Reinforcement Learning For Thermal-Aware Many-Core Task Allocation

Shiting (Justin) Lu, Russell Tessier, and Wayne Burleson
University of Massachusetts
Department of Electrical and Computer Engineering
Amherst, MA, USA
{jlu, tessier, burleson}@ecs.umass.edu

ABSTRACT

To maintain reliable operation, task allocation for many-core processors must consider the heat interaction of processor cores and network-on-chip routers in performing task assignment. Our approach employs *reinforcement learning*, a machine learning algorithm that performs task allocation based on current core and router temperatures and a prediction of which assignment will minimize maximum temperature in the future. The algorithm updates prediction models after each allocation based on feedback regarding the accuracy of previous predictions. Our new algorithm is verified via detailed many-core simulation which includes on-chip routing. Our results show that the proposed technique is fast (scheduling performed in < 1 ms) and can efficiently reduce peak temperature by up to 8^oC in a 49-core processor (4.3^oC on average) versus a competing task allocation approach for a series of SPLASH-2 benchmarks.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]: Miscellaneous

General Terms

Performance, Design, Reliability

Keywords

Thermal aware, task allocation, reinforcement learning

1. INTRODUCTION

The thermal behavior of many-cores has grown to become a major performance and fault tolerance concern. High power density impacts circuit reliability and chip lifetime and can lead to the frequent initiation of remediation techniques such as DVFS. Although chip cooling technologies are frequently deployed, system-level thermal management techniques are necessary to prevent thermal emergencies and maintain high processor performance.

GLSVLSI'15, May 20–22, 2015, Pittsburgh, PA, USA.

http://dx.doi.org/10.1145/2742060.2742078.

Data intensive applications implemented on many-cores benefit from low latency and high bandwidth on-chip communication. The heat dissipated by the routers not only affects router temperature, but also the temperature of neighboring cores. Effective task scheduling for thermal management considers power dissipated in all many-core components, including NoC routers. In this paper, a new task allocation scheme based on machine learning is described which considers both processor core and NoC router temperatures. The allocation scheme aims to reduce the peak temperature of the chip, a leading cause of device failure. In performing task assignment, the allocator uses *current* chip temperature information and stochastic *predictions* about how each possible task assignment is likely to affect maximum many-core temperature in the future. During the next allocation interval, the accuracy of the previous prediction is checked and the model used to make the prediction is updated. As allocation proceeds, the accuracy of the model converges. This *reinforcement learning* is adaptive and scalable and it results in rapid convergence within about 200 task allocations.

Our work is supported by careful thermal analysis of typical many-cores and their reaction to task allocation. It is observed that many-cores running the same workloads but with different allocations can have a hot spot temperature difference of up to 8^oC due to heat interactions with neighboring cores. Experiments show that our approach can reduce peak temperature by 4.3^oC on average for a many-core which is moderately loaded versus a previous approach [1].

2. BACKGROUND

Approaches to reduce many-core overheating have been developed using a variety of techniques including remediation (e.g. DVFS), thermal-aware floorplans, and run-time task allocation. Issues impacting thermal-aware task allocation for core- and network-level management are summarized below.

Thermal Constraint Optimization: For multi-core systems, thermal-aware task allocation assigns tasks to cores to optimize thermal conditions. Different task distributions can result in substantially varied thermal chip profiles [2], so task allocation must consider the spatial thermal correlation of cores, caches, NoC routers and other processor components. A number of popular many-core allocation techniques [2] [3] use objective functions to minimize maximum temperature subject to constraints to meet physical design limits. Since it is computationally expensive in such systems to model the thermal behavior of all system components, in

general, existing allocators do not consider many-core router temperatures or the thermal impact of earlier allocation decisions while performing task assignment.

In Yu *et al.* [2], a thermal-aware allocation technique was developed that focused on maximizing computation throughput and meeting the peak temperature requirement for 3D architectures. Similarly, K-means clustering was used to perform multi-core task scheduling in Yeo *et al.* [4] with a goal of temperature reduction. In Coskun *et al.* [3], the authors formulated multi-core task scheduling statically as an integer linear programing problem by taking temperature into account. Hanumaiah *et al.* [5] provided an online thread migration technique focused on thermal balancing. Our approach differs from these previous approaches by learning from and adapting to thermal dynamics rather than relying on static thermal models. Our technique implicitly includes the thermal impact of circuit components and also considers the impact of thermal interface materials and cooling conditions, among other factors.

