NPU-Accelerated Imitation Learning for Thermal Optimization of QoS-Constrained Heterogeneous Multi-Cores

Martin Rapp, Heba Khdr, Nikita Krohmer, and Jörg Henkel

Abstract—Application migration and dynamic voltage and frequency scaling (DVFS) are indispensable means for fully exploiting the available potential in thermal optimization of a heterogeneous clustered multi-core processor under user-defined quality of service (QoS) targets. However, selecting the core to execute each application and the voltage/frequency (V/f) levels of each cluster is a complex problem because 1) the diverse characteristics and QoS targets of applications require different optimizations, and 2) per-cluster DVFS requires a global optimization considering all running applications. State-of-the-art resource management techniques for power or temperature minimization either rely on measurements that are often not available (such as power) or fail to consider all the dimensions of the problem (e.g., by using simplified analytical models). Imitation learning (IL) enables to use the optimality of an oracle policy, yet at low run-time overhead, by training a model from oracle demonstrations. We are the first to employ IL for temperature minimization under QoS targets. We tackle the complexity by training a neural network (NN) and accelerate the NN inference using a neural processing unit (NPU). While such NN accelerators are becoming increasingly widespread on end devices, they are so far only used to accelerate user applications. In contrast, we use an existing accelerator on a real platform to accelerate NN-based resource management. Our evaluation on a *HiKey 970* board with an *Arm big.LITTLE* CPU and an NPU shows significant temperature reductions at a negligible run-time overhead, with unseen applications and different cooling than used for training.

Index Terms—Machine learning, Imitation learning, Neural networks, AI accelerators, Thermal management, Quality of service, Processor scheduling, Task migration

✦

1 INTRODUCTION

Eᴌᴇᴠᴀᴛᴇᴅ on-chip temperature accelerates aging mechanisms in processors, and thereby degrades the system reliability [2], [3]. Moreover, in mobile devices, it may adversely affect the user experience since it leads to an increased skin temperature [4]. That makes temperature minimization of paramount importance. The two main knobs to reduce the temperature are application migration, to dynamically change the mapping of applications to cores, and DVFS. Using these knobs without considering the application characteristics misses significant optimization opportunities and may degrade the QoS of the applications, thereby also degrading the user experience [5]. The reason is that the impact on performance and power when migrating an application between clusters differs from one application to another [6]. Similarly, the sensitivities of performance and power to DVFS also vary. Hence, the possibilities of QoS-constrained thermal optimization vary between applications as the following motivational example demonstrates.

1.1 Motivational Example

In Scenario 1 in Fig. 1, we execute one application, *adi* or *seidel-2d* from the *Polybench* [7] suite, on an Arm big.LITTLE

- *M. Rapp, H. Khdr, and J. Henkel are with the Chair for Embedded Systems, Department of Computer Science, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany.*
  E-mail: martin.rapp@kit.edu, heba.khdr@kit.edu, henkel@kit.edu
- *N. Krohmer was with the Chair for Embedded Systems, Department of Computer Science, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany. E-mail: nikita-krohmer@web.de*
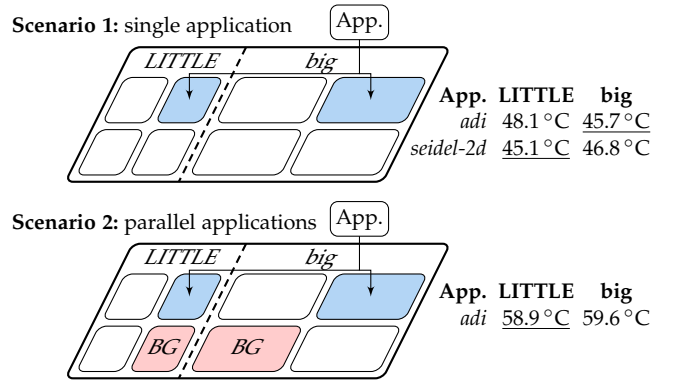- *A subset of this work has been first presented in DATE'22 [1].*

Fig. 1. On Arm big.LITTLE, the optimal mapping that minimizes the temperature under QoS targets varies between applications, and with other parallel applications (BG). The clusters are operated at the lowest V/f levels that satisfy all QoS targets.

CPU. The QoS target is selected as 30 % of the performance, measured in instructions per second (IPS), that is reached at the highest V/f level on the big cluster. The clusters are operated at the lowest V/f level that satisfies the QoS target. Intuitively, executing the applications on the LITTLE cluster should minimize the temperature. However, this is not always the case. For *adi*, mapping it to the big cluster instead minimizes the temperature. The reason is that *adi* requires 1.8 GHz on the LITTLE cluster to reach its QoS target, but only 0.7 GHz on the big cluster. In contrast, *seidel-2d* reaches its QoS target already at 1.2 GHz on the

LITTLE cluster, and requires 1.0 GHz on the big cluster, resulting in a similar temperature on both clusters, with a small advantage of the LITTLE cluster. The reason for the different V/f level requirements at different clusters is that the applications benefit differently from the out-of-order execution and larger caches on the big cluster. Consequently, such different application characteristics render different mappings optimal. *Optimal thermal management needs to consider application characteristics and QoS targets.*

Scenario 2 studies *adi* with the same QoS target as in Scenario 1 but now, additional background applications with high QoS targets run on both clusters. Intuitively, as in Scenario 1, mapping *adi* to the big cluster should still minimize the temperature. However, the background applications require to operate both clusters at the peak V/f level to reach their QoS targets. Since our platform has per-cluster DVFS, *adi* is also executed at the peak V/f level. In this case, mapping *adi* to the LITTLE or big cluster has almost the same temperature, unlike what has been observed in Scenario 1. Hence, per-cluster DVFS affects the optimal mapping when several applications run in parallel. *Optimal thermal management needs to perform global optimization considering the characteristics of all running applications.*

1.2 Challenges and Contributions

There are several challenges in temperature minimization on heterogeneous multi-core processors under QoS targets. Firstly, there is high complexity in all involved aspects of the platform. For instance, the power and performance of applications depend on the instruction sequence, CPU microarchitecture, memory architecture, and V/f level, while temperature depends on the power density, floorplan, and cooling. Secondly, the workload, i.e., the executed applications and their arrival times, is commonly not known at design time. Therefore, the management policy must not be specific to selected applications but achieve good management for any workload. Thirdly, per-cluster DVFS forces all applications on the same cluster to run at the same V/f level, requiring global optimization. Finally, there is limited access to measurements. For instance, most platforms, such as the one studied in this work, have no power sensors and only few temperature sensors.

Many works perform optimization with models for individual aspects such as power, performance, or temperature. These models can be built analytically [8] or by machine learning (ML) [9], [10]. However, building such models requires fine-grained access to internal measurements of processor-internal properties like power, which may not be available. To solve this, end-to-end learning of management decisions based on the available measurements can be employed. The two main methods to achieve this are reinforcement learning (RL) and IL. In both cases, NN learning can be used to cope with the high complexity [11].

RL suffers from several problems. It requires to combine objective and constraints in a single scalar reward, which does not reflect their different properties and may lead to suboptimal actions (reward hacking [12]). Moreover, RL trains at run time. This is computationally expensive, preventing a low-overhead implementation, and may result in instability such as catastrophic forgetting, leading to suboptimal management decisions. However, run-time thermal minimization while satisfying QoS targets requires a lightweight, yet near-optimal optimization to improve user experience, and a stable policy to avoid abrupt QoS violations and jumps in the temperature. *IL is the only method that provides all of these capabilities.* In particular, it enables using the optimality of an oracle policy, which explicitly considers objectives and constraints, yet at low run-time overhead, by design-time training of a model from oracle demonstrations. Design-time training until convergence also provides stability. However, since IL does not perform run-time retraining, the model must be trained such that it is capable to cope with the different scenarios that may happen at run time. This includes for instance, different workloads, or different cooling capabilities.

Motivated by the advantages of IL, researchers have started to apply IL in resource management [13], [14], [15], [16], but they all target power or energy optimization. This significantly differs from temperature optimization due to spatial (heat transfer) and temporal (heat capacity) effects that do not exist in power/energy. We are the first to employ IL for temperature optimization.

To accelerate ML-based resource management, few works have proposed their own specific ML accelerators [17], [18]. However, they incur additional area overhead to the used platform and are only applicable to platforms that feature this specific accelerator. Recently, generic NN accelerators, e.g., NPUs or DSPs, became common in end devices such as smartphones [19]. These accelerators are intended to increase the performance and energy-efficiency of user applications that perform NN inference. Despite their increasing spread and benefits, these existing accelerators have never been used to speed up NN-based resource management, and we are the first to do that.

