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

MARTIN RAPP, HEBA KHDR, NIKITA KROHMER, and JÖRG HENKEL, Karlsruhe Institute of Technology (KIT), Germany

Thermal optimization of a heterogeneous clustered multi-core processor under user-defined quality of service (QoS) targets requires application migration and dynamic voltage and frequency scaling (DVFS). 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 for power or temperature minimization either relies on measurements that are commonly not available (such as power) or fails to consider all the dimensions of the optimization (e.g., by using simplified analytical models). To solve this, machine learning (ML) methods can be employed. In particular, imitation learning (IL) leverages 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) at design time and accelerate the run-time NN inference using a neural processing unit (NPU). While such NN accelerators are becoming increasingly widespread, they are so far only used to accelerate user applications. In contrast, we use for the first time an existing accelerator on a real platform to accelerate NN-based resource management. To show the superiority of IL compared to reinforcement learning (RL) in our targeted problem, we also develop multi-agent RL-based management. Our evaluation on a HiKey 970 board with an Arm big.LITTLE CPU and an NPU shows that IL achieves significant temperature reductions at a negligible run-time overhead. We compare TOP-IL against several techniques. Compared to ondemand Linux governor, TOP-IL reduces the average temperature by up to 17°C at minimal QoS violations for both techniques. Compared to the RL policy, our TOP-IL achieves 63% to 89% fewer QoS violations while resulting similar average temperatures. Moreover, TOP-IL outperforms the RL policy in terms of stability. We additionally show that our IL-based technique also generalizes to different software (unseen applications) and even hardware (different cooling) than used for training.

CCS Concepts: • Hardware → Temperature optimization; • Computing methodologies → Neural networks; • Computer systems organization → Multicore architectures;

Additional Key Words and Phrases: Machine learning, imitation learning, neural networks, AI accelerators, thermal management, quality of service, processor scheduling, task migration

This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—Project Number 146371743—TRR 89 Invasive Computing.

Authors’ address: M. Rapp, H. Khdr, N. Krohmer, and Jörg Henkel, Karlsruhe Institute of Technology (KIT), Haid-und-Neu-Str. 7, Karlsruhe 76131, Germany; e-mails: {martin.rapp, heba.khdr}@kit.edu, nikita-krohmer@web.de, henkel@kit.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM. 1084-4309/2023/11-ART16 $15.00 https://doi.org/10.1145/3626320

ACM Transactions on Design Automation of Electronic Systems, Vol. 29, No. 1, Article 16. Pub. date: November 2023.
1 INTRODUCTION

Elevated on-chip temperature has severe impacts on chip reliability, as it accelerates aging mechanisms, including electromigration, time-dependent dielectric breakdown, and Bias Temperature Instability [29]. Enforcing a temperature constraint helps to ensure reliable operation on the chip during the designed lifetime. However, minimizing temperature will even help to improve reliability and prolong lifetime [8]. Therefore, many thermal management techniques in the literature have targeted to minimize temperature, such as [8, 26, 41]. Moreover, the goal of minimizing temperature is of paramount importance on smartphones and tablet devices. In particular, these devices are often touched by user skin from the back cover and the screen for relatively long periods, thereby heat dissipation from these devices directly affects user experience. As reported in [12], complaints of excessive temperatures are common to many smartphones. In summary, minimizing on-chip temperatures will help in improving both system reliability and user experience.

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 [30]. The reason is that the impact on performance and power when migrating an application between clusters differs from one application to another [16]. 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 Figure 1, we execute one application, adi or seidel-2d from the Polybench [42] suite, on an Arm big.LITTLE 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, but requires 1.0 GHz on the big cluster, resulting in a slightly lower temperature on 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 required to operate both clusters at the peak V/f level to reach their QoS targets. Since our

A subset of this work was first published in [33].
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.

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\(^1\) 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 the 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 a few or even a single temperature sensors.

Many works perform optimization with models for individual aspects such as power, performance, or temperature. These models can be built analytically\(^3\) or by ML\(^2, 34\). 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 RL and IL. In both cases, NN learning can be used to cope with the high complexity\(^4\).

RL suffers from several problems. It requires combining objectives and constraints in a single scalar reward, which does not reflect their different properties and may lead to suboptimal actions (reward hacking\(^1\)). 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

\(^1\)Note that unless noted otherwise, complexity in this context refers to the complex behavior of hardware and software, which is difficult to model accurately, not to algorithmic complexity.
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 [15, 22, 28, 37], 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 [13, 24]. 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 [18]. 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 multi-agent 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 policy that is learned by IL generalizes to unseen workloads and different cooling settings than what has been used during training.

