ML-Based Thermal and Cache Contention Alleviation on Clustered Manycores With 3-D HBM

Mohammed Bakr Sikal, Heba Khdr, Lokesh Siddhu, and Jörg Henkel, *Fellow, IEEE*

***Abstract*—Enabled by the recent advancements in 2.5D/3-D integration and packaging, the integration of clustered many-core processors with high-bandwidth memory (HBM) is gaining prominence to satisfy the increasing memory bandwidth demands. Although this integration can offer significant performance gains, it is still limited by cache contention in the final-level cache on the clusters and by the thermal issues in the 3-D HBM. While the existing state-of-the-art resource management techniques have tackled these issues in isolation, we argue that the cache contention and the temperature of both the manycore and the HBM must be considered jointly to harness the full performance potential of such modern architectures. To cover this gap in the literature, we present *MTCM*, the first resource management technique that considers the cache contention in maximizing the system performance, while maintaining the thermal safety across both the manycore and the HBM stack. Enabled by our accurate, yet lightweight, neural network models, our proposed task migration and dynamic voltage and frequency scaling policies can accurately predict the impact of runtime decisions on the performance and temperature of both the subsystems. Our extensive evaluation experiments reveal a significant performance improvement over existing state of the art by up to 1×, while maintaining thermal safety of both the manycore and the HBM.**

***Index Terms*—3-D high-bandwidth memory (HBM), cache contention alleviation, clustered many-core processor, machine learning models, neural networks (NNs), smart resource management, thermal-aware management.**

I. Introduction

THE GROWING demand for higher performance in computing systems has highlighted the importance of improving memory bandwidth, necessitating structuring changes across the memory hierarchy. One major breakthrough in this landscape is the introduction of high-bandwidth memory (HBM). With its 3D-stacked architecture and multiple channels, HBM significantly boosts memory bandwidth, enhances power efficiency and lowers latency, meeting the demands of memory-intensive tasks. The recent advancements in 2.5D/3-D packaging technologies have also facilitated the integration of HBM with commercial clustered manycore processors [1], where multiple cores are grouped into clusters and share resources like the last-level cache (LLC). A simplified high-level view of this architecture is shown in Fig. 1. With such integration, the multiple memory channels of the HBM can be leveraged for the parallel data access by the memory controllers on the different clusters, thereby reducing memory bottlenecks and improving the system performance. However, this integration faces some challenges, stemming from the inherent limitations of both the subsystems, i.e., the clustered manycore and the HBM.

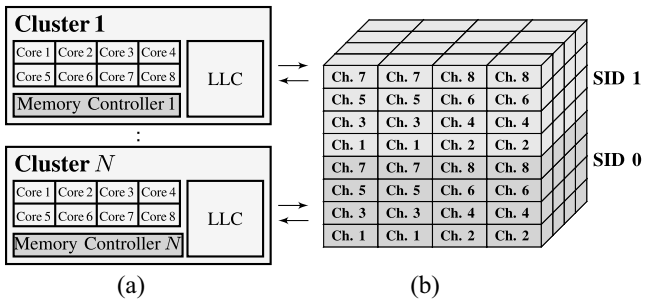

Fig. 1. Our novel technique targets modern system architectures [1] with HBM as the main memory of a clustered manycore to address LLC contention and temperature problems in both subsystems. (a) Clustered manycore. (b) HBM main memory.

On the one hand, due to the high thermal density of HBM, parallel accesses to the thermally coupled memory layers can elevate the temperature of the stack to unsafe margins [2]. Such elevated temperatures trigger the dynamic thermal management (DTM) unit, which transitions the impacted memory channels to a low-power state [3] until the thermal violation is recovered, causing long memory request stalls, thus degrading the overall system performance. To alleviate these thermal problems, different solutions have been explored in the literature, ranging from the architecture-level embedded cooling mechanisms [2] to the system-level thermal management techniques [3], [4].