We make the following novel contributions in this work:

- We design, train, and employ NN-based IL for temperature optimization under QoS targets, as it enables near-optimal decisions at low run-time overhead. Our solution, *TOP-IL*, employs application migration and DVFS on heterogeneous multi-cores.
- We accelerate *TOP-IL* using an existing generic NN accelerator (NPU) on a real platform.
- We develop RL-based thermal optimization and show that IL outperforms RL in terms of achieving the target objective and run-time stability.
- We demonstrate that the learned policy generalizes to unseen workloads and different cooling settings than what is used during training.

2 RELATED WORK

The state-of-the-practice Android/Linux resource management [27] performs application mapping and migration (scheduling), and DVFS. Most schedulers are designed for homogeneous multi-core processors. However, Global Task Scheduling (GTS) aims at increasing the energy efficiency of heterogeneous processors by migrating mostly-idle applications to the LITTLE cluster. Android/Linux performs DVFS with different governors, such as *powersave* for power minimization or *ondemand* for a trade-off between power and performance. However, these techniques do not consider

TABLE 1
Overview of related work

| Technique | Method | Goal | Actions | | Optimization | | Per-clust. | Het. | Unkn. | Multi- | Lim. Power |
			Map./Mig.	DVFS	Temp.	QoS	DVFS	Cores	Apps.	Prog.	Sensors
ondemand/ powersave	Rules	max perf./ min P	✓	✓	×	×	✓	✓	✓	✓	✓
[20]	RL	max perf st. P	×	✓	×	×	×	(✓)[1]	✓	✓	×
[21]	RL	min E st. R	×	✓	×	×	×	×	✓	✓	×
[18]	RL	min EDP	×	✓	×	×	✓	✓	✓	(✓)[1]	✓
[22]	RL	max R st. QoS	✓	✓	×	✓	(✓)[1]	(✓)[1]	✓	×	✓
[23]	RL	min P st. QoS	✓	✓	×	✓	×	(✓)[1]	✓	✓	×
[24]	RL	min T	✓	×	✓	×	(✓)[1]	(✓)[1]	✓	✓	✓
[25]	RL	min T	✓	×	✓	×	(✓)[1]	(✓)[1]	✓	✓	✓
[26]	RL	min T st. QoS	✓	✓	✓	✓	(✓)[1]	(✓)[1]	×	×	✓
[13]	IL	min E	(✓)[2]	✓	×	×	✓	✓	✓	✓	×
[14]	IL	min E st. QoS	×	✓	×	✓	×	(✓)[1]	×	✓	×
[15]	IL	min E st. QoS	(✓)[2]	✓	×	✓	✓	✓	✓	✓	×
[16]	IL	min E st. QoS	(✓)[2]	✓	×	✓	✓	✓	✓	✓	×
TOP-IL (our)	IL	min. T st. QoS	✓	✓	✓	✓	✓	✓	✓	✓	✓

T: temperature, P: power, E: energy, R: reliability.   [1] Not studied, likely applicable with minor changes.   [2] Controls the number of active cores.

application characteristics nor their QoS targets, and only indirectly affect the temperature (via power or energy).

ML provides powerful algorithms for system-level optimization [28]. Supervised learning can be used to train models that predict system properties like performance or power [29]. Such models enable rule-based power/thermal management to predict the impact of a decision, and thereby achieve proactive management [9]. However, model training requires access to measurements like per-core power, which are often not available in real-world processors [30].

Several works have employed RL for power/thermal optimization [31]. The works in [18], [20], [21] use RL for power management via DVFS. However, they neither consider temperature nor QoS. The work in [22] optimizes the reliability under QoS using both migration and DVFS. While reliability depends on the temperature, the two are not interchangeable. For instance, a part of the reward function in [22] minimizes thermal cycling, which is unrelated to the absolute temperature. In addition, the work does not cope with several applications running in parallel. In [23], RL is employed at the core level. A high-level coordinator translates the system goal, i.e., minimizing power, into core-level target IPS. Then, core-level RL agents select the V/f level to manage the core IPS accordingly. However, this work also does not consider temperature, is not applicable to per-cluster DVFS, and requires run-time power measurements. Several works employ RL for temperature optimization. The work in [24] performs migration for temperature minimization based on per-core temperature measurements. In [25], the temperature is minimized via mapping applications at arrival time. However, these works do not consider QoS. Finally, [26] considers both temperature and QoS. It uses application mapping and DVFS. However, this work analyzes intermediate compiler-level representations of applications, and, hence, is only applicable to known applications. In addition, it does not cope with several applications running in parallel. Table 1 summarizes these works.

Several recent works employ IL for system-level optimization. The work in [13] trains a model to predict the optimal number of active cores and per-cluster V/f levels to minimize the energy. In [14], an IL technique is proposed for DVFS to minimize the energy under a QoS targets. They train a separate policy per application, and, hence, cannot cope with unknown applications. The work in [15] uses IL to select the types, number, and V/f levels of active cores, for several optimization goals, e.g., minimize the energy under a QoS targets. Finally, a hierarchical IL technique is proposed in [16] to select the number of active cores and the per-cluster V/f level to maximize the energy efficiency of a heterogeneous multi-core processor under QoS targets. These works divide the application execution into phases and record performance counters, performance, and power for each phase at different configurations (number of active cores, V/f levels, etc.). Oracle demonstrations are created by finding the optimal sequence of configurations per phase. This only works because power, performance, and energy of a phase depend only on the used configuration in this phase. However, this does not apply to temperature, which is subject to both spatial (heat transfer) and temporal (heat capacity) effects that do not exist in power/energy. Consequently, the temperature during a phase additionally depends on all configurations of all previous phases. This would require an exponential number of traces, which is infeasible. In addition, the power sensors required for the oracle are often not available in real-world processors. Table 1 also summarizes these works. IL has not yet been employed for thermal optimization despite its unique capabilities to combine the optimality of an oracle policy with a low run-time overhead. We are the first to do that.

In summary, none of these works targets temperature minimization under QoS targets, and considers heterogeneous cores with per-cluster DVFS running parallel applications.

3 PROBLEM FORMULATION

We target a heterogeneous multi-core processor with per-cluster DVFS, where $\mathcal{F}_x$ is the list of frequencies of cluster x and f_x is its current V/f level. There are two clusters in our platform, LITTLE and big, i.e., $x \in \{l, b\}$, but our solution is compatible with any number of clusters. The processor

executes parallel applications, each with its own QoS target Q_k and current QoS q_k, which are expressed in terms of the IPS. We target an open system, where a priori unknown applications arrive at a priori unknown times. Our solution does not rely on run-time power measurement, as they are often not available on real-world processors [30].

Objective	minimize the on-chip temperature
Constraint	maintain QoS of applications (IPS)
Knobs	app.-to-core mapping (migration), per-cluster DVFS.

We split the problem into two parts: 1) application-to-core mapping (via application migration), and 2) per-cluster DVFS. Decisions on application migration are made with NN-based IL, while the DVFS is implemented in a simple control loop. While it would be intuitive to train a single NN for both migration and DVFS, performing only migration with the model reduces its complexity (create training data, topology, inference overhead). Nevertheless, we consider V/f level information as input for migration decisions to achieve near-optimal decisions. We accelerate the run-time inference with an NPU. The design-time training and run-time management are described in Sections 4 and 5.1, respectively. Section 5.2 describes the DVFS control loop.

4 IL-BASED APPLICATION MIGRATION

Employing IL requires to select features, create oracle demonstrations, and train the model that is used at run time.

4.1 Feature Selection

The features need to accurately describe the platform state to be able to make near-optimal migration decisions, and need to be observable at run time. The optimal mapping of an *application of interest (AoI)* depends on a) its characteristics, which affect its power and performance on different clusters, b) its QoS target, which determines the suitable clusters and required V/f levels, and c) other (background) applications, which determine the available cores, the required V/f levels per cluster to satisfy QoS targets of the background applications, and affect the temperature distribution.

The selected features (Table 2) cover all three aspects (a-c). The AoI characteristics (a) comprise the current QoS and the number of L2D accesses per second. The latter indicates the memory-/compute-intensiveness of the AoI. We use the Linux *perf* API to read performance counters (IPS and L2D accesses). The current mapping of the AoI provides information about the source core and cluster, thereby providing context to the performance counter readings. It is represented as one-hot encoding of all cores. The QoS target (b) is represented in terms of IPS. The background (c) is represented by the core utilizations, as well as by the estimated V/f level change if the AoI would not be executed (for each cluster). The latter indicates potential temperature savings if the AoI is migrated to another cluster. This is calculated by first estimating the minimum V/f level $\tilde{f}_{k,min}$ for each running application k that is required to satisfy its QoS target Q_k. During training data generation at design time, $\tilde{f}_{k,min}$ can be determined from the execution traces. At run time, no traces at other V/f levels are available,

TABLE 2
The Selected Features for IL-based Migration (per Application)

Feature	Count	Feature	Count
AoI QoS (a)	1	AoI QoS target (b)	1
AoI L2D accesses (a)	1	$\tilde{f}_{x \setminus AoI}/f_x$ (c)	2
AoI curr. mapping (a)	8	Core utilizations (c)	8

and linear scaling from the current V/f level $f_{x(k)}$ of its cluster $x(k)$ is performed instead:

$$\tilde{f}_{k,min} = \min\{f \in \mathcal{F}_{x(k)} : q_k \cdot f/f_{x(k)} \geq Q_k\} \quad (1)$$

This estimate is calculated at run time based on the current QoS q_k in the current execution phase, i.e., $\tilde{f}_{k,min}$ *does not need to be known at design time and may change over time.* Finally, the required V/f level without the AoI is determined per cluster x as the maximum among all other applications:

$$\tilde{f}_{x \setminus AoI} = \max\{\tilde{f}_{k,min} : \text{app. } k \text{ mapped to } x \wedge k \neq AoI\} \quad (2)$$

4.2 Oracle Demonstrations (Training Data)

The training data need to indicate the optimal migration w.r.t. QoS and temperature for a variety of scenarios. To this end, we collect measurements of temperature and performance counters (traces) of benchmark applications in various scenarios and extract training data from the traces.