The remainder of this article is organized as follows. We discuss the differences between our work to related work in Section 2. The problem formulation is defined in Section 3. We introduce the design-time process of our IL-based application migration technique, including feature selection, training data generation using Oracle demonstrations, and NN model design and training in Section 4. The run-time aspects of TOP-IL comprise application migration and DVFS in Section 5. Section 6 introduces an RL-based application migration technique, which serves as a comparison to TOP-IL. Finally, we present comprehensive experimental studies in Section 7 and conclude our article in Section 8.

2 RELATED WORK

The state-of-the-practice Android/Linux resource management [20] 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 tradeoff between power and performance. However, these techniques do not consider application characteristics nor their QoS targets, and only indirectly affect the temperature (via power or energy). Research works consider application characteristics while managing on-chip resources targeting several optimization problems on heterogeneous multi-core processors. For instance, the technique proposed in [21] targets to maximize the performance under a temperature constraint. A scheduling technique presented in [36] aims at meeting fairness constraints while maximizing the energy efficiency. The work in [30] minimizes
the power under QoS targets via stochastic power budgeting. A survey summarizing many resource management techniques for heterogeneous multi-core processors is presented in [38]. A large body of these techniques depend on profiling applications at design time to build models that estimate application-specific power and performance. Therefore, they cannot be employed for unknown applications. Coping with unknown applications requires prediction models. Several related works have created analytical models to predict application-specific power and performance of heterogeneous multi-core processors [31, 39, 40]. Many of these works [31, 40] only develop prediction models but do not present any resource management technique. The work in [39] minimizes the energy under QoS targets using analytical energy and performance models to predict run-time contention. Building analytical models is not always feasible due to the complexity involved in several software and hardware aspects. In contrast, ML is a promising solution to build models that cope with complexity in software and hardware and can generalize to unseen workloads. Therefore, recent works have employed ML to build prediction models that support resource management [32].

Various ML approaches can be used for this purpose. Supervised learning can be used to train models that predict system properties like performance, power, or temperature [23]. However, model training requires access to measurements like per-core/per-cluster power, which are often not available in real-world processors [25]. Another branch of ML algorithms enables to directly select thermal management decisions without the need to build a power/temperature model first that requires power measurements that might not be available as discussed earlier. This can be achieved with RL or IL.

RL has been employed in several works for power or thermal optimization [11]. The works in [6, 9, 24] use RL for power management via DVFS. However, they neither consider temperature nor QoS. The work in [7] 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 [7] minimizes thermal cycling, which is unrelated to the absolute temperature. In addition, the work does not cope with several applications running in parallel. In [10], 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 [27] performs migration for temperature minimization based on per-core temperature measurements. In [41], 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.

Several recent works employ IL for system-level optimization. The work in [15] trains a model to predict the optimal number of active cores and per-cluster V/f levels to minimize the energy. In [22], an IL technique is proposed for DVFS to minimize the energy under a QoS target. They train a separate policy per application, and, hence, cannot cope with unknown applications. The work in [28] 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 target. Finally, a hierarchical IL technique is proposed in [37] 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 snippets (sequences of executed instructions) and record performance counters, performance, and power for each snippet at different configurations (number of active cores, V/f levels, etc.). Oracle demonstrations are created by finding the optimal sequence
Table 1. Overview of Related Work

| Technique     | Method | Goal | Map./Mig. | DVFS | Temp. | QoS | Per-clust. | Het. | Unkn. | Multi- | Lim. | Power | Sensors |
|---------------|--------|------|-----------|------|-------|-----|------------|------|-------|--------|------|-------|---------|
| ondemand/     | Rules  | max perf./ | ✓ | ✓ | x | x | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| powersave     | min P  | | | | | | | | | | | |
| [36] Non-ML   | max E eff. st F | ✓ | ✓ | x | x | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| [21] Non-ML   | max perf. st T | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [30] Non-ML   | min P st QoS | x | ✓ | ✓ | ✓ | x | x | x | ✓ | ✓ | ✓ | ✓ |
| [31] Non-ML   | model perf.. P | (✓)* | x | x | x | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| [40] Non-ML   | model E | (✓)* (✓)* | x | x | ✓ | ✓ | ✓ | x | x | x | x | x |
| [39] Non-ML   | min E st QoS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| [6] RL       | max perf. st P | x | ✓ | x | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| [9] RL       | min E st R | x | ✓ | x | x | x | x | x | x | ✓ | ✓ | ✓ |
| [24] RL      | min EDP | x | ✓ | x | x | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| [7] RL      | max R st QoS | ✓ | ✓ | x | ✓ | (✓)** | (✓)** | ✓ | x | ✓ | ✓ | ✓ |
| [10] RL      | min P st QoS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| [27] RL      | min T | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [41] RL      | min T | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [26] RL      | min T st QoS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| [15] IL      | min E | (✓)** | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [22] IL      | min E st QoS | ✓ | ✓ | x | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [28] IL      | min E st QoS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [37] IL      | min E st QoS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| TOP-IL (ours)| IL min T st QoS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