On the other hand, concurrent execution of applications on the same cluster on a clustered manycore can lead to two types of interference problems. First, contention for the limited available cache space on a cluster can slowdown the execution time of the running applications. Second, the heat transfer between the active cores can raise the temperature of the chip, activating the thermal control circuitry (TCC) [5], which in turn throttles the voltage/frequency (VF) levels across the clusters to cool down the cores. State-of-the-art system-level resource management techniques have addressed both the types of interference jointly and in isolation by means of application

Manuscript received 31 July 2024; accepted 1 August 2024. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Mohammed Bakr Sikal.)*

The authors are with the Chair for Embedded Systems, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: bakr.sikal@kit.edu; heba.khdr@kit.edu; lokesh.siddhu@kit.edu; henkel@kit.edu).

Digital Object Identifier 10.1109/TCAD.2024.3438998

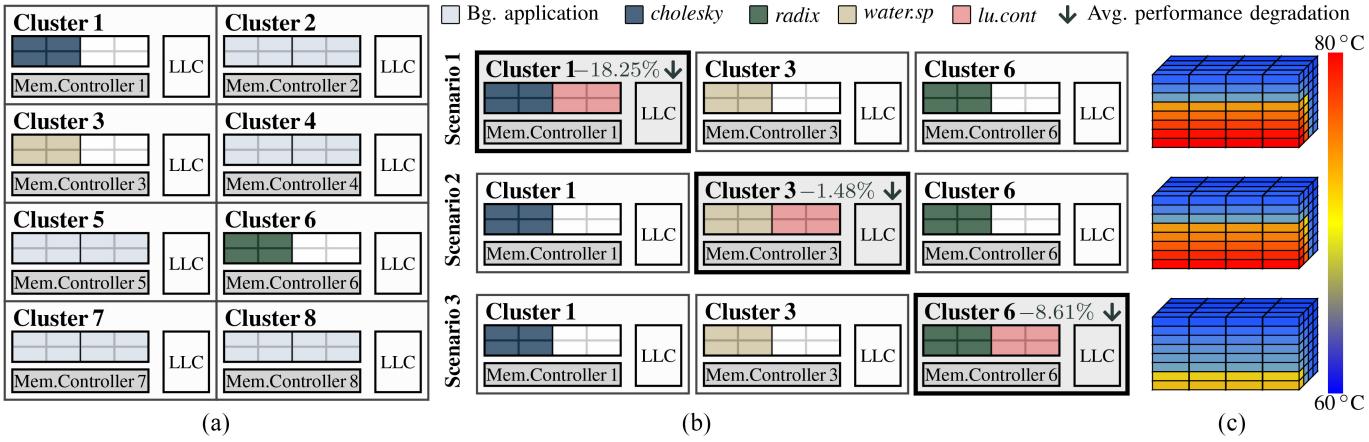

Fig. 2. Although selecting the application-to-cluster mapping in Scenario 2 leads to higher performance gains compared to the other scenarios due to low cache contention on the cluster, it leads to a temperature violation on the HBM. Overlooking such implications at runtime would trigger the DTM, thus suppressing the intended performance gains. (a) Initial state. (b) Application-to-cluster mapping scenarios. (c) HBM heatmaps.

mapping, application migration, and dynamic voltage and frequency scaling (DVFS) [6], [7], [8], [9], [10], [11].

Nonetheless, the limitations of both the clustered manycore and the HBM have only been addressed individually in the literature. As demonstrated in the following motivational example, the integration of HBM and clustered manycores introduces new intricate tradeoffs between the thermal and cache interferences in both the susbsystems, whereby the performance potentials might be missed if the contention and thermal impacts on the two subsystems are overlooked.