Adaptive Random: To evaluate the effectiveness of our new approach in Section 5, we compare against a widely-used thermal-aware allocation algorithm. The heuristic *adaptive random* algorithm selects the coolest core for task allocation [1] under a set of calculated probabilities. Potential allocations to cores are assigned weights based not only on the current temperatures, but also on their thermal history. These weights measuring thermal history are adjusted in real time as the cores execute a dynamic workload. Stochastic assignment is employed to allocate a new task to a core based on its current temperature, thermal history, and the thermal conditions of neighboring cores. The storage of thermal histories incurs memory cost. The temperatures for each core are stored for a period of time (1∼10K samples per core) to capture the thermal characteristics for specific workloads.

Thermal Impact of Network-on-Chip: Hot spots can occur in routers when heavy data traffic is present. If the system is dedicated to a specific application, static task mapping on NoC systems can achieve thermal balance over the on-chip network [6]. In a general-purpose system, tasks are dynamically mapped to cores, so heat generated by the NoC should be carefully managed by routing packets in a thermally-aware fashion [7]. However, these approaches do have limitations: (a) the thermal impact of processor cores is underestimated or ignored and (b) application-specific designs are employed, so they have limited generality.

Machine learning for thermal and power management: Ge and Qiu [8] proposed a temperature reduction technique based on reinforcement learning for media applications. The agent learns the workload and dynamically adjusts frequency to control thermal violations. Similar techniques were applied in a power management context [9] [10]. Chen *et al.* [11] proposed workload allocation based on reinforcement learning to reduce the peak temperature in a data center. The approach avoids local heating by assigning workloads in a spatially dispersed fashion.

3. THERMAL AWARE TASK ALLOCATION USING REINFORCEMENT LEARNING

This section introduces a task allocation approach based on reinforcement learning (RL). After performing an allocation, the approach "learns" how well its core selection was during the previous allocation, and updates decision-making models accordingly. This technique effectively *rewards* good allocation choices. Thus, the two main contributions are (1) the use of rewards in determining task allocation and (2) the inclusion of localized NoC router temperatures in making allocation decisions for neighboring cores.

3.1 Thermal-Aware Task Allocation Overview

At a high level, the steps taking place during each task allocation can be described as follows: (1) Temperature readings are collected from temperature sensors located in each processor core; (2) The maximum temperature T_{max} among all sensors is recorded; (3) The temperature values are used to determine the best assignment of a task to an idle processor core based on a temperature-based *utility function*; (4) The model used to formulate the utility function is updated.

The *utility function* effectively determines which assignment is likely to affect the maximum temperature of the chip the least. This effect is determined by considering the processor core's instantaneous temperature and the temperature of the attached router and surrounding cores. The formulation of the utility function and its model is based on RL.

3.2 Reinforcement Learning

In RL [12], an agent (the task allocator in our case) explores an environment by taking actions and observing the resultant reward. The reward of a particular action (assigning a task to a specific core) reflects the metric to be optimized (maximum temperature). For our system, as task allocations are performed, the model used to make assignments is refined in a learning process. The task allocator gradually refines the model based on temperatures measured a time period after an allocation is performed. Effectively, the allocator *learns* how to respond to a specific environmental condition (e.g. temperatures measured from temperature sensors) based on the results of previous assignments when cores and routers had a similar temperature profile. Formally, reinforcement learning consists of the following

- A set of environment states: $\mathcal{S}$, in this case temperature readings from sensors;
- A set of available actions on the current state : $\mathcal{A}$, task assignments to specific cores;
- A rule to evaluate the reward for taking the action at a specific state: $\mathcal{R}$;
- The goal is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, i.e. what action (assignment) should be taken at the current environmental (temperature) state.