T: temperature, P: power, E: energy, R: reliability, F: fairness.
*Only develop prediction models, no resource management technique.
**Not studied, likely applicable with minor changes.
***Controls the number of active cores.

of configurations per snippet. This only works because power, performance, and energy of a given snippet depend only on the used configuration in this snippet. 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 snippet additionally depends on all configurations of all previous snippets. 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. 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.

Table 1 provides an overview of the related work. 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 preemptive 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. The processor executes several independent aperiodic single-threaded applications in parallel (multi-program execution), each with its own QoS target \( Q_k \) and current QoS \( q_k \), which are expressed in terms of the IPS. Other metrics like application heartbeats [17] could also be used instead if available but typically require changes to the applications. However, these metrics must be comparable across applications, i.e., the same metric must be used across all managed applications. The one-thread-per-core model is used [5]. 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 [25].

\(^2\)QoS targets can also be seen as a soft-real time constraint.
**Objective** minimize the on-chip temperature

**Constraint** maintain QoS of applications (IPS)

**Knobs** application-to-core mapping and migration, per-cluster DVFS.

As discussed in the introduction, there is high complexity in the behavior of hardware and software, which is difficult to model accurately. For instance, it is too complex to build a run-time capable analytical model that estimates the impact of migrating applications on the performance of all running applications and on the chip temperature. The reason is that the exact instruction sequence of all applications, microarchitecture of the CPU pipeline, caches, interconnects and buses, memory, the floorplan, and the semiconductor technology all play a role. Furthermore, the complexity of the optimization space (action space), i.e., the number of possible resource management actions at each control epoch, is also high. In particular, when there are \( k \) applications running on \( n \) cores that belong to several clusters with V/f levels \( \mathcal{F}_x \) each, the number of possible migration actions and V/f level selections, considering one-thread-per-core model is \( \frac{n!}{(n-k)!} \cdot \prod_x |\mathcal{F}_x| \). This number is already large for rather small systems but even grows exponentially with the number of cores and parallel applications. In our setup, \((n = 8, x \in \{l, b\}, |\mathcal{F}_l| = 7, \text{ and } |\mathcal{F}_b| = 9, \) this is up to 2,540,160 possible actions in each control epoch (maximum number for \( k = n = 8 \)) to select from.

We tackle the high complexity of the hardware and software, combined with high complexity of the optimization space by (1) employing an NN, with (2) the further simplifications to only migrate one application in each control epoch, and (3) to separate migration and DVFS policies.

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.

Our target platform is an Arm big.LITTLE chip, which has two clusters, LITTLE and big, i.e., \( x \in \{l, b\} \), and each cluster has four cores. For clarity, the following sections describe TOP-IL for this target platform. Nevertheless, our solution is also applicable to different numbers of clusters and cores per cluster.

### 4 IL-BASED APPLICATION MIGRATION

Employing IL requires to select features, create oracle demonstrations, and train the model that will be 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, and linear scaling from the current V/f level $f_{x(k)}$ of its cluster $x(k)$ is performed instead:

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

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. Equation (1) assumes linear scaling of the performance with the frequency. This might not always be the case, e.g., with memory-bound applications, whose performance scales sublinearly with the frequency. We discuss the impact of inaccurate estimates of $\tilde{f}_{k,\min}$ in Section 5.2. However, performing linear scaling at run time enables us to compute approximate values of $\tilde{f}_{k,\min}$ and $\tilde{f}_{x\backslash AoI}$ fast, and is still sufficiently accurate to be used as a feature in the model. Finally, the required V/f level without the AoI is determined per cluster $x$ as the maximum among all other applications and used as a feature:

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

All features are normalized to be usable with an NN.

### 4.2 Oracle Demonstrations (Training Data)

The training data needs to indicate the optimal migration w.r.t. QoS and peak 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. In particular, we periodically sample these values at a granularity of 20 Hz.