Collect Traces: The process to collect traces is depicted in the upper part of Fig. 2. Since this is the most time-consuming part of training, redundant executions must be avoided. The straightforward approach to collect traces would be to select a scenario, i.e., a combination of AoI, its QoS target, and background, and execute it once per mapping of the AoI to each free core. However, this creates redundant executions. The reason is that with per-cluster DVFS, only the application with the highest QoS target, i.e., highest required V/f level, determines the V/f level of the cluster. As a result, scenarios that differ only in the QoS, may result in the same selected V/f levels.

We avoid redundancy by obtaining traces for different combinations of per-cluster V/f levels and afterwards select different QoS targets to create training data. This optimization requires a constant QoS of the benchmarks that are used to create the training data, i.e., no execution phases. *As the evaluation demonstrates, our model also generalizes to applications with execution phases.* To further accelerate collecting traces, we stop traces after 10^{10} instructions of the AoI, which is large enough to observe significant differences in the temperature between traces but still reduces the time to collect a trace, and obtain traces for a reduced set of V/f levels. However, *TOP-IL* supports applications with more executed instructions. We execute the background of each scenario for 2 min before starting the AoI to ensure consistent initial temperature. We randomize the order of executions to avoid any remaining systematic error. We use active cooling with a fan because it prevents triggering dynamic thermal management (DTM), which would throttle the V/f levels unpredictably, polluting the training data. *We show in our evaluation that the trained NN also can be used without retraining for different cooling, i.e., without a fan.*

Figs. 3a and 3b present an illustrative excerpt of the collected traces (performance of the AoI and temperature) for a single selection of background applications and AoI (*seidel-2d*). In this example, only the two cores 3 and 6 are free. The other cores are running background applications.

Extract Training Data: The lower part of Fig. 2 shows the steps to extract training data from the collected traces: select many QoS targets, find the corresponding traces, and create training examples. We first select a combination of background and AoI from the traces. Then, we sweep the values of the QoS target Q_{AoI} of the AoI, and the required V/f levels of the background $\tilde{f}_{l\backslash AoI}, \tilde{f}_{b\backslash AoI}$. Next, we find the corresponding trace when mapping the AoI on core j with the selected parameters. The V/f levels f_l, f_b of this trace are the lowest levels to satisfy $Q_{AoI}, \tilde{f}_{l\backslash AoI}$, and $\tilde{f}_{b\backslash AoI}$:

$$f_l, f_b = \arg\min_{f'_l, f'_b}(f'_l \geq \tilde{f}_{l\backslash AoI} \wedge f'_b \geq \tilde{f}_{b\backslash AoI} \wedge q_{AoI}(f'_l, f'_b) \geq Q_{AoI}) \quad (3)$$

The peak temperature for each mapping of the AoI to each free core j is determined from these traces. We observe that in many cases, several mappings result in a very close temperature (e.g., mappings to different LITTLE cores). In our experiments, there is on average one additional mapping that is within $1\,°C$ of the temperature obtained with the optimal mapping. Therefore, we use a soft label $l_j \in [0, 1]$, indicating the quality of mapping the AoI to core j:

$$l_j = \begin{cases} 0 & \text{core } j \text{ occ. by background} \\ -1 & \text{core } j \text{ cannot meet } Q_{AoI} \quad (4) \\ e^{-\alpha(T_j - \min_{j'} T_{j'})} & \text{otherwise} \end{cases}$$

Cores that are used by the background get $l_j=0$. Mappings that violate the QoS target at the highest V/f level get $l_j = -1$. The mapping with the lowest temperature has $l_j=1$. For other mappings, the higher the temperature is compared to the optimum, the closer l_j gets to 0. The parameter α determines a trade-off between tolerating slightly higher temperatures and susceptibility to temperature measurement noise. We empirically set $\alpha=1$. Fig. 3c lists some illustrative examples. For instance, when selecting $Q_{AoI} = 400 \cdot 10^6$ IPS, $\tilde{f}_{l\backslash AoI} = 1.4\,GHz$, and $\tilde{f}_{b\backslash AoI} = 0.7\,GHz$ (Line I), the minimum frequencies of LITTLE/big to satisfy all QoS targets are $1.8\,GHz/0.7\,GHz$ and $1.4\,GHz/1.2\,GHz$ for a mapping of the AoI to cores 3 and 6, respectively. This results in respective temperatures of $42.5\,°C$ and $46.6\,°C$, i.e., a mapping to core 3 is cooler. Therefore, the respective labels for cores 3 and 6 are 1 and 0.02. Fig. 3c also lists examples where the two cores result in similar temperature, where core 6 is beneficial, and where core 3 cannot meet the QoS target, even at the highest V/f levels (Line II).

After creating the label, the features that describe an execution of the AoI with the selected QoS and background are determined from the traces according to Section 4.1. One training example is created for each free core, where the AoI could be executed when determining the optimal migration, i.e., each source of a migration. This is illustrated in Fig. 3d with a few examples. By creating one training example for *every* free core for each selection of $Q_{AoI}, \tilde{f}_{l\backslash AoI}$, and $\tilde{f}_{b\backslash AoI}$, the process of training data generation is already exhaustive because the policy is trained to recover from each potential mapping of the AoI. This is the reason why we do not need
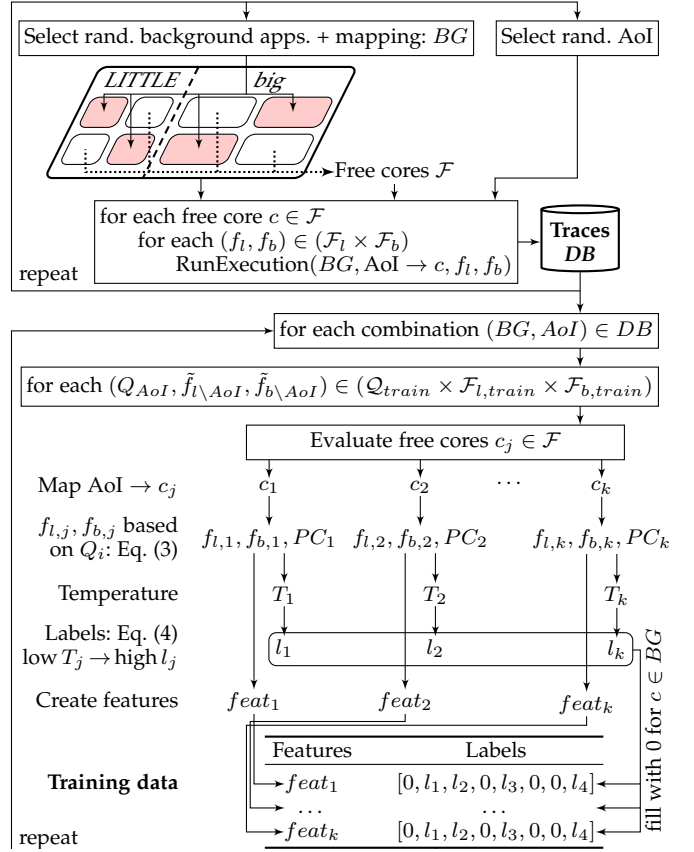

Fig. 2. Design-time training data generation for IL-based migration.

to employ algorithms like *DAgger* [32], which initially only train the policy on the optimal sequence of management decisions, and only gradually add training data to recover from suboptimal decisions to increase the robustness of the model. We create 19,831 training examples from 100 combinations of AoI and background.

4.3 IL Model Creation and Training

We build a fully-connected NN model and decide its topology (number of layers and neurons) by neural architecture search (NAS). Fig. 4 shows the result of the grid search to determine the depth and width of the NN. The best topology uses 4 hidden layers with 64 neurons, each. The hidden layers use ReLU activation, the output layer with 8 neurons does not use an activation function. We use *Adam* optimizer with momentum. The exponentially decaying learning rate is set at $0.01 \cdot 0.95^{(epoch)}$. We use mean squared error (MSE) loss and early stopping with a patience of 20 epochs. Three models are trained with different random seed to demonstrate that the training is robust to the weight initialization, as will be shown in Section 7.