A *utility function* can be developed to allow for the desired mapping. Our *Q learning* approach provides a reinforcement learning formulation for task allocation. A utility value is defined to find a policy π for Q learning as follows.

$$Q(s,a) = E\left[\sum_{i=0}^{\infty}(\gamma^i r_{t+i}|s_t = s, a_t = a)\right] \qquad (1)$$

The utility $Q(s,a)$ indicates the expected temperature *rewards*, r, (both present, $i = 0$, and future) which can be obtained by performing task assignment action a for temperature vector state s at time step t. Predicted future rewards are discounted via a discount factor γ. The optimal policy takes action which maximizes the utility Q. During each task assignment at time $t+1$, utility Q for temperature vector s and assignment a in (1) can be approximated

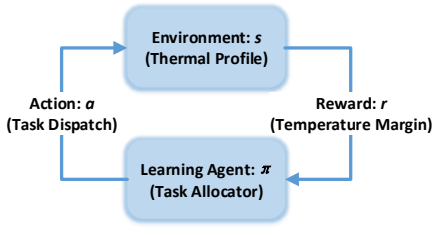


Figure 1: RL cycle for thermal-aware task allocation

as follows [12]:

$$Q_{t+1}(s_{t+1},a) = Q_t(s_t,a) + \alpha(\underbrace{r_t(s,a)}_{\substack{\text{current}\\ \text{reward}}} + \gamma \underbrace{\max_{a'} Q_t(s_{t+1},a')}_{\text{future reward}} - Q_t(s_t,a)) \tag{2}$$

In (2), α is the *learning rate*, which helps control the convergence of the algorithm. The new utility includes a combination of the previous utility, the current reward, and a discounted version of the future reward (Q_t values for other processors a'). This iterative equation is known to converge to an optimal point [12]. Fig. 1 shows the iterative reinforcement learning process. Each cycle represents one task allocation. In our implementation, the task allocator serves as the learning agent and the environmental state is the chip thermal profile which is read from on-chip temperature sensors. Task allocation decisions impact the thermal condition of the whole chip. After each task allocation, the allocator collects system thermal information to assess the reward of the *last* allocation action and to select a core for the next task. The details on how to apply Q and update the model used to determine it are discussed subsequently.

3.3 Definitions

On-chip thermal sensors are often deployed in processors to assist thermal management [13]. In our approach, a set of thermal sensor readings from m temperature sensors are used to represent the thermal state of the silicon. To apply RL in thermal-aware task allocation, the environment state is a temperature vector,

$$s = \left[s^1, s^2, s^3, ..., s^m\right]. \tag{3}$$

Each temperature value in the vector is a temperature reading from one of m different on-chip thermal sensors deployed in different locations on the chip. In a many-core, the task allocator is implemented in a dedicated core which typically performs other system-level management functions. This core dispatches new tasks to other available, working cores. So a task allocation selects a specific, idle core for a task execution. If processor cores are indexed from 1 to n, the possible actions are defined as:

$$\mathcal{A} = \{1, 2, 3, ..., n-1, n\} \tag{4}$$

where $\mathcal{A}$ represents all possible assignments to processors. An assignment to a specific processor is an action a.

The construction of the reward function is a key step in effectively performing RL-based allocation. Since the *maximum* temperature adversely impacts the performance and reliability of a many-core, reduction of this value is the goal. Generally, a many-core will have a pre-set *emergency* threshold temperature T_{em} which serves as a temperature boundary. System remediation (e.g. voltage scaling, task migration) is needed if a maximum temperature passes this point. We define the reward of an action as the difference between the emergency temperature and the current peak temperature T_{max}:

$$r = T_{\text{em}} - T_{\text{max}} \tag{5}$$

The higher the reward, the bigger the temperature margin.

3.4 Utility Function Approximation

In our approach, a processor which has a high utility Q value is more likely to receive a task assignment. If the number of possible temperatures for a core and associated router is relatively small, the utility function $Q(s,a)$ in (2) can be represented as a lookup table using temperature vector s and target processor core a as inputs. In other words, for every input temperature vector s, a Q value which has been previously determined and refined for a core a can be identified and used to make the current allocation decision. This approach leads to two issues: (1) Q values must be learned over time and stored in the lookup table and (2) temperature readings can span a large range of continuous values that would have to be discretized. As the state space of temperatures becomes large, using a lookup table for Q learning becomes intractable due to memory limitations and the difficulty of updating it in a timely fashion. Therefore, a continuous function is needed to map state-action (temperature-target processor) pairs to Q values.