**Collect Traces:** The process of collecting traces is depicted in the upper part of Figure 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., the 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 afterward 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

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

Table 2. The Selected Features for IL-based Migration (per Application)
evaluation demonstrates, our model also generalizes to applications with execution phases. To further accelerate collecting traces, we obtain traces for a reduced set of V/f levels and 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. However, TOP-IL supports applications with more executed instructions at run time, as our evaluation shows. We execute the background of each scenario for 2 min before starting the AoI to ensure a consistent initial temperature. The order of executions is randomized 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. Our evaluation shows that the trained NN also can be used without retraining for different cooling, i.e., without a fan.

Figure 3(a) and (b) present an illustrative excerpt of the collected traces (performance of the AoI and peak 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 Figure 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
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 peak 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 the required V/f levels of the background $\tilde{f}_l^{AoI}$, $\tilde{f}_b^{AoI}$. 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 $Q_{AoI}$, $\tilde{f}_l^{AoI}$, $\tilde{f}_b^{AoI}$:

$$f_l, f_b = \arg\min_{f_l', f_b'} (f_l' \geq \tilde{f}_l^{AoI} \land f_b' \geq \tilde{f}_b^{AoI} \land q_{AoI}(f_l', f_b') \geq Q_{AoI}).$$

The peak temperature for each mapping of the AoI to each free core $j$ is determined from the periodic measurements. 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 $1\degree C$ 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...
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 $\alpha$ determines a tradeoff between tolerating slightly higher temperatures and susceptibility to temperature measurement noise. We empirically set $\alpha = 1$. Figure 3(c) lists some illustrative examples. For instance, when selecting $Q_{AoI} = 400 \cdot 10^6$ IPS, $\tilde{f}_{L\setminus AoI} = 1.4$ GHz, and $\tilde{f}_{B\setminus AoI} = 0.7$ 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 $^\circ$C and 46.6 $^\circ$C, i.e., a mapping to core 3 is cooler. Therefore, the respective labels for cores 3 and 6 are 1 and 0.02. Figure 3(c) 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 on, i.e., each source of a migration to the optimal core. This is illustrated in Figure 3(d) with a few examples. By creating one training example for every free core for each selection of $Q_{AoI}$, $\tilde{f}_{L\setminus AoI}$, and $\tilde{f}_{B\setminus AoI}$, 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 to employ algorithms like DAGger [35], which initially only trains the policy on the optimal sequence of management decisions, and only gradually adds training data to recover from suboptimal decisions to increase the robustness of the model. We created 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). Figure 4 shows the result of the grid search to determine the
Fig. 5. Illustration of TOP-IL at run time. Application migration uses the NPU to accelerate predicting the best migration per application.

depth and width of the NN. The best topology uses four hidden layers with 64 neurons, each. The hidden layers use ReLU activation, the output layer with eight neurons does not use an activation function. We use Adam optimizer with momentum. The exponentially decaying learning rate is set at $0.01 \cdot 0.95^{(\text{epoch})}$. 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 (Figure 5) integrates IL-based application migration with a per-cluster DVFS control loop. It is based on the IL model that is trained at design time as described in the previous section. We do not perform any retraining at run time, which enables a very low run-time overhead, as will be evaluated in Section 7.5.

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, 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 at design time for one AoI, which is migrated, and several other background applications. We perform parallel inference, where each application is used as the AoI once. Thereby, we can evaluate all potential migrations of all active applications to each core in a single (batched) inference pass. 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 $c(k)$:

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

The result of this optimization is to migrate application $\hat{k}$ to core $\hat{c}$. The migration policy is executed each 500 ms. This 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.
Fig. 6. The overhead of application migration is negligible. In the worst case, frequent migration between clusters slows down applications by less than 4% compared to the theoretical expected performance from profiling the applications on both clusters.

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. We do not reserve any specific core for TOP-IL but let Android schedule it to any core. Since the overall run-time overhead of TOP-IL is minimal (as will be reported in Section 7.5), this barely affects the running applications.

Since, we perform migration each 500 ms, also 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. We compare the resulting performance to a theoretical expected performance, which is obtained from profiling the application when it is statically mapped to the big and LITTLE cluster, respectively. All runs use the highest V/f levels on both clusters. The migration overhead \( m \) is calculated by

\[
m = \frac{1}{2} \cdot \frac{1/\text{big} + 1/\text{LITTLE}}{1/\text{migrate}} - 1 \tag{6}
\]

\( t_{\text{big}} \), \( t_{\text{LITTLE}} \), and \( t_{\text{migrate}} \) are the execution times of the applications when executed on a big core, a little core, or migrated periodically between the two. For a given application, i.e., given total number of executed instructions, the performance (in IPS) is proportional to \( 1/t \). 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 Figure 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.