5 RUN-TIME TEMPERATURE / QOS MANAGEMENT

The run-time part of *TOP-IL* (Fig. 5) integrates IL-based application migration with a per-cluster DVFS control loop.

5.1 Application Migration with NPU-Accelerated IL

If K applications run in parallel, each should be migrated to its optimal core w.r.t. temperature and QoS. However,

(a) Trace results* (running AoI on core 3 on the LITTLE cluster)

Performance q		f_b		
		0.7 GHz	1.2 GHz	1.5 GHz	...
	0.5 GHz	137 MIPS	140 MIPS	139 MIPS	...
f_l	1.4 GHz	366 MIPS	363 MIPS	373 MIPS	...
	1.8 GHz	471 MIPS I	478 MIPS	479 MIPS	...

Temperature T		f_b		
		0.7 GHz	1.2 GHz	1.5 GHz	...
	0.5 GHz	35.8 °C	42.3 °C	50.7 °C	...
f_l	1.4 GHz	40.5 °C	46.2 °C	53.7 °C	...
	1.8 GHz	42.5 °C I	49.6 °C	56.1 °C	...

(b) Trace results* (running AoI on core 6 on the big cluster)

Performance q		f_b		
		0.7 GHz	1.2 GHz	1.5 GHz	...
	0.5 GHz	256 MIPS	455 MIPS	563 MIPS II	...
f_l	1.4 GHz	255 MIPS	455 MIPS I	563 MIPS	...
	1.8 GHz	256 MIPS	454 MIPS	562 MIPS	...

Temperature T		f_b		
		0.7 GHz	1.2 GHz	1.5 GHz	...
	0.5 GHz	38.0 °C	46.2 °C	52.2 °C II	...
f_l	1.4 GHz	38.4 °C	46.6 °C I	56.5 °C	...
	1.8 GHz	39.5 °C	48.8 °C	57.0 °C	...

(c) Examples for calculating the labels

Q_{AoI}	$\tilde{f}_{l\backslash AoI}$	$\tilde{f}_{b\backslash AoI}$	Trace results (AoI on core 3)			Trace results (AoI on core 6)			Labels	
			$f_{l,3}$	$f_{b,3}$	T_3	$f_{l,6}$	$f_{b,6}$	T_6	$l_0 \ldots l_7$	
400 MIPS	1.4 GHz	0.7 GHz	1.8 GHz	0.7 GHz	42.5 °C	1.4 GHz	1.2 GHz	46.6 °C	0 0 0 1.00 0 0 0.02 0	I
200 MIPS	1.4 GHz	1.2 GHz	1.4 GHz	1.2 GHz	46.2 °C	1.4 GHz	1.2 GHz	46.6 °C	0 0 0 1.00 0 0 0.65 0	
400 MIPS	0.5 GHz	1.5 GHz	1.8 GHz	1.5 GHz	56.1 °C	0.5 GHz	1.5 GHz	52.2 °C	0 0 0 0.02 0 0 1.00 0	
500 MIPS	0.5 GHz	0.7 GHz	–	0.7 GHz	–	0.5 GHz	1.5 GHz	52.2 °C	0 0 0 −1 0 0 1.00 0	II

(d) Training data examples

f_l	f_b	q_{AoI}	Q_{AoI}	AoI curr. map.	Core utils.	$\tilde{f}_{l\backslash AoI}/f_l$	$\tilde{f}_{b\backslash AoI}/f_b$	Labels $l_0 \ldots l_7$	
1.8 GHz	0.7 GHz	471 MIPS	400 MIPS	0 0 0 1 0 0 0 0	1 1 1 0 1 1 0 1	0.76	1.00	0 0 0 1.00 0 0 0.02 0	I
1.4 GHz	1.2 GHz	455 MIPS	400 MIPS	0 0 0 0 0 0 1 0	1 1 1 0 1 1 0 1	1.00	0.56	0 0 0 1.00 0 0 0.02 0	
1.8 GHz	0.7 GHz	471 MIPS	500 MIPS	0 0 0 1 0 0 0 0	1 1 1 0 1 1 0 1	0.28	1.00	0 0 0 −1 0 0 1.00 0	II
0.5 GHz	1.5 GHz	563 MIPS	500 MIPS	0 0 0 0 0 0 1 0	1 1 1 0 1 1 0 1	1.00	0.46	0 0 0 −1 0 0 1.00 0	

Fig. 3. Illustrative example for training data generation. Only cores 3 and 6 are available for the AoI. (a) and (b) show the trace results (AoI performance and temperature) for the two free cores and several combinations of V/f levels f_l and f_b. (c) demonstrates the label calculation for a given AoI QoS target Q_{AoI}, and minimum required V/f level to maintain the QoS of the background $(\tilde{f}_{l\backslash AoI}, \tilde{f}_{b\backslash AoI})$. For each mapping, the minimum V/f levels that satisfy all QoS targets are determined to obtain the temperature. Labels are calculated by Eq. (4). (d) lists some training examples. (I) highlights an example, in which a mapping to the LITTLE cluster is optimal. (II) highlights an example, in which the LITTLE can not reach the QoS target even at the highest V/f level. *The number of L2 cache accesses has been omitted from the traces and features for brevity.
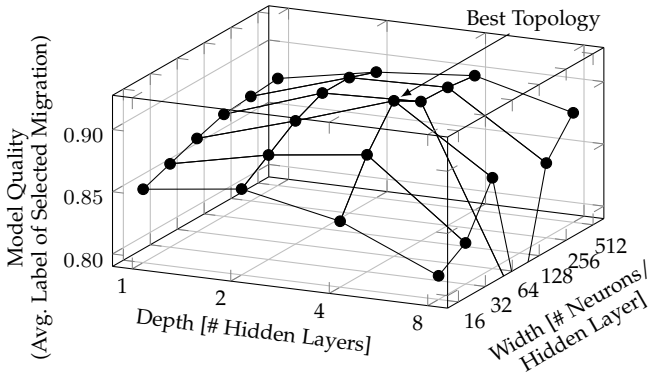

Fig. 4. The best topology uses 4 hidden layers with 64 neurons, each.

Fig. 5. Illustration of *TOP-IL* at run time. Application migration uses the NPU to accelerate predicting the best migration per each application.

migrating several applications at once results in a high number of potential combinations, i.e., large action space, and the impact of several migrations at once would be difficult to predict. We solve this by migrating only one application at a time, but we find in each iteration the best migration among all possible migrations of all applications. Our NN model has been trained for one AoI, which is migrated, and several other background applications. We perform parallel inference, where each application is used as the AoI once. The inference output is a matrix, where
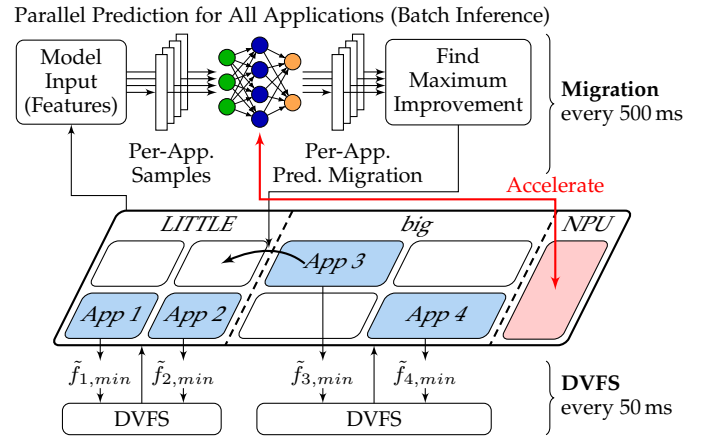
each entry $\tilde{l}_{k,c}$ is the rating of mapping application k to core c. The best migration maximizes the improvement in the rating compared to the current mapping $c(k)$:

$$\hat{k}, \hat{c} = \arg\max_{k',c'} \left(l_{k',c'} - l_{k',c(k')} \right) \tag{5}$$

The result of this optimization is to migrate application $\hat{k}$ to core $\hat{c}$. The migration policy is executed each 500 ms. This
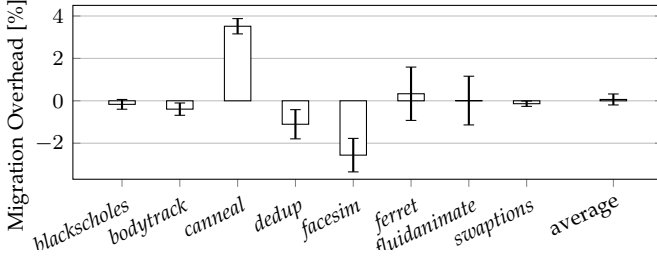
Fig. 6. The performance impact of application migration is negligible.

is fast enough to adapt to changing workload phases of the applications, which run for several minutes, but still allows to maintain a reasonable overhead.