A. Motivational Example

We simulate a 64-core processor (detailed in Section VI), organized into eight clusters and using an eight-channel HBM as main memory, similar to the HBM2E in [12]. The memory controller in each cluster is configured to use one of the eight channels of the HBM, following a cluster-to-channel mapping, where the cluster n uses the channel n similar to [4] and [13] as shown in Fig. 2(a). We consider 80 °C as the temperature constraint for the manycore and HBM. The default mechanisms that react to the thermal violations, i.e., TCC on the manycore and the low-power state DTM on the HBM are both disabled to highlight the impact of resource management decisions on the temperature of both the subsystems. Initially, clusters 2, 4, 5, 7, and 8 are fully occupied by compute-intensive background tasks. These clusters are running at the minimum 1.0 GHz VF level and do not issue any memory accesses throughout this experiment. Clusters 1, 3, and 6 are hosting applications with different characteristics from the SPLASH-2 [14] benchmark suite: *cholesky*, *water.sp*, and *radix*, respectively. For ease of analysis, clusters 1, 3, and 6 are set to run at a medium fixed VF level, i.e., 2.8 GHz, to ensure that no thermal violation occurs on the manycore side. With a fourth application *lu.cont* also requiring four cores to execute, three application-to-cluster mapping scenarios are possible, in each, *lu.cont* will run in parallel on one cluster with another application as shown in Fig. 2(b). We study the performance and thermal implications of the three mapping scenarios.

In scenario 1, *lu.cont* is mapped to cluster 1 to co-execute with *cholesky*. The resulting average performance degradation for both the applications compared to their corresponding execution times when running alone on a cluster reaches a significant slowdown of 18.25%. In addition to this performance degradation, we also observe thermal violations on multiple banks, reaching up to 85 °C on the channel 1. This is due to the intense memory accesses issued by *cholesky* and *lu.cont* throughout their execution, which elevates the temperature of multiple memory banks in both their assigned channel 1 and directly adjacent channels 2, 3, and 4, as shown in Fig. 2(c). In scenario 2, *lu.cont* is mapped to cluster 3 to coexecute with *water.sp*. This application-to-cluster mapping results in better overall system performance compared to the previous scenario with a minimal slowdown of −1.48%, indicating minimal contention between *water.sp* and *lu.cont*. While *water.sp* barely issues any memory accesses, *lu.cont* heavily utilizes channel 3 throughout its execution. Combined with the intense accesses from *cholesky* on channel 1, and given that channels 1 and 3 are vertically adjacent, temperature on many memory banks on the bottom-most layers on the HBM stack again exceeds the 80 °C threshold. In scenario 3, we map *lu.cont* to cluster 6 to coexecute with *radix*, this time far from the heavily accessed channel 1. As expected, temperature throughout the execution remains below the threshold across the HBM stack, as the intense memory accesses from *cholesky* and *lu.cont* are now distributed across the less thermally coupled channels, i.e., channels 1 and 6. However, an average performance degradation of 8.61% is observed for both the applications.

Although scenario 2 resulted in the best overall system performance with minimal cache contention, it is not a thermally safe application-to-cluster mapping option. Applying it at runtime will trigger the DTM, transitioning all the affected memory channels to a low-power state, eventually suppressing the intended performance benefits of such mapping. Therefore, given the observed slowdowns and temperatures, scenario 3 remains the mapping option that maximized performance under the considered temperature constraints. At runtime, as the running applications change their execution phases, the resulting cache contention and temperature effects on both the manycore and the HBM change accordingly. Therefore, a system-level resource management should also adapt to such

changes in order to harness the full performance potentials of the system. In this work, we adapt to such runtime changes by dynamically adjusting both the application-to-cluster mapping through task migration and the VF levels of the clusters through DVFS.

B. Challenges and Contributions

As demonstrated in the previous motivational example, maximizing performance requires a comprehensive analysis of the resulting cache contention-induced slowdowns in all the possible application-to-cluster mappings, along with their impacts on the temperatures of both the manycore and the HBM. At runtime, a similar analysis should be conducted to evaluate the possible application-to-cluster mappings before applying a migration. Therefore, there is a need to predict the postmigration system performance, as well as the manycore and HBM temperatures, which is challenging for the following reasons.