Since we do not assume any design-time knowledge about applications, we map new arriving applications to a random free core. Additionally, the mapping is optimized by migration in the next migration epoch, hence the initial mapping for new arriving applications plays a minor role.
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 $\tilde{f}_{k,\text{min}}$ per application $k$, as defined in Equation (1). It then determines the minimum required V/f level per cluster $x$ to satisfy the QoS target of all applications running on it:

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

Since the run-time estimates of $\tilde{f}_{k,\text{min}}$ 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 Section 7.

**Impact of prediction inaccuracy in $f_{k,\text{min}}$:** The estimations $\tilde{f}_{k,\text{min}}$ are used both by the migration policy and the DVFS control loop. The DVFS control loop ensures that the QoS is maintained. Since it anyways only scales the V/f levels by one step at a time, inaccuracies have no effect as long as the general trend (higher QoS at higher V/f levels still holds). If the migration policy receives inaccurate estimations, it may lead to suboptimal migrations, which may lead to higher temperature because the DVFS control loop needs to go to higher V/f levels. However, it does not lead to QoS violations.

5.3 Algorithmic Complexity of TOP-IL

Our algorithm comprises two parts, application migration and DVFS. Application migration first computes the features for each application as AoI. This step has algorithmic complexity $O(n^2)$ because the computation of $\tilde{f}_x \setminus \text{AoI}$ requires to consider each pair of applications. The next step is the batched NN model inference, which has a theoretical algorithmic complexity of $O(n)$. It is important to notice that for smaller values of $n$, the latency is in fact independent of $n$ by using batch parallelism in the NPU. Finally, determining the best migration to perform also has algorithmic complexity $O(n^2)$ because it needs to consider each pair of applications and cores. Overall, migration has algorithmic complexity $O(n^2)$. The DVFS control loop has algorithmic complexity $O(n)$ because it needs to consider each application only once. Therefore, the overall algorithmic complexity of TOP-IL is $O(n^2)$.

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 the stability of the learned policy. To demonstrate this in a quantitative comparison, there is a need for 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. The closest technique in the literature to ours is the one proposed in [27] because it shares with us the same optimization goal, i.e., temperature minimization, and does not require the existence of power sensors, and can cope with unknown applications. We therefore base the state-of-the-art RL baseline on [27]. Since [27] does not support multiple applications, we extend it by combining it with [19] as the latter does. Additionally, we extend [27] with parts of TOP-IL.
to introduce DVFS and QoS. 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 by 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 Figure 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 [27]. There is one action per core, indicating a migration to this core, i.e., in total eight actions. The $Q$-table contains 2,304 entries, which is similar in size to what is reported in [24].

The reward function needs to combine the objective (temperature minimization) and constraint (QoS target) into a single scalar value. The objective is similar to [27], which only rewards a low temperature $T$: $r=80^\circ\text{C}-T$. We extend it to penalize QoS violations:

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

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

6.2 Multi-Agent Learning for Parallel Applications
As discussed earlier, we combine [27] with [19] and instantiate one RL agent per application. Mediation between the agents is required to avoid (1) contradicting decisions by different agents,
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 as in [19]. 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. Figure 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 in the system.

### 6.3 Algorithmic Complexity

The algorithmic complexity of Therm-RL is the same as that of TOP-IL. The reasons are that (1) the DVFS control loop is the same, (2) the feature computation for the ML migration decision is the same, (3) the RL agents have algorithmic complexity $O(n)$ because there are $O(n)$ agents, each requiring $O(1)$ to determine the $Q$-values for the current state (a row of the $Q$-table), and (4) finding the optimal migration has algorithmic complexity $O(n^2)$ because it requires to evaluate each pair of application and core. Overall, this leads to an algorithmic complexity of $O(n^2)$, i.e., same as for TOP-IL.

### 6.4 Training

We select the training parameters as in [27]. We use an $\epsilon$-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 bootstrapping the agent, 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. During our evaluation, the agents are then already well-trained and mostly perform exploitation of the pre-trained policy. 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 HiKey 970 [25] 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. Only a single thermal sensor is exposed to Linux on the HiKey 970 board. The on-chip temperature is monitored with the on-board thermal sensor with a frequency of 20 Hz. As mentioned in Section 3, we consider single-threaded applications. Nevertheless, multi-threaded applications can be also considered if threads have their own individual QoS targets.