To further reduce the overhead of the NN inference, we employ the already existing NPU of the *HiKey 970* board. The available parallelism in the NPU allows performing parallel inference for all applications simultaneously in a single batch. The NPU is accessible via the *HiAI DDK*, which originally is designed to speed up user apps. We develop a C++ binary that runs in user space, uses the Linux *perf* API and the /proc filesystem to read performance counters and information about running applications, employs the NPU for inference via the *HiAI DDK* (non-blocking call), and uses the Linux *affinity* feature for migration.

Since, we perform migration each 500 ms, the migration overhead, e.g., due to cold caches, is negligible. We perform experiments to quantify the worst-case overhead, i.e., periodically migrating an application between the big and LITTLE cluster in each migration epoch. The migration overhead m is calculated by:

$$ m = \frac{1/2 \cdot (1/t_{\text{big}} + 1/t_{\text{LITTLE}})}{1/t_{\text{migrate}}} - 1 \qquad (6) $$

The numerator represents the average performance of the big and LITTLE clusters, while the denominator represents the measured performance with periodic migration. We repeat each experiment three times and plot the average and standard deviation of the migration overhead of several applications in Fig. 6. The overhead differs between applications because of their different memory and cache intensity. For some applications, (*dedup*, *facesim*), we observe a negative overhead, which we interpret as follows. If an application has different execution phases that benefit differently from the features of big cluster, potential correlation between the migration epoch and the execution phases improves the performance of these applications, and thereby results in a negative overhead. The maximum worst-case migration overhead is less than 4%, while the average worst-case migration overhead is 0.1%, which is negligible.

5.2 Control Loop for Per-Cluster DVFS

The IL-based migration is integrated with a DVFS control loop to select the per-cluster V/f-levels. The control loop utilizes the estimated $\tilde{f}_{k,min}$ per application k, as defined in Eq. (1). It then determines the minimum required V/f level per cluster x to satisfy the QoS target of all applications running on it:

$$ \tilde{f}_x = \max\{\tilde{f}_{k,min} : \text{application } k \text{ mapped to cluster } x\} \qquad (7) $$

Since the run-time estimates of $\tilde{f}_{k,min}$ are based on linear scaling, they are only accurate for small V/f level changes. Therefore, we adjust the current V/f level f_x by only one step towards $\tilde{f}_x$ and call this control loop more frequently than migration, i.e., every 50 ms. We skip two iterations, one when application migration is executed and one directly after a migration, to account for transient effects of cold caches that result in spurious QoS violations. Idle clusters are operated at the lowest V/f level. We use the Linux *userspace* governor to set per-cluster V/f levels.

The combination of IL-based application migration and DVFS control loop enables us to achieve temperature optimization under QoS targets, as evaluated in the Section 7.

6 RL-BASED APPLICATION MIGRATION

As discussed earlier, RL is another method for end-to-end learning and directly making management decisions, like IL. However, IL outperforms RL in terms of stability of the learned policy. To demonstrate this in a quantitative comparison, there is a need to implement an RL-based technique *Therm-RL* that has the same goal as our IL-based *TOP-IL*. Section 2 reviewed the state-of-the-art techniques that employ RL for application mapping/migration or DVFS. However, none of them targets the same goal as ours and considers heterogeneous cores with per-cluster DVFS running parallel applications. Therefore, this section presents an RL-based application migration policy, motivated by the state of the art, to serve as a baseline for the IL-based policy described in Section 4. To enable a fair comparison between RL and IL, we also perform only migration with RL and employ the same DVFS control loop described in the previous section.

TOP-IL achieved independence from the number of running applications by performing independent inference per each running application, denoted the AoI, to find the optimal migration. RL additionally requires to perform run-time training, which requires maintaining information about the previous state. Therefore, we instantiate one agent per application. This has the additional benefit of maintaining state and action spaces at a reasonable size, as will be discussed in the next section. The overall structure of *Therm-RL* is depicted in Fig. 7.

6.1 State, Action, and Reward

The state space used for the RL agent comprises the same features as also used for the IL model. In particular, these are the QoS, number of L2D accesses, and the current mapping of the AoI, as well as the frequencies and utilizations of the big and LITTLE clusters. All these features are quantized to maintain a Q-table with a reasonable size. For instance, the information about the QoS is represented by a binary signal indicating whether or not the QoS target is met.

The action space is selected the same as with our IL technique, which is also the same as in [24]. There is one action per core, indicating a migration to this core, i.e., in total 8 actions. The Q-table contains 2,304 entries, which is similar in size to what is reported in [18].

The reward function needs to combine the objective (temperature minimization) and constraint (QoS target) into
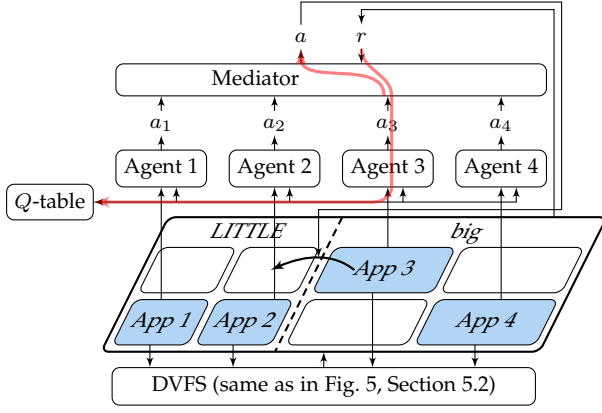
Fig. 7. RL-based migration instantiates one agent per application. All agents share the Q-table. A mediator selects the single executed action a from each per-agent action a_i (in this example from Agent 3). Only the selected agent updates the Q-table based on the reward.

a single scalar value. The objective is similar to [24], which only rewards a low temperature T: $r=80°\text{C}-T$. We extend it to penalize QoS violations:

$$r = \begin{cases} 80°\text{C} - T & \text{if } \forall i : q_i \geq Q_i \\ -200 & \text{otherwise (QoS violation)} \end{cases} \quad (8)$$

We have empirically tuned the negative reward of -200 in case of a QoS violation, in order to achieve a good trade-off between low temperature and low QoS violations.

6.2 Multi-Agent Learning for Parallel Applications

As discussed earlier, we instantiate one RL agent per application. Mediation between the agents is required to avoid 1) contradicting decisions by different agents, and 2) instability in the learning. Contradicting migration decisions could result if two agents decide to perform a migration at the same time to the same core. Such decisions should be not executed, because applications sharing a core would likely violate QoS targets. Moreover, even two migrations at the same time to different cores should be avoided, as simultaneous migrations might nullify the benefits of each other. Additionally, a change in temperature when performing two migrations at once can not be traced back to either of the two, causing instability in the learning.

We, therefore, implement a mediator between the agents, similar to [33]. The mediator selects the best action among the individual actions selected by each agent based on the highest Q-value, and executes it. After having executed the action, the reward obtained in the next control step should only be used to perform learning about this action, not about actions from other agents that have not been selected. Therefore, the mediator forwards the reward only to the agent selected in the previous step to perform learning. Fig. 7 illustrates the mediation process. All agents share a common Q-table to improve generalization to different applications, and to immediately start with a trained policy when a new application arrives to the system.

6.3 Training

We select the training parameters as in [24]. We use an ϵ-greedy policy with $\epsilon=0.1$, a discount factor $\gamma=0.8$, and

a learning rate $\alpha=0.05$. As the Q-table is initialized with constant values, a high-quality RL policy is only obtained after significant training. Therefore, the initial performance of an RL policy is not representative. We avoid this by first training a policy until convergence ($\sim$3 h) on a different random workload from what is used later in the evaluation. We then store the Q-table and load it at the beginning of each evaluation run. To reduce the impact of randomness on the policy performance, three policies are trained with different random seeds, like with the IL model.

7 EXPERIMENTAL EVALUATION

We perform experiments on a *HiKey970* [30] board. It employs a HiSilicon Kirin 970 smartphone SoC that implements the common Arm big.LITTLE architecture with four Arm Cortex-A53 and four Arm Cortex-A73 cores. It supports per-cluster DVFS with frequencies up to 1.84 GHz and 2.36 GHz, respectively. Furthermore, it comes with an NPU to accelerate NN inference. The board runs Android 8.0. We place the board in an A/C room to maintain a constant ambient temperature. The on-chip temperature is monitored with the on-board thermal sensor with a frequency of 20 Hz.