Due to the high complexity of (2), it is not realistic to find a closed form representation for the Q function. However, the function can be approximated by the linear combination of a series of basis functions, $\phi_i(s)$.

$$Q(s,a) = \sum_{i=0}^{k} \theta_i^a \times \phi_i(s) \tag{6a}$$

$$a = 1, 2, ..., n \tag{6b}$$

Here, θ_i^a are k weight parameters for core a that are refined after each allocation to the core (k is defined in the next section). Each task assignment to a core (e.g. an action) corresponds to a set of weight parameters θ_i^a for core a. Following the updating of Q values at time $t+1$, weight parameters (θ_i^a) for the processor selected during the previous allocation t are updated according to the gradient descent technique [14]. Here a' includes all cores except a.

$$\theta_i^a(t+1) = \theta_i^a(t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a))\phi_i(s_t) \tag{7}$$

3.5 Basis Function

The basis function for Q value function approximation is a radial basis function (RBF), defined as follows:

$$\phi_i(s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-||c-s||^2/2\sigma^2} \tag{8}$$

$$c = \left[c^1, c^2, c^3, ..., c^m\right] \tag{9}$$

The formulation includes constant parameters σ^2, a scalar, and m-element vector c. The elements of $c-s$ provide context regarding the temperature difference between sensor readings s and typical temperature measurements c. We assume that the usual working temperature range of a chip is $[330, 360]$ Kelvin. Temperature *centers* are specified in

Algorithm 1 RL-based Task Allocation Algorithm

1: Initialize weight parameters $\theta_i^a \leftarrow 0$;
2: Read temperature values s from temperature sensors;
3: Apply a random task allocation;
4: **for** each task allocation episode **do**
5: Get current temperature values s_{t+1};
6: Calculate reward function for the last action based on (5);
7: For state s_{t+1}, calculate utility value, $Q(s_{t+1}, a)$, for all processors a;
8: Find maximum Q value from the above step and update the selected processor for the task: $a \leftarrow \max_{a'} Q(s_t, a')$;
9: Update weight parameters for θ_i^a according to (7);
10: Apply action a with probability p or an alternative action with probability $1 - p$;
11: **end for**

this range. The value of each of the elements in c is defined as one of v temperature centers within this range:

$$c^1, c^2, ..., c^m \in \{340, 350\} \quad (10)$$

The above example shows $v = 2$ centers. Since each of the values in the c vector can take on any of the v values, there are $k = v^m$ combinations for the c vector. Thus, v^m basis functions are available to approximate one Q function for each processor a. Since there are n total processors (possible allocation actions), the total number of parameters is $n * v^m$. Note that v is generally quite small (2 or 3).

3.6 RL-based Task Allocation Algorithm

In a many-core system, the task dispatcher is responsible for monitoring thermal state and assigning tasks to cores. Algorithm 1 describes the task allocation procedure performed by the dispatcher. Initially, the θ_i^a weight parameters of the Q function (6a) are initialized to zero. Steps 5-10 are performed for each task allocation. An allocation can be invoked when a new task arrives or an overheating situation is detected and a task must be migrated. The reward is calculated for the last allocation action at Step 6 and current thermal states are obtained by collecting temperatures from thermal sensors at Step 5. Steps 7 and 8 determine the task assignment to an idle core (action a) which leads to the maximum Q. Information is updated once the appropriate allocation action is determined. Our technique is stochastic, i.e. the determined action is taken with probability $p = 0.9$, a value determined empirically. Other assignments for a specific allocation are applied with equal probability of $\frac{1-p}{n-1}$. Using this approach, potentially good actions are not excluded and the environment is extensively explored.

4. EXPERIMENTAL APPROACH

Power, temperature and performance simulation was used to verify the effectiveness of our new task allocation scheme and to perform comparisons to *adaptive random* task allocation. In this section, detailed descriptions of the simulation platform, task models and the simulator flow are presented.

Many-core Floorplan: A mesh topology was used to build many-core systems for verification. Both routers and processor cores in 45 nm technology were evaluated. McPAT [15] was used to estimate the area and power consumption for the ten architectural component in the processor based on the parameters in Table 1. DSENT [16] was used to estimate the area and power for the router based on the configuration in Table 2 and estimated router usage based on traffic flow in the floorplan. HotFloorPlan was used to generate the floorplan. It was fed with core and router area information to generate the floorplan for a single core. The many-core floorplan was obtained by replicating single-core building blocks.

Table 1: Core Config.