TOP-IL is compared with bootstrapped Therm-RL, 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 clusters, 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 runs on Android 8.0 on HiKey 970.

**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. *Unseen QoS targets* that have not been used for training with a very high probability. This is achieved by using random QoS targets for all applications.
3. *Different cooling:* We perform experiments also with passive cooling (without a fan) instead of the active cooling used for training data generation.
4. *Randomness in the training and at run time:* We train three models with different random seeds to demonstrate the robustness of weight initialization. We then repeat the experiments three times, where each repetition uses a different model, and report the average and standard deviation of results. This demonstrates robustness to run-time variability due to random workload fluctuations.

In addition, we demonstrate

5. 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 Figure 1, i.e., we run the two applications *adi* and *seidel-2d*. Figure 8(a) shows the selected cluster (mapping) of *adi*. A mapping to the 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. Figure 8(b) 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 applications running in parallel, as will be shown in Section 7.3.

### 7.2 Single-Application Workloads

In this section, we evaluate the capabilities of all techniques to reduce the temperature under QoS targets for different applications. To evaluate the generalization to unseen applications, we use applications from the PARSEC benchmark suite [4], where none of them has been used in the training data. In particular, we ran eight experiments, each with one application from PARSEC. These applications are *blackscholes*, *bodytrack*, *canneal*, *dedup*, *facesim*, *ferret*, *fluidanimate*, and *swaptions*.

ACM Transactions on Design Automation of Electronic Systems, Vol. 29, No. 1, Article 16. Pub. date: November 2023.
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 the motivational example in Figure 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.

Fig. 9. 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.

The QoS targets are set such that they can be met at the highest V/f level on the LITTLE cluster. We repeat each experiment three times with different IL or RL models to alleviate randomness in the training data and at run time, as discussed above. Figure 9 visualizes the results in terms of average temperature (mean and standard deviation for three repetitions) and QoS violations. GTS+
ondemand reaches the highest temperature. The other three techniques all result in a similar low temperature. As expected, GTS+powersave violates almost all QoS targets, for all three experiments. Therefore, we can see that the number of violations is three for almost all applications. The only exception is canneal, which is memory-intensive and 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. This causes frequent migrations, as depicted in the illustrative example in Figure 8(a). Frequent migrations may result in low V/f levels during a short period after the migration because the DVFS control loop requires a few iterations to converge to the level required to reach the QoS target. During this time, the QoS may be temporarily violated, potentially resulting in a global QoS violation throughout 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.

7.3 Main Experiment: Parallel Mixed Workload

We consider now more intensive workloads that consist of mixed parallel applications including seen and unseen applications. In particular, we create a mixed workload of 20 randomly selected applications among blackscholes, bodytrack, canneal, dedup, facesim, ferret, fluidanimate, and swappings from PARSEC [4], and adi, fddt-2d, floyd-warshall, gramschmidt, heat-3d, jacobi-2d, seidel-2d, and syr2k from Polybench [42]. 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. Thereby, the average/peak system utilizations vary from 17%/50% to 32%/88%, 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. We report the average number of applications that violate their QoS since each workload has multiple applications.

Figure 10 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, to evaluate generalization to different cooling systems. 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 a few QoS violations. This result is independent of the cooling.

To explain these results we analyze the selected mappings and V/f levels. Figure 11 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
Fig. 10. 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 Therm-RL use models trained with different random seeds.

Fig. 11. Total CPU time (among all arrival rates) per cluster and V/f level per technique in the experiments without a fan (Figure 10(b)).

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 reasons 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 Figure 10. 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.
Fig. 12. The overhead of the DVFS control loop increases with the number of executed applications, whereas application migration has a constantly low overhead due to the parallel NN inference with the NPU.

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 $1 \pm 0.5^\circ C$ of the optimum in $82 \pm 5\%$ of the cases. The selected mapping is, on average, only $0.5 \pm 0.2^\circ C$ 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 Figures 8–11 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. Figure 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 because 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.
REFERENCES

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. arXiv:1606.06565. Retrieved from https://arxiv.org/abs/1606.06565

[2] Karanakar R. Basireddy, Amit Kumar Singh, Bashir M. Al-Hashimi, and Geoff V. Merrett. 2019. AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multicores. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39, 10 (2019), 2206–2217.

[3] Ganapati Bhat, Gaurav Singla, Ali K. Unver, and Umit Y. Ogras. 2017. Algorithmic optimization of thermal and power management for heterogeneous mobile platforms. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26, 3 (2017), 544–557.