Performance Prediction: Estimating the migration impact on the performance considering contention is a complex problem. Contention is affected by both the system's hardware characteristics, e.g., cache capacity and associativity, memory bandwidth, etc., and by the characteristics of the concurrently running applications on such system, i.e., cache accesses, misses, etc. This contention behavior is complex and hard to be modeled neither analytically nor via profiling of applications at design time as established in [7]. We tackle this challenge by training a lightweight neural network (NN) model that is able to learn the relevant cache contention indicators, e.g., cache accesses, cache misses, memory accesses, etc. from a limited training data generated as design time, and generalize to unseen contention scenarios at runtime.

Predicting Temperature: While predicting the manycore temperature for different application-to-cluster mappings is straightforward, thanks to the well-known RC thermal model [15], predicting the temperature impact on the HBM is a challenging problem. The memory controllers on the source and target clusters involved in a task migration are configured to use different memory channels. Therefore, after the migration is performed, the memory accesses of the migrated application will be serviced by the target memory channel. Consequently, the power consumption of the banks spanned by the source and target channels will be impacted, resulting in a new per-bank power distribution map on the HBM. Given such a power map, a thermal model can be used to accurately predict the postmigration temperature of the HBM. However, as the two clusters could be set to different VF levels, an application would show a different memory access behavior once migrated to the target cluster, thereby resulting in different bank power consumptions. Thus, to accurately construct the postmigration per-bank power distribution map, there is a need for a model to scale the number of memory accesses of the migrated application. As detailed in Section IV, we address this challenge by training a second lightweight NN that is able to accurately scale the memory accesses of applications to the new target VF level.

Runtime Overhead of Temperature Prediction: To ensure the thermal safety, a system-level resource management technique must operate within brief epochs, as to adapt to runtime dynamics and the execution phases of running applications. For instance, the Linux governor employs DVFS within epochs as short as 1 ms and initiates migrations within 10 ms-long epochs [16]. Given these tight timing constraints, the traditional finite difference methods, e.g., Hotspot [15] or 3D-ICE [17], or Green's function-based approaches, e.g., 3DSim [18], are not viable for runtime use due to their extensive computational overhead, i.e., prediction times in tens to thousands of milliseconds [19]. In contrast, machine learning (ML)-based techniques have been demonstrated to provide the necessary accuracy with significantly lower computational costs, i.e., microsecond-level predictions. However, no ML-based thermal simulators are available as open-source solutions [19]. To overcome this limitation, we propose training two highly accurate, yet lightweight, NN-based thermal models, one for the manycore and another for the HBM. These models are designed to efficiently and accurately predict the steady-state temperature of their respective subsystems at runtime with an inference time of a few microseconds.

Enabled by our NN models, we build a novel resource management technique, *MTCM*, that combines both the task migration and DVFS to maximize performance considering contention, while enforcing thermal constraints of both the clustered manycore and the HBM. *MTCM* periodically attempts to achieve a better cache contention balance across the system by migrating applications between the clusters. Before applying a migration, *MTCM* evaluates the postmigration potential performance gains and thermal impacts on both the manycore and the HBM, then only proceeds if the migration would indeed improve performance without causing a temperature violation on any of the two subsystems. In addition, as applications change characteristics throughout their execution, *MTCM*'s DVFS policy is invoked periodically to boost the clusters when the thermal headroom is available, and to throttle the clusters to avoid the thermal violations on both the manycore and the HBM due to sudden surges in power consumption.

Our novel contributions are as follows.