L1-I	16KB
L1-D	16KB
L2	256KB
ITLB	16 entries
DTLB	16 entries

Table 2: Router Config.

port number	5
frequency	2.0 Ghz
VC per port	8
flit size	144 bits
buffer length	24 flits

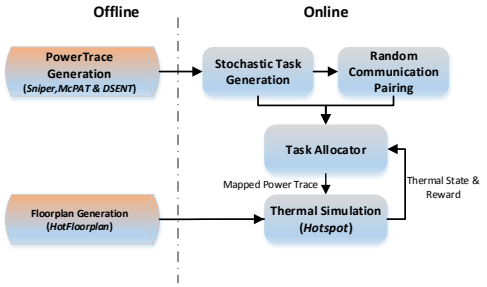


Figure 2: Simulation flow for thermal-aware task allocation experimentation

Benchmark Workload: Twelve benchmarks from the SPLASH-2[1] suite were used to test our platform. Each allocated task consists of an instantiation of one of the benchmarks. Communication between tasks was randomly assigned. To determine dynamic temperature values during many-core execution, power values for all processor core components and associated routers were determined. The power traces of the SPLASH-2 benchmarks were captured using the McPAT-integrated Sniper simulator [17]. HotSpot[2] was used to convert power values into temperature values. We use an $M/M/c$ queuing model to mimic the task arrival and task execution duration in the many-core system. In this model, task arrival is modeled as a Poisson process whose inter-arrival time is exponentially distributed; the execution time of tasks is also exponentially distributed. The number of cores in the system is n. The task arrival rate is defined as λ and the service rate is defined as μ. The system utilization, ρ, is given by $\rho = \frac{\lambda}{n\mu}$. Effectively, $n\rho$ defines the steady-state number of processor cores which is used to service tasks.

Simulation Flow: The simulation flow is shown in Fig. 2. As a first step, the power traces for each benchmark for a single-core floorplan are generated. Router power consumption under different loads is also calculated. The task allocator is implemented in conjunction with HotSpot which reports simulated chip temperatures. The task allocator retrieves temperature points which represent the s vector in

[1]http://liuyix.org/splash2-benchmark
[2]http://lava.cs.virginia.edu/HotSpot

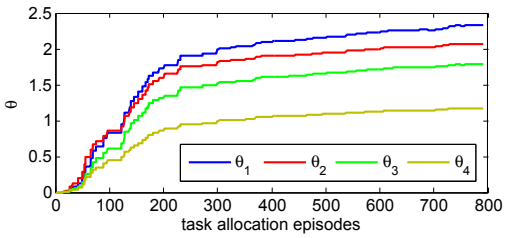


Figure 3: Convergence of θ_i values for a 25-core processor

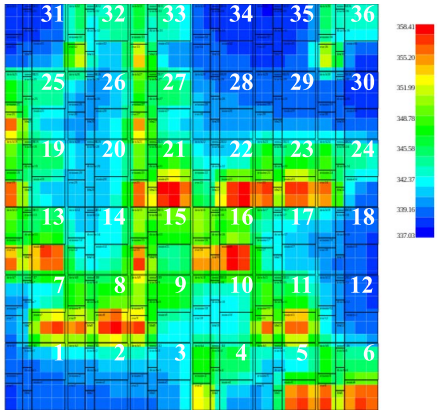


Figure 4: A thermal map snapshot for a 36-core system

(3). When a new task is generated for allocation, its communication is paired with other tasks. The appropriate power trace for the task is identified and mapped onto the floorplan. The HotSpot simulator reads the mapped power trace and performs thermal simulation. The maximum temperature is then reported to the allocator for reward calculation.

5. RESULTS

Our approach has been validated via simulation using 16-core, 25-core, 36-core and 49-core systems. To implement the reinforcement learning technique, the learning rate is set to $\alpha = 0.8$ and the discount rate is set to $\gamma = 0.8$ (Section 3.2). These parameters were determined empirically.

5.1 Effectiveness Validation

The convergence of our reinforcement learning model was evaluated over a series of task allocations. In the experiment, task allocations of randomly-selected SPLASH-2 benchmarks were performed to all 25 processor cores. A selection of θ_i values is shown in Fig. 3. Initially, θ_i values are all zeros and they begin to converge after $200 \sim 300$ allocation episodes. Other θ_i values showed similar behavior.