[4] Christian Biena, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The parsec benchmark suite: Characterization and architectural implications. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT). ACM.

[5] Silas Boyd-Wickizer, Haibo Chen, Rong Chen, Yandong Mao, Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yuehua Dai, Yang Zhang, and Zheng Zhang. 2008. Corey: An operating system for many cores. In Proceedings of the Symposium Operating System Design and Implementation (OSDI).

[6] Zhao Chen, Dimitrios Stamoulis, Student Member, and Diana Marculescu. 2018. Profit : Priority and power/performance optimization for many-core systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37, 10 (2018), 2064–2075.

[7] Anup Das, Rishad A. Shafik, Geoff V. Merrett, Bashir M. Al-Hashimi, Akash Kumar, and Bharadwaj Veeravalli. 2014. Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multi-core systems. In Proceedings of the Design Automation Conference (DAC).

[8] Anup Kumar Das, Akash Kumar, Bharadwaj Veeravalli, Francky Cattahoor, Anup Kumar Das, Akash Kumar, Bharadwaj Veeravalli, and Francky Cattahoor. 2018. Run-time adaptations for lifetime improvement. Reliable and Energy Efficient Streaming Multiprocessor Systems (2018), 127–142.

[9] Sai Manoj Pudukotai Dinakarrao, Arun Joseph, Anand Haridass, Muhammad Shafique, Jörg Henkel, and Houman Homayoun. 2019. Application and thermal-reliability-aware reinforcement learning based multi-core power management. ACM Journal on Emerging Technologies in Computing Systems 15, 4 (2019), 1–19.

[10] Bryan Donyanavard, Armin Sadighi, Florian Maurer, Tiago Mück, Amir Rahmani, Andreas Herkersdorf, and Nikil Dutt. 2019. SOSA: Self-optimizing learning with self-adaptive control for hierarchical SOC management. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.

[11] Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. 2011. Economic learning for thermal-aware power budgeting in many-core architectures. In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES). ACM, 189–196.

[12] Begum Egilmez, Gokhan Memik, Seda Ogrenci-Memik, and Oguz Ergin. 2015. User-specific skin temperature-aware DVFS for smartphones. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE).

[13] Quintin Fettes, Mark Clark, Razvan Bunescu, Avinash Karanth, and Ahmed Louri. 2019. Dynamic voltage and frequency scaling in NoCs with supervised and reinforcement learning techniques. IEEE Transactions on Computers 68, 3 (2019), 375–389.

[14] Anthony T. C. Goh. 1995. Back-propagation neural networks for modeling complex systems. Artificial Intelligence in Engineering 9, 3 (1995), 143–151.

[15] Ujjwal Gupta, Chetan Arvind Patil, Ganapati Bhat, Prabhath Mishra, and Umit Y. Ogras. 2017. DyPO: Dynamic pareto-optimal configuration selection for heterogeneous MPSoCs. ACM Transactions on Embedded Computing Systems 16, 5s (2017), 1–20.

[16] Jörg Henkel, Heba Khdr, and Martin Rapp. 2019. Smart thermal management for heterogeneous multicores. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE). IEEE, 132–137.

[17] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. 2010. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In Proceedings of the International Conference on Autonomic Computing (iCAC), 79–88.

[18] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. AI Benchmark: Running deep neural networks on android smartphones. In Proceedings of the European Conference on Computer Vision (ECCV).

[19] Rahul Jain, Preeti Ranjan Panda, and Sreenivas Subramoney. 2017. Cooperative multi-agent reinforcement learning-based co-optimization of cores, caches, and on-chip network. ACM Transactions on Architecture and Code Optimization 14, 4 (2017), 1–25.

[20] Brian Jeff. 2013. Big, little technology moves towards fully heterogeneous global task scheduling. ARM White Paper (2013), 1–13.
NPU-Accelerated Imitation Learning for Thermal Optimization

References:

[21] Heba Khdr, Santiago Pagani, Erices Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. 2016. Power density-aware resource management for heterogeneous tiled multicore. IEEE Transactions on Computers 66, 3 (2016), 488–501.

[22] Ryan Gary Kim, Wonje Choi, Zhuo Chen, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2017. Imitation learning for dynamic VFI control in large-scale manycore systems. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 25, 9 (2017), 2458–2471. https://doi.org/10.1109/TVLSI.2017.2700726

[23] Yeseong Kim, Pietro Mercati, Ankit More, Emily Shriver, and Tajana Rosing. 2017. P2: Phase-based power/performance prediction of heterogeneous systems via neural networks. In Proceedings of the International Conference on Computer-Aided Design (ICCAD). IEEE, 683–690.