1) We train two lightweight NN models to accurately predict the impact of task migration on the system performance considering cache contention in clusters, and the impact of changing the VF levels on the HBM access behavior of applications.
2) We train two lightweight NN-based thermal models, engineered to accurately predict the temperature of the clustered manycore and the HBM within the stringent time constraints at runtime.
3) Enabled by our fast and accurate NN models, we present *MTCM*, the first resource management technique that jointly mitigates cache contention while enforcing thermal safety on the systems with an integrated clustered manycore and HBM by means of the task migration and DVFS.

The remainder of this article is organized as follows. Section II covers the state-of-the-art research related to our work. After formally formulating the problem in Section III. Section IV introduces our approach to data generation and training of our four NN models. Section V then describes

the details of our proposed *MTCM*. Section VI presents the evaluation results of *MTCM* compared to the two state-of-the-art techniques in terms of the system performance, core and HBM temperature, and thermal efficiency. Section VI-E discusses the runtime overhead of our *MTCM*. Section VII concludes this article.

II. Related Work

The related works to our proposed *MTCM* are techniques having addressed the inherent problems of clustered manycores and 3-D HBM. On the manycore side, many researchers have addressed the thermal and cache interference problems. Kim et al. [20] employed both the task migration and cluster-level DVFS to maximize performance of a thermally constrained clustered heterogeneous multicore processor. The work in [21] proposed ML-based DVFS technique that considers cache contention at the level of the cluster, in order to satisfy the performance constraints of running applications. Mishra et al. [22] built ML-based scheduling technique that maps applications to clusters with the minimum predicted contention. A recent technique [6] used intercluster task migration to balance the thermal interference across the chip. The optimized thermal headroom is then exploited by a cluster-level DVFS policy to boost the performance of running applications. Two other works [23], [24] have used reinforcement learning-based DVFS and task mapping to minimize temperature of a multicore under performance constraints. A more recent work [25] has employed imitation learning-based task migration to minimize the temperature of a multicore under application latency constraints. Nonetheless, none of these techniques has considered the temperature of main memory, as they have targeted the systems with the traditional 2-D DRAMs, which are not prone to thermal issues. Porting these techniques to the systems where HBM is the main memory, can lead to the suboptimal resource management decisions as demonstrated in Section I-A.

With the increasing adoption of HBM in modern architectures, more recent works in the literature have addressed its main recognized bottleneck: temperature. Researchers in [2] and [26] propose advanced embedded cooling mechanisms and architecture-level optimization schemes to address the problem of high temperature in the HBM stack. Lo et al. [27] considered memory-channel temperatures within their proposed memory-page allocation technique as to avoid the formation of thermal hotspots on the heavily accessed memory banks. Shen et al. [3] proposed a reinforcement learning-based technique combining DVFS and low-power states to maximize the performance under the temperature constraints of both the processor and the HBM. Another recent work *NeuroMap* [4] proposed a technique to maximize performance on a system with a manycore processor and HBM. By assigning memory channels to core groups, their technique dynamically adjusts the mapping of applications to different core groups to manage the temperature of the HBM. Additionally, DVFS and low-power states are used to further maximize performance while enforcing the thermal safety. Though the temperature of the manycore is overlooked in their approach, *NeuroMap* [4] remains the closest in the literature to our proposed technique.

In summary, a gap exists in the literature, as no resource management technique has jointly considered the cache contention problem within the performance maximization on the modern clustered manycores with HBM, while enforcing a thermally safe operation of both the subsystems. As highlighted in Section I-A, cache contention and the temperature of both the manycore and the HBM must be considered jointly to harness the full performance potential of such modern architectures.