Fig. 4 shows a thermal snapshot of a 36-core processor at the seven minute time point. Q values are calculated for all possible allocation choices at this time point. Fig. 5 shows the magnitude of Q values for corresponding cores indexed in Fig. 4. As seen in the figure, the non-zero Q value for action 22 is lowest among all actions because the heat stress for core 22 is significant. An allocation to core 22 will negatively impact the chip peak temperature. Actions 3, 18, 29 and 35 have relatively high Q values. From the chip thermal map, it is seen that cores 3, 18, 29 and 35 are relatively cool and their neighboring cores are also in

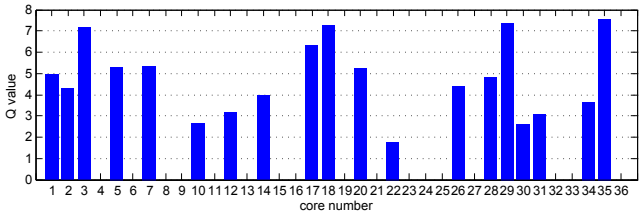


Figure 5: Q values for different actions at the thermal state given in Fig. 4.

a favorable thermal condition. We also notice that cores 1 and 31 are cool, but their Q values are not as high as the previous four cores. These two cores are in the corner of the chip and the thermal conductivity of air is much lower than silicon. The allocator effectively learns this information over time via reinforcement.

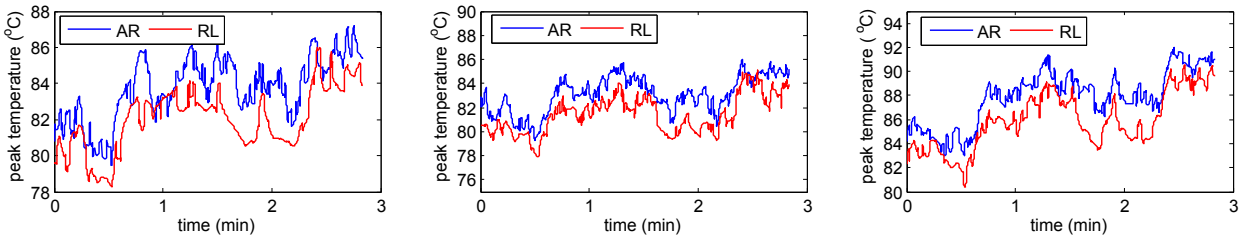
5.2 Peak Temperature Reduction

The peak temperature can be effectively reduced versus previous approaches through the use of the proposed allocation technique. A series of experiments are conducted to observe the peak temperature of the chip in comparison with the *adaptive random* approach [1] over five minute execution runs. This allocator assigns tasks to one of the coolest available cores based on probabilities determined from core temperature histories. In our implementation of the adaptive random approach, we also included the impact of router temperatures on allocation to allow for a fair comparison versus reinforcement learning. Table 3 shows the average peak temperature over time for different core counts and system utilization ratios for the two approaches. For a low system utilization (ρ), adaptive random and reinforcement learning have almost the same performance in terms of peak temperature. In this case, chip temperature is impacted more by the workload intensity inside specific cores and less by the distribution of tasks. Our technique performs better when the system is moderately loaded (about half used). For example, compared to adaptive random, our approach reduces peak temperature by 8.1^oC in a 49-core system (4.3^oC across all many-core configurations for $\rho = 0.4$ and 0.6). Fig. 6 shows the differences in the peak temperatures over time between the two approaches for 16-, 25-, and 36-core systems. Comparisons between reinforcement learning and adaptive random indicate that the former approach is more effective.

The time cost of the reinforcement learning allocator was evaluated for different numbers of sensors, m, and two sets of temperature centers, v. Table 4 shows the average computational time for one task allocation. The temperature centers used in (8) are selected from two sets, $\{340, 350\}$ and $\{335, 345, 355\}$, respectively. For most cases, the computational time of task allocation is $< 1\ ms$. Since the task allocation is only invoked when there is an incoming task or a thermal emergency, the frequency of allocation is typically on the order of seconds. Therefore, the percentage time cost of allocation with respect to the allocation interval is negligible. Although the adaptive random technique also has very low overhead time cost ($<< 1\ ms$) for each task allocation, it must track temperature sensor readings over time (one sample per 100 ms) regardless of task allocation rate. As a result, the technique becomes less favorable in a system which has a low task arrival rate.

Table 3: Peak temperature comparison between reinforcement learning (ours) and adaptive random [1]. The value ρ indicates the average number of cores used during execution. A random sampling of SPLASH-2 benchmarks were used as tasks.