TOP-IL is compared with *Therm-RL* presented in Section 6, as well as with state-of-the-practice solutions, Linux GTS, paired with either *ondemand* or *powersave* governors. GTS assigns applications to a cluster depending on the computational requirements, i.e., mostly-idle and performance-hungry applications are migrated to the LITTLE and big cluster, respectively. *Ondemand* aims at providing a high performance but saving power when low performance is required. It achieves this by scaling the V/f-levels according to the CPU utilization, where V/f levels are upscaled if the utilization exceeds a fixed threshold, and downscaled if it falls below a second threshold. *Powersave* minimizes the power consumption by always operating at the lowest V/f levels, irrespective of the associated performance losses. These Linux policies are not aware of detailed application characteristics or QoS targets. *GTS/ondemand* is the default configuration that is shipped with Android 8.0 on *HiKey970*.

Generalization and Robustness: We demonstrate that *TOP-IL* and the employed NN model can cope with: 1) *Unseen applications* that have not been used for training. 2) *Different cooling*: We perform experiments also with passive cooling (without a fan) instead of the active cooling used for training data generation. 3) *Randomness in the training and at run time*: We train three models with different random seeds to demonstrate the robustness to weight initialization. We then repeat the experiments three times, where each repetition uses a different model, and report average and standard deviation of results. This demonstrates robustness to run-time variability due to workload fluctuations. In addition, we demonstrate 4) the *stability* of the learned policy.

7.1 Illustrative Example

We first present an illustrative example comparing the migration decisions of IL and RL. We study the same case as presented in the motivational example in Fig. 1, i.e., we run the two applications *adi* and *seidel-2d*. Fig. 8a shows the selected cluster (mapping) of *adi*. A mapping to the

(a) *adi* (*optimal mapping: big)

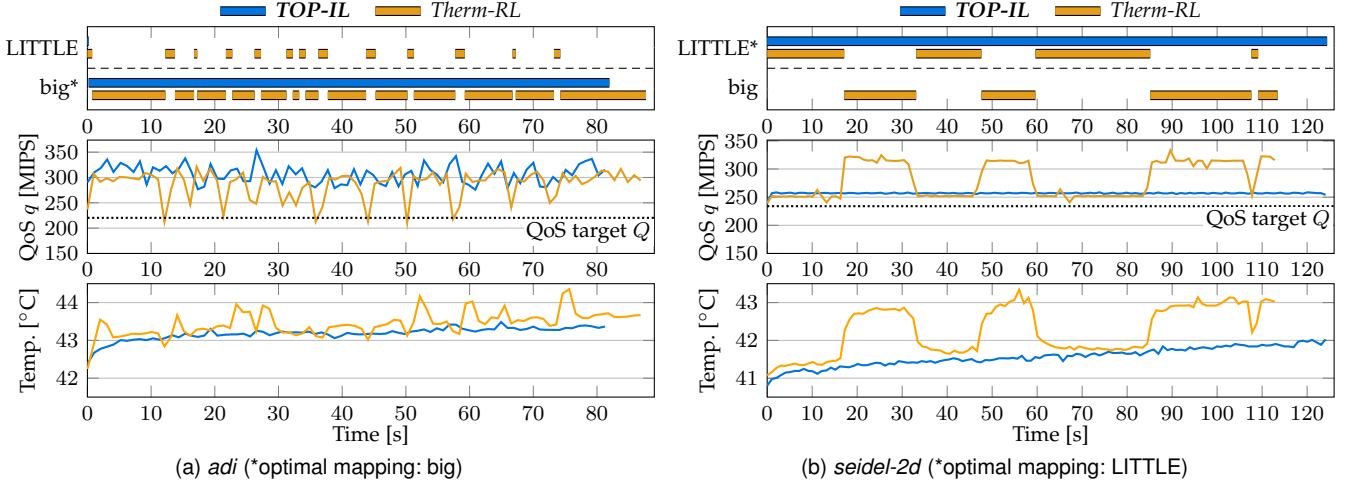(b) *seidel-2d* (*optimal mapping: LITTLE)

Fig. 8. Illustrative example demonstrating the mappings chosen by our *TOP-IL* and *Therm-RL* with the two applications *adi* and *seidel* studied already in Fig. 1. Our *TOP-IL* selects the optimal mapping for both applications. *Therm-RL* in general shows a similar trend but is unstable, selecting also suboptimal mappings. The QoS targets are reached in all cases. However, *Therm-RL* increases the temperature during suboptimal mappings.

(a) With a fan (same as for oracle demonstrations)

(b) Without a fan
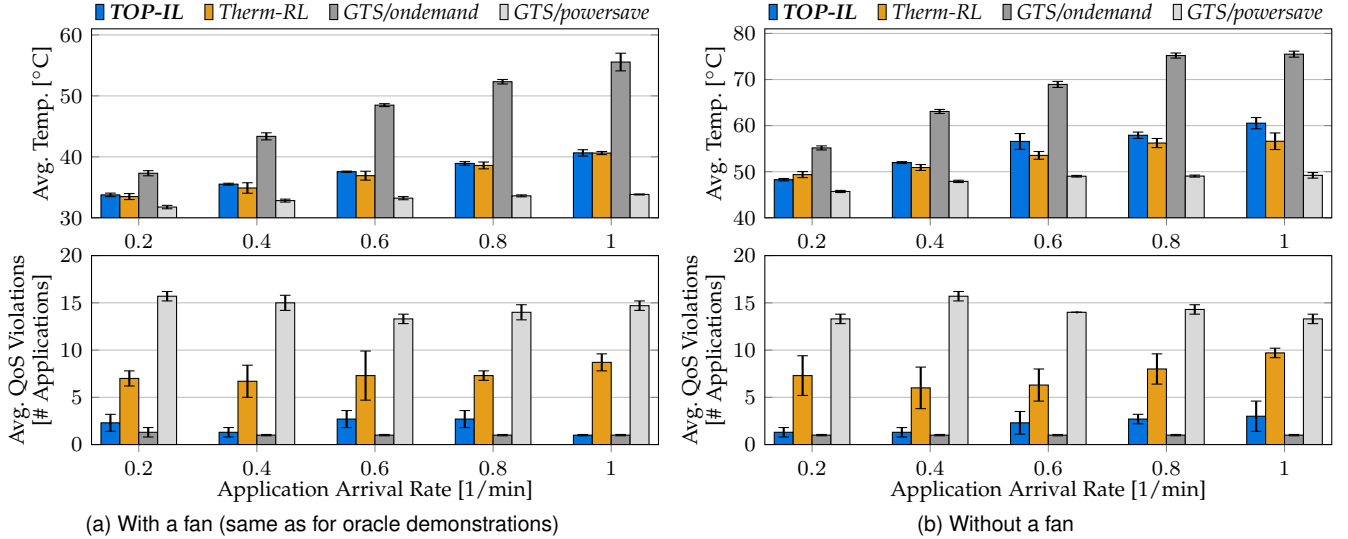
Fig. 9. Main results: Our *TOP-IL* significantly reduces the temperature, while achieving low QoS violations. This is the case both when running with a fan, as when recording the traces for the oracle demonstrations, but also without the fan, demonstrating the generalization of our model. Bars show mean and standard deviation over three experiments. *TOP-IL* and RL use models trained with different random seeds.

big cluster is optimal. *TOP-IL* always selects the optimal mapping. *Therm-RL* also mostly selects a mapping to the big cluster but infrequently migrates *adi* to the LITTLE cluster. In both cases, *adi* reaches its QoS target. The temperature reached by the two techniques is also similar, as they select the same mapping most of the time. Fig. 8b shows the mappings selected with *seidel-2d*, for which the LITTLE cluster is optimal. *TOP-IL* again consistently selects the optimal mapping. In contrast, *Therm-RL* is more unstable and migrates *seidel-2d* irregularly between both clusters. This results in an unnecessarily high QoS during the time on the big cluster, which also results in a higher temperature during these periods. These examples illustrate that the policy learned with IL is stable and consistently selects the optimal mapping, in contrast to RL, which is more unstable. This ultimately results in a lower temperature. The instability of RL leads to even worse results (QoS violations) with more

realistic workloads with multiple parallel applications, as will be shown in the next section.

