. 77–87.

[4] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: adaptive dvfs and thread packing under power caps," in *MICRO*. ACM, 2011, pp. 175–185.

[5] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ISLPED*. IEEE, 2007, pp. 38–43.

[6] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *PACT*. ACM, 2010, pp. 29–40.

[7] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *ACM SIGARCH computer architecture news*, vol. 37, no. 3. ACM, 2009, pp. 314–324.

[8] J. Sartori and R. Kumar, "Distributed peak power management for many-core architectures," in *DATE*. IEEE, 2009, pp. 1556–1559.

[9] J. M. Cebrián, J. L. Aragon, and S. Kaxiras, "Power token balancing: Adapting cmps to power constraints for parallel multithreaded workloads," in *IPDPS*. IEEE, 2011, pp. 431–442.

[10] W. Liu, Y. Tan, and Q. Qiu, "Enhanced q-learning algorithm for dynamic power management with performance constraint," in *DATE*. European Design and Automation Association, 2010, pp. 602–605.

[11] Y. Tan, W. Liu, and Q. Qiu, "Adaptive power management using reinforcement learning," in *ICCAD*. ACM, 2009, pp. 461–467.

[12] D.-C. Juan and D. Marculescu, "Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors," in *ISLPED*. ACM, 2012, pp. 97–102.

[13] G.-Y. Pan, J.-Y. Jou, and B.-C. Lai, "Scalable power management using multilevel reinforcement learning for multiprocessors," *ACM TODAES*, vol. 19, no. 4, p. 33, 2014.

[14] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *ICCAD*. ACM, 2006, pp. 747–754.

[15] H. Jung and P. M., "Improving the efficiency of power management techniques by using bayesian classification," in *ISQED*. IEEE, 2008, pp. 178 – 183.

[16] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *MICRO*. IEEE Computer Society, 2008, pp. 318–329.

[17] D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling," in *CODES+ ISSS*. IEEE, 2013, pp. 1–10.

[18] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 363–374.

[19] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries, "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*. ACM, 2010, pp. 311–316.

[20] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores," in *DATE*. IEEE, 2011, pp. 1–6.

[21] A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011.

[24] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO-42nd*. IEEE, 2009, pp. 469–480.

[25] J. W. J. Williams, "Algorithm-232-heapsort," pp. 347–348, 1964.