III. Problem Formulation

We target a clustered manycore with the M cores and K clusters, each hosting M/K cores and a shared LLC. Cores in the cluster k are operated at the same VF level f_k by the cluster-level DVFS. Matrix $\mathbf{G} = [g_{i,k}]_{M\times K}$ indicates core-to-cluster mapping, where $g_{i,k} = 1$ if the core i is in the cluster k. The HBM main memory comprises the L layers with B banks each and C channels, each channel ch spanning banks B_{ch}. Each cluster k accesses memory through a dedicated controller MC_k configured to access a specific channel ch [13]. Cores in the cluster k issue memory requests to MC_k. Core temperatures, $[t_i]_M$, are estimated using a thermal model (TM) TM_{cores}, based on the per-core power consumption map P_{cores}. Similarly, the thermal model of the HBM TM_{hbm} predicts per-bank temperatures $[t_i]_{B\times L}$ given the per-bank power map P_{banks}. Per-channel temperature T_{ch} is the maximum of all banks in channel ch, aligning with the HBM JEDEC standards [28]. The same temperature threshold T_{thresh} is considered for both the subsystems. In our open system [29], A multithreaded applications with unknown arrival times are mapped to the clusters, with mappings defined by $Q=[q_{k,a}]_{K\times A}$. Each application a runs h_a parallel threads, requiring h_a cores as per the one-thread-per-core model [30]. Thread-core mappings are given by $V=[v_{i,a}]_{M\times A}$. Our goal is to optimize response time and manage cache contention while ensuring thermal safety for cores and HBM. Our proposed *MTCM* addresses this problem by training the NN models at design time (Section IV), which empower our runtime task migration and DVFS policies at runtime (Section V).

IV. NN-Based Models

Before migrating an application at runtime, there is a need to predict the postmigration impact on the system performance, in addition to the resulting manycore and the HBM temperatures, which is challenging as demonstrated in Sections I-A and I-B. We overcome these challenges by employing the NN models that are trained at design time and used at runtime to predict these unknown postmigration metrics. In the following sections, we first present our methodology to train NN_p and NN_m, our proposed performance and memory access prediction models. Second, we introduce our NN-based thermal models, NN_{cores} and NN_{hbm} for predicting the temperature of the clustered manycore and the HBM, respectively.

A. Performance and Memory Access Prediction

Fig. 3 summarizes the training methodology of both our performance and memory access models NN_p and NN_m, which consists of the following key steps.

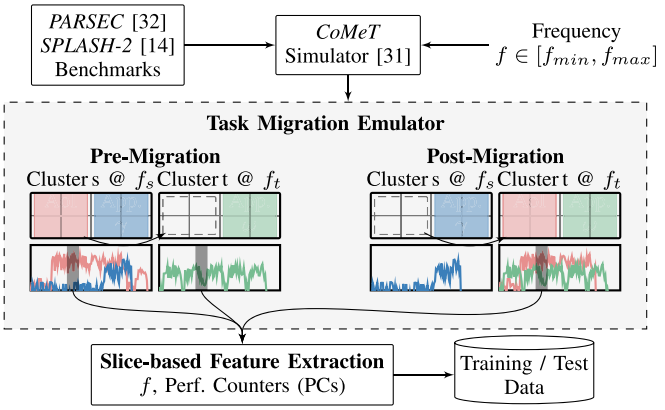

Fig. 3. Our slice-based training data generation methodology uses a fine-grained analysis of execution traces of applications to extract the characteristics of their execution phases, relevant to the prediction of postmigration performance and memory access behavior.

1) Isolating the Cache Contention Effects: First, we run an extensive set of simulations of multithreaded benchmark applications at all the supported VF levels. During each simulation, per-thread performance monitoring counters (PMC)s are collected periodically, i.e., each 1 ms, throughout the execution, including multiple performance-, cache- and main memory-relevant metrics, e.g., instructions per second (IPS), IPC, L1/L2/L3 cache accesses/misses, memory reads and writes, etc. These contention-free alone execution traces serve as a reference to the best performance achievable by applications at a specific VF level.

Second, we run all the combinations of applications App_a and App_O, i.e., shared executions, on the same cluster, each having four threads. By restricting the combinations to only two applications, any observed degradation in the execution time of App_a compared to its alone execution time, is directly due to the parallel execution with App_O and vice-versa. Moreover, following the one-thread-per-core model [30], these combinations fully occupy the cores on a cluster on our target platform, i.e., eight cores per cluster, representing an upper bound of occupation representativeness.