7.2 Main Experiment: Parallel Mixed Workload

We now evaluate the capabilities of all techniques to reduce the temperature under QoS targets. We create a mixed workload of 20 randomly selected applications among *blackscholes*, *bodytrack*, *canneal*, *dedup*, *facesim*, *ferret*, *fluidanimate*, and *swaptions* from PARSEC [34], and *adi*, *fdtd-2d*, *floydwarshall*, *gramschmidt*, *heat-3d*, *jacobi-2d*, *seidel-2d*, and *syr2k* from *Polybench* [7]. Only the *Polybench* applications (except *jacobi-2d*) have been used for training *TOP-IL* and *Therm-RL*. All other applications are unseen. We select a random QoS target for each application. The arrival times are distributed by a *Poisson* distribution with varying arrival rate to test different system loads. With *TOP-IL*, the average/peak
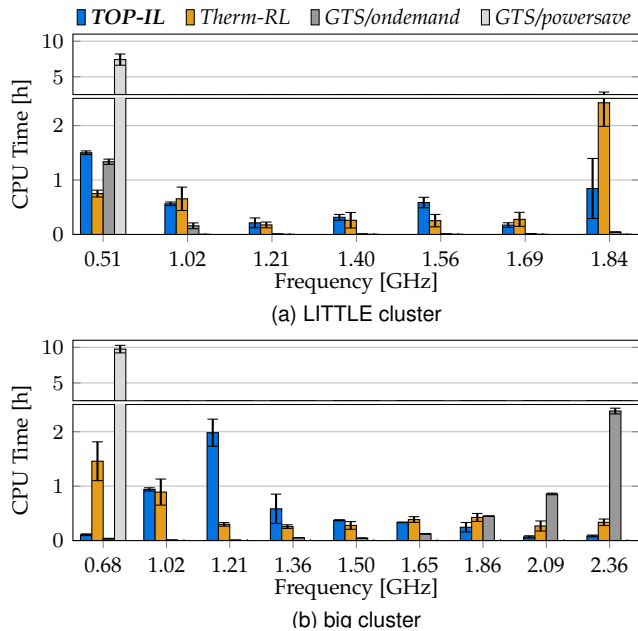
Fig. 10. Total CPU time (among all arrival rates) per cluster and V/f level per technique in the experiments without a fan (Fig. 9b).
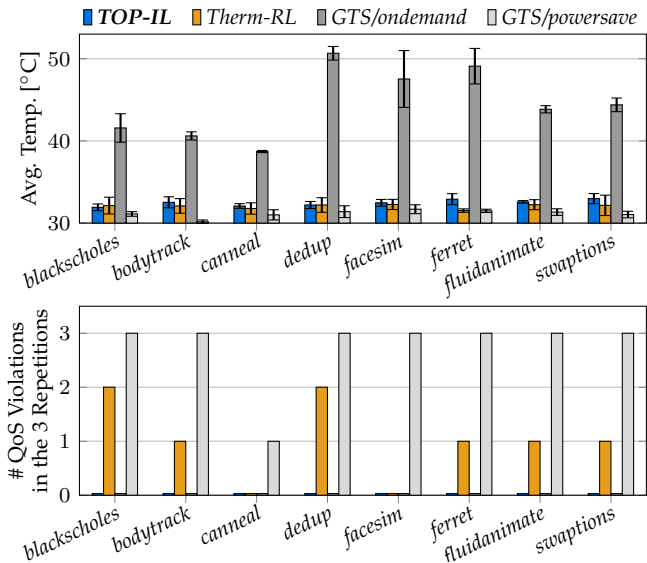

Fig. 11. Our *TOP-IL* is the only technique to achieve no performance violations, yet low temperature for all single application workloads. All applications are unseen, i.e., not used for training.

system utilizations vary from 13 %/38 % to 37 %/75 %, for minimum and maximum arrival rates, respectively. We let the board cool down for 10 min between experiments. All experiments are performed three times (with different models for *TOP-IL* and *Therm-RL*), as explained earlier.

Figs. 9a and 9b shows the results (mean and standard deviation for three repetitions) for the cooling with a fan, i.e., like for training data generation, *and without a fan*, i.e., different from the training data, respectively. *TOP-IL* reduces the average temperature by up to 17 °C compared to *GTS/ondemand* at only slightly more QoS violations. *GTS/powersave* achieves the lowest temperature but the majority of applications violate their QoS target. Finally, the temperature with *Therm-RL* is similar to *TOP-IL*. However, *TOP-IL* achieves 63 % to 89 % fewer QoS violations. *TOP-IL* is the only technique to achieve temperature minimization at few QoS violations. This result is independent of the cooling.

To explain these results we analyze the selected mappings and V/f levels. Fig. 10 plots the distribution (mean and standard deviation for the three repetitions) of the total CPU time (time executing an application) for executing the workload at all arrival rates according to the cluster and selected V/f level for the experiment without a fan. GTS favors the big cluster and *ondemand* selects high frequencies when applications are executed. As a result, *GTS/ondemand* uses most CPU time at the highest V/f level on the big cluster, leading to low QoS violations. However, this also leads to high temperature and ultimately even causes thermal throttling, forcing *GTS/ondemand* to occasionally reduce the V/f levels. In contrast, *powersave* always selects the lowest V/f level. The reduced performance increases the number of simultaneously running applications, which forces GTS to also use the LITTLE cluster. As a result, *GTS/powersave* uses CPU time on both clusters at the lowest V/f level, leading to the lowest temperature but many QoS violations. *Therm-RL* uses a lot of CPU time on the LITTLE cluster at

the highest V/f level and on the big cluster at the lowest V/f level. In both cases, a migration to the other cluster would likely have been beneficial to either be able to satisfy the QoS target, or to reduce the temperature. In particular, the high CPU time spent on the LITTLE cluster at peak V/f level explains the high number of QoS violations. The reason for the suboptimal mapping decisions of *Therm-RL* are policy instability due to continual exploration in online learning and combining objectives and constraints into a single scalar reward. In contrast, *TOP-IL* uses more time on the big cluster at rather low V/f levels, which allows it to meet the QoS target at a low temperature, as seen in Fig. 9. We also did this analysis for the experiment with a fan and found similar results (except for no throttling with *GTS/ondemand*). *In summary, TOP-IL is the only technique to achieve temperature minimization at low QoS violations. This is achieved for mixed workloads containing unseen applications, for different cooling setting than used during training, and is reproducible for models trained with different random initialization.*

7.3 Single-Application Workloads

The results of the previous section contain both seen and unseen applications. To further demonstrate the generalization, we run experiments with *only unseen* applications. The QoS targets are set such that they can be met at the highest V/f level on the LITTLE cluster. As in the previous section, we repeat each experiment three times with different IL or RL models. Fig. 11 visualizes the results in terms of average temperature and QoS violations. As in the previous experiments, *GTS/ondemand* reaches the highest temperature. The other three techniques all result in a similar low temperature. As there is only one application per workload, it can either reach or violate its QoS target. We therefore report the number of executions with a QoS violation instead of the average number of applications that violate their QoS. As expected, *GTS/powersave* violates almost all QoS targets. The only exception is *canneal*, which is memory-intensive and
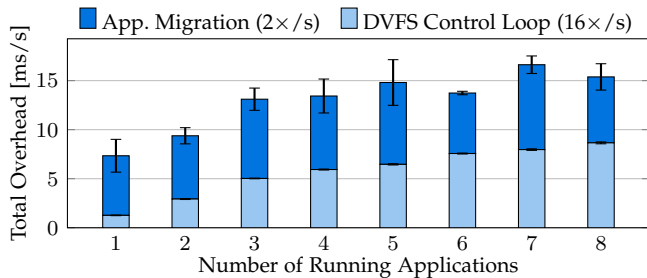
Fig. 12. The overhead of the DVFS control loop increases with the number of executed executions, whereas application migration has a constantly low overhead due to the parallel NN inference with the NPU.

its performance depends less on the CPU V/f level. *Therm-RL* also violates the performance constraint in 33 % of the executions. The reason is that the policy learned with RL suffers from instabilities, which causes frequent migrations. After each migration, the DVFS control loop requires a few iterations to determine the V/f level. During this time, the QoS may be temporarily violated, potentially resulting in a global QoS violation among the whole execution. The only technique that achieves both a low temperature and no QoS violations is *TOP-IL*. These experiments demonstrate again the capabilities of *TOP-IL* to effectively minimize the temperature under a QoS target, but most importantly also *the generalization capabilities of TOP-IL to unseen applications*.

7.4 Model Evaluation

This section evaluates the NN model in isolation. We split the training/test data into training and test based on the AoI, where seven out of nine benchmarks are only used for training (same as in previous sections), and others only for testing. As discussed earlier, our goal is to select any near-optimal mapping in case several mappings result in a similar temperature. The following reports the mean and standard deviation across three models trained with different random seeds. Our model selects a mapping within $1\,°C$ of the optimum in $82\pm5\,\%$ of the cases. The selected mapping is, on average, only $0.5\pm0.2\,°C$ hotter than the optimum. *This demonstrates that our training process is robust and consistently creates models that make near-optimal decisions*.

7.5 Run-Time Overhead

The results in Figs. 8 to 11 already inherently contain the run-time overhead (additional CPU load, induced temperature) of *TOP-IL* as it is running in parallel to the workload. We perform in this section additional experiments to explicitly evaluate the overhead of our technique. We study different system utilization values, i.e., different numbers of running applications. Fig. 12 presents the results. The DVFS control loop is executed 16 times per second. Its overhead increases with the number of managed applications. The main component is reading the performance counters, which scales linearly with the number of applications. In contrast, the overhead of the migration policy, which is executed twice per second, barely changes with more running applications. This is as its main component is the NN inference, which uses parallel inference of the NN, and thereby maintains a constant low latency. In the

worst case, the DVFS control loop and migration policy have an overhead of $8.7\,ms/s$ and $8.6\,ms/s$ ($0.54\,ms$ and $4.3\,ms$ per invocation), respectively. The total run-time overhead of *TOP-IL* is $\leq 1.7\,\%$, and therefore negligible. It is important to notice that we use a single-threaded implementation of *TOP-IL*, i.e., the overhead only affects a single core.