[24] Eunji Kwon, Sodam Han, Yoonho Park, Jeongho Yoon, and Seokhyeong Kang. 2021. Reinforcement learning-based power management policy for mobile device systems. IEEE Transactions on Circuits and Systems I: Regular Papers 68, 10 (2021), 4156–4169.

[25] Linaro 96Boards. 2018. HiKey970. Retrieved from https://96boards.org/product/hkey970/. Accessed 2023-02-21.

[26] Di Liu, Shi-Gui Yang, Zhenli He, Mingxiong Zhao, and Weichen Liu. 2021. CARTAD: Compiler-assisted reinforcement learning for thermal-aware task scheduling and DVFS on multicore. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 41, 6 (2021), 1813–1826.

[27] Shiting Lu, Russell Tessier, and Wayne Burleson. 2015. Reinforcement learning for thermal-aware many-core task allocation. In Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI). 379–384.

[28] Sumit K. Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. 2019. Dynamic management of heterogeneous mobile platforms via imitation learning. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27, 12 (2019), 2842–2854. DOI: https://doi.org/10.1109/TVLSI.2019.2926106

[29] Vijayekrishnan Narayanan and Yuan Xie. 2006. Reliability concerns in embedded system designs. Computer 39, 1 (2006), 118–120.

[30] Anuj Pathania, Heba Khdr, Muhammad Shafique, Tulika Mitra, and Jörg Henkel. 2018. QoS-aware stochastic power management for many-cores. In Proceedings of the Design Automation Conference (DAC).

[31] Mihai Pricopi, Thamirimalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Power-performance modeling on asymmetric many-cores. In Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES). IEEE.

[32] Martin Rapp, Hussam Amrouch, Yibo Lin, Bei Yu, David Pan, Marilyn Wolf, and Jörg Henkel. 2021. MLCAD: A survey of research in machine learning for CAD. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 41, 10 (2021), 3162–3181.

[33] Martin Rapp, Nikita Krohmer, Heba Khdr, and Jörg Henkel. 2022. NPU-accelerated imitation learning for thermal and QoS-aware optimization of heterogeneous multi-cores. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.

[34] Martin Rapp, Mohammed Bakr Sikal, Heba Khdr, and Jörg Henkel. 2021. SmartBoost: Lightweight ML-driven boosting for thermally-constrained many-core processors. In Proceedings of the Design Automation Conference (DAC).

[35] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS).

[36] Bagher Salami, Hamid Noori, and Mahmoud Naghibzadeh. 2020. Fairness-aware energy efficient scheduling on heterogeneous multi-core processors. IEEE Transactions on Computers 70, 1 (2020), 72–82.

[37] Anderson L. Sartor, Anish Krishnakumar, Samet E. Arda, Umit Y. Ogras, and Radu Marculescu. 2020. HiLate: Hierarchical and lightweight imitation learning for power management of embedded SoCs. IEEE Computer Architecture Letters 19, 1 (2020), 63–67.

[38] Amit Kumar Singh, Somdip Dey, Klaus McDonald-Maier, Karunakar Reddy Basireddy, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2020. Dynamic energy and thermal management of multi-core mobile platforms: A survey. IEEE Design and Test 37, 5 (2020), 25–33. DOI: https://doi.org/10.1109/MDAT.2020.2982629

[39] Stavros Tzilis, Pedro Trancoso, and Ioannis Sourdis. 2019. Energy-efficient runtime management of heterogeneous multicore using online projection. ACM Transactions on Architecture and Code Optimization 15, 4 (2019), 1–26.

[40] Evangelos Vasilakis, Ioannis Sourdis, Vassilis Papaefstathiou, Antonis Psathakis, and Manolis G. H. Katevenis. 2017. Modeling energy-performance tradeoffs in ARM big, little architectures. In Proceedings of the 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE.

[41] Shi-Gui Yang, Yuan-Yuan Wang, Di Liu, Xu Jiang, Hui Fang, Yu Yang, and Mingxiong Zhao. 2019. ReLeTa: Reinforcement learning for thermal-aware task allocation on multicore. arXiv:1912.00189. Retrieved from https://arxiv.org/abs/1912.00189

[42] Tomofumi Yuki and Louis-Noël Pouchet. 2015. Polybench 4.0.

Received 21 February 2023; revised 5 July 2023; accepted 15 September 2023

ACM Transactions on Design Automation of Electronic Systems, Vol. 29, No. 1, Article 16. Pub. date: November 2023.