	System Utilization											
	$\rho = 0.1$			$\rho = 0.4$			$\rho = 0.6$			$\rho = 0.8$		
core number	ours (^oC)	[1] (^oC)	Δ^oC	ours (^oC)	[1] (^oC)	Δ^oC	ours (^oC)	[1] (^oC)	Δ^oC	ours (^oC)	[1] (^oC)	Δ^oC
16	73.2	74.1	0.9	81.5	86.4	4.9	86.8	88.5	1.7	93.8	93.5	-0.3
25	74.6	74.3	-0.3	83.1	86.7	3.6	87.5	90.8	3.3	95.1	96.0	0.9
36	72.8	73.2	0.4	83.8	88.3	4.5	87.3	91.1	3.8	96.4	96.1	-0.3
49	70.5	71.4	0.9	78.0	86.1	8.1	86.2	90.8	4.6	93.1	93.8	0.7



(a) 16 cores (b) 25 cores (c) 36 cores

Figure 6: Peak temperature comparison over time for $\rho = 0.4$ for reinforcement learning (RL) and adaptive random (AR)

Table 4: Time cost For RL task allocation (ms)

Temperature Center Set	Sensor Numbers 5	6	7	8	9
{340, 350}	0.014	0.024	0.043	0.081	0.150
{335, 345, 355}	0.074	0.216	0.635	1.927	6.210

6. CONCLUSIONS

A reinforcement learning based task allocation strategy is presented to address localized overheating in many-core systems. RL quality metrics are used to make optimized allocation decisions based on the accuracy of previous allocation choices. The experiments show that the proposed technique is capable of capturing the complex on-chip thermal environment induced by dynamic workload distribution. Our results show that the proposed technique is fast (scheduling performed in <1 ms) and can efficiently reduce peak temperature by 4.3^oC on average in moderately-loaded many-core processors for a collection of SPLASH-2 benchmarks.

7. REFERENCES

[1] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. DATE*, Mar. 2007, pp. 1659–1664.

[2] C. H. Yu, C.-L. Lung, Y.-L. Ho, R.-S. Hsu, D.-M. Kwai, and S.-C. Chang, "Thermal-aware on-line scheduler for 3-D many-core processor throughput optimization," *IEEE TCAD*, vol. 33, no. 5, May 2014.

[3] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *IEEE TVLSI*, vol. 16, no. 9, pp. 1127–1140, Sep. 2008.

[4] I. Yeo and E. J. Kim, "Temperature-aware scheduler based on thermal behavior grouping in multicore systems," in *Proc. DATE*, May 2009, pp. 946–951.

[5] V. Hanumaiah, S. Vrudhula, and K. Chatha, "Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors," *IEEE TCAD*, vol. 30, no. 11, pp. 1677–1690, Nov. 2011.

[6] W. Hung et al., "Thermal-aware IP virtualization and placement for networks-on-chip architecture," in *Proc ICCD*, Oct. 2004, pp. 430–437.

[7] Z. Qian and C.-Y. Tsui, "A thermal-aware application specific routing algorithm for network-on-chip design," in *Proc. ASP-DAC*, 2011.

[8] Y. Ge and Q. Qiu, "Dynamic thermal management for multimedia applications using machine learning," in *Proc. DAC*, June 2011, pp. 95–100.

[9] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Proc. CODES+ISSS*, Oct 2011.

[10] G.-Y. Pan, J.-Y. Jou, and B.-C. Lai, "Scalable power management using multilevel reinforcement learning for multiprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 4, pp. 33:1–33:23, Aug. 2014.

[11] H. Chen and etc, "Spatially-aware optimization of energy consumption in consolidated data center systems," in *ASME 2011 Pac. Rim Tech. Conf. and Exhibition on Packaging and Integration of Elec. and Phot. Sys.*

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

[13] E. Rotem et al., "Power management architecture of the 2nd generation Intel core microarchitecture, formerly codenamed Sandy Bridge," in *Hot Chips*, August 2011.

[14] L. Baird and A. W. Moore, "Gradient descent for general reinforcement learning," *Adv. in Neural Information Proc. Sys.*, pp. 968–974, 1999.

[15] L. Sheng et al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. IEEE/ACM Micro*, 2009.

[16] C. Sun et al., "DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. IEEE/ACM Int'l Symp. on NoC*, 2012, pp. 201–210.

[17] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *Int'l Conf. for High Perf. Comput., Network., Stor. and Analysis*, 2011.