8 CONCLUSION

Temperature minimization under QoS targets requires application migration and DVFS. Optimization can only be achieved by jointly considering the diverse characteristics and QoS targets of all running applications, and, hence, is a complex problem. We tackle the complexity with NN-based IL, which enables us to combine the optimality of the oracle policy with a low run-time overhead. We employ the existing NPU of a smartphone SoC to accelerate the run-time inference. Our policy offers stable management and generalizes to different workloads and cooling settings than what has been used for training.

ACKNOWLEDGMENTS

REFERENCES

[1] M. Rapp, N. Krohmer, H. Khdr, and J. Henkel, "NPU-Accelerated Imitation Learning for Thermal- and QoS-Aware Optimization of Heterogeneous Multi-Cores," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 2022.

[2] H. Khdr, H. Amrouch, and J. Henkel, "Aging-Constrained Performance Optimization for Multi Cores," in *Design Automation Conf. (DAC)*. IEEE, 2018.

[3] S. Wang and J.-J. Chen, "Thermal-Aware Lifetime Reliability in Multicore Systems," in *Int. Symp. on Quality Electronic Design (ISQED)*, 2010.

[4] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin, "User-Specific Skin Temperature-Aware DVFS for Smartphones," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2015, pp. 1217–1220.

[5] A. Pathania, H. Khdr, M. Shafique, T. Mitra, and J. Henkel, "QoS-aware Stochastic Power Management for Many-Cores," in *Design Automation Conf. (DAC)*, 2018.

[6] J. Henkel, H. Khdr, and M. Rapp, "Smart Thermal Management for Heterogeneous Multicores," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 2019, pp. 132–137.

[7] T. Yuki and L.-N. Pouchet, "Polybench 4.0," 2015.

[8] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms," *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 544–557, 2017.

[9] K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett, "AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 10, pp. 2206–2217, 2019.

[10] M. Rapp, M. B. Sikal, H. Khdr, and J. Henkel, "SmartBoost: Lightweight ML-Driven Boosting for Thermally-Constrained Many-Core Processors," in *Design Automation Conf. (DAC)*, 2021.

[11] A. T. Goh, "Back-Propagation Neural Networks for Modeling Complex Systems," *Artificial Intelligence in Engineering*, vol. 9, no. 3, pp. 143–151, 1995.

[12] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," *arXiv preprint arXiv:1606.06565*, 2016.

[13] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Y. Ogras, "DyPO: Dynamic Pareto-Optimal Configuration Selection for Heterogeneous MPSoCs," *ACM Tran. on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, 2017.

[14] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation Learning for Dynamic VFI Control in Large-Scale Manycore Systems," *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2458–2471, 2017.

[15] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning," *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2842–2854, 2019.

[16] A. L. Sartor, A. Krishnakumar, S. E. Arda, U. Y. Ogras, and R. Marculescu, "Hilite: Hierarchical and Lightweight Imitation Learning for Power Management of Embedded SoCs," *IEEE Computer Architecture Letters (CAL)*, vol. 19, no. 1, pp. 63–67, 2020.

[17] Q. Fettes, M. Clark, R. Bunescu, A. Karanth, and A. Louri, "Dynamic Voltage and Frequency Scaling in NoCs with Supervised and Reinforcement Learning Techniques," *IEEE Tran. on Computers (TC)*, vol. 68, no. 3, pp. 375–389, 2019.

[18] E. Kwon, S. Han, Y. Park, J. Yoon, and S. Kang, "Reinforcement Learning-Based Power Management Policy for Mobile Device Systems," *IEEE Tran. on Circuits and Systems I: Regular Papers (TCAS-I)*, 2021.

[19] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, "AI Benchmark: Running Deep Neural Networks on Android Smartphones," in *Euro. Conf. on Computer Vision (ECCV)*, 2018.

[20] Z. Chen, D. Stamoulis, S. Member, and D. Marculescu, "Profit : Priority and Power / Performance Optimization for Many-Core Systems," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2064–2075, 2018.

[21] S. M. P. Dinakarrao, A. Joseph, A. Haridass, M. Shafique, J. Henkel, and H. Homayoun, "Application and Thermal-Reliability-Aware Reinforcement Learning based Multi-Core Power Management," *ACM Jrnl. on Emerging Tech. in Computing Systems (JETC)*, vol. 15, no. 4, 2019.

[22] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-based Inter-and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Design Automation Conf. (DAC)*, 2014.

[23] B. Donyanavard, A. Sadighi, F. Maurer, T. Mück, A. Rahmani, A. Herkersdorf, and N. Dutt, "SOSA: Self-optimizing Learning with Self-adaptive Control for Hierarchical SoC Management," *Int. Symp. on Microarchitecture (MICRO)*, pp. 685–698, 2019.

[24] S. Lu, R. Tessier, and W. Burleson, "Reinforcement Learning for Thermal-Aware Many-Core Task Allocation," in *Great Lakes Symp. on VLSI (GLSVLI)*, 2015, pp. 379–384.

[25] S.-G. Yang, Y.-Y. Wang, D. Liu, X. Jiang, H. Fang, Y. Yang, and M. Zhao, "ReLeTa: Reinforcement Learning for Thermal-Aware Task Allocation on Multicore," *arXiv preprint arXiv:1912.00189*, 2019.

[26] D. Liu, S.-G. Yang, Z. He, M. Zhao, and W. Liu, "CAR-TAD: Compiler-Assisted Reinforcement Learning for Thermal-Aware Task Scheduling and DVFS on Multicores," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

[27] B. Jeff, "big.LITTLE Technology Moves Towards Fully Heterogeneous Global Task Scheduling," *ARM white paper*, 2013.

[28] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Pan, M. Wolf, and J. Henkel, "MLCAD: A Survey of Research in Machine Learning for CAD (Keynote Paper)," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

[29] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosing, "P^4: Phase-Based Power/Performance Prediction of Heterogeneous Systems via Neural Networks," in *International Conf. on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 683–690.

[30] Linaro 96Boards, "Hikey970," https://96boards.org/product/hikey970/.

[31] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic Learning for Thermal-Aware Power Budgeting in Many-Core Architectures," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES)*. ACM, 2011, pp. 189–196.

[32] S. Ross, G. Gordon, and D. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *International Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2011, pp. 627–635.

[33] R. Jain, P. R. Panda, and S. Subramoney, "Cooperative Multi-Agent Reinforcement Learning-based Co-Optimization of Cores, Caches, and On-Chip Network," *ACM Tran. on Architecture and Code Optimization (TACO)*, vol. 14, no. 4, 2017.

[34] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2008.

Martin Rapp has successfully defended his Ph.D. in Computer Science at Karlsruhe Institute of Technology (KIT) in May 2022 under the supervision of Prof. Dr. Jörg Henkel. Mr. Rapp received a B.Sc. and M.Sc. degree – both with distinction – in Computer Science from the KIT in 2014 and 2016, respectively. His current research focuses on resource-constrained machine learning: ML-based run-time resource management for many-core architectures and distributed resource-aware on-device training of neural networks. ORCID 0000-0002-5989-2950

Heba Khdr received her Ph.D. (Dr.-Ing.) in Computer Science from Karlsruhe Institute of Technology (KIT) in July 2018 under the supervision of Prof. Jörg Henkel. Mrs. Khdr received her B. Sc in Computer Science from Aleppo University in Syria, with excellent grade and the first rank. She is currently a research group leader at the Chair for Embedded Systems (CES) at KIT. Her main research interests are resource management techniques that consider power, temperature and aging issues in embedded processors.

Nikita Krohmer received a B.Sc. degree in Computer Science from Technical University of Berlin in 2018 and a M.Sc. degree in Computer Science from Karlsruhe Institute of Technology in 2021. His research interests lie in embedded machine learning and applied artificial intelligence.

Jörg Henkel received the Diploma and Ph.D. (summa cum laude) degrees from the Technical University of Braunschweig, Germany. He was a Research Staff Member with NEC Laboratories, Princeton, NJ, and is currently the Chair Professor of embedded systems with the Karlsruhe Institute of Technology, Karlsruhe, Germany. His research focus is on co-design for embedded hardware/software systems. Dr. Henkel has received six best paper awards from major CAD conferences. He served as the Editor-in-Chief for the ACM Transactions on Embedded Computing Systems and IEEE Design&Test. He has led several conferences as a General Chair incl. ICCAD, ESWeek etc. He coordinates the DFG Program SPP 1500 "Dependable Embedded Systems" and is a site coordinator of the DFG TR89 collaborative research center on "Invasive Computing." He is the Chairman of the IEEE Computer Society, Germany Chapter. He is a Fellow of the IEEE.