2) Emulating Task Migration Scenarios: The obtained alone and combined simulation traces are passed to a *task migration emulator*. For each application App_a, our emulator fetches all the simulation traces, where App_a runs in parallel on a cluster with another application App_O. These traces are used to construct premigration and postmigration scenarios, where the application App_a initially runs on a source cluster s with an application App_O at the VF level f_s, then migrated to a target cluster t to run with another application App'_O at f_t. For each constructed scenario, the collected 1 ms periodic PMCs from the premigration and postmigration simulations are saved, including their periodic average IPS and the VF levels f_s and f_t. As tasks can be migrated to empty clusters, our emulator also constructs such scenarios, where App_a runs alone on a cluster after migration, achieving its peak alone execution performance at f_t.

3) Extracting Representative Samples: The data from the previous phase can be used to train the NN model to predict any of the collected postmigration PMCs of App_a, e.g., memory accesses or IPS. These collected traces assume that the applications are mapped to the same cluster throughout their whole execution. At runtime, however, an application could start its execution on one cluster, then be migrated to a new one in the following migration epoch, since *MTCM*'s migration policy will be invoked periodically every 10 ms. Therefore, there is a need to predict the postmigration PMCs of App_a over only 10 ms-long windows, not over the whole shared execution of applications. To achieve this goal, we perform *time-based slicing* of execution traces from the task migration scenarios of the previous phase at a 1 ms granularity. Iteratively, for each extracted 10 ms-long execution slice in a premigration shared execution trace $\{\text{App}_a, \text{App}_O\}$, the corresponding slice in each thread in the alone execution of App_a is also extracted. Similarly, for each slice in the postmigration shared execution trace of $\{\text{App}_a, \text{App}'_O\}$, the corresponding slice in each thread in the alone execution of App'_O is also extracted. These per-thread slices are used to construct aggregated slices, representing a task migration scenario at a 10 ms granularity. It is important to note that this data engineering approach would teach the model that App_a is running in parallel with multiple threads rather than a single application. It is important to note that threads will not be active throughout the whole execution of an application. Therefore, this step of the process will also generate slices that represent the lower bound of core occupation representativeness, where only one thread is active, i.e., only one core is occupied. As such, at runtime, when App_a is running in parallel with any number of multithreaded applications on the same cluster and any number of active threads, their corresponding aggregated per-thread data will be used to construct the slices of App_O and App'_O. With the obtained dataset of approximately two million rows, our NN_p and NN_m models can be trained to learn that, given the premigration PMCs of App_a and App_O at f_s, and given the alone characteristics of App'_O at f_t, predict the postmigration IPS and memory accesses of App_a at f_t, respectively.

4) Selecting Features and Model Topologies: This final phase aims at finding the minimum set of features and the smallest topology for each model, such that a high prediction accuracy is achieved under a reasonable runtime overhead. The generated dataset in the previous phase is used to train both the performance NN_p and the memory access NN_m models. using *Lasso regression* and the *Pearson product-moment correlation*, we first identify the importance of the collected features to the prediction label in each model. After progressively eliminating the least-important features, e.g., L1-I and L1-D cache accesses and misses, etc., we split the dataset randomly to 75% training and 25% test sets. Guided by an initial exploration using the *KerasTuner* library, we conduct a design-space search for smaller topologies, i.e., numbers of layers and number of neurons per layer, and hyperparameters, e.g., batch size, regularization, dropout factors, etc. Given the mean-absolute-percentage error (MAPE) score and the inference overhead of each obtained model on our target system, the search space is either enlarged or narrowed-down, as to balance the accuracy-overhead tradeoff. Fig. 4 shows the different topologies explored for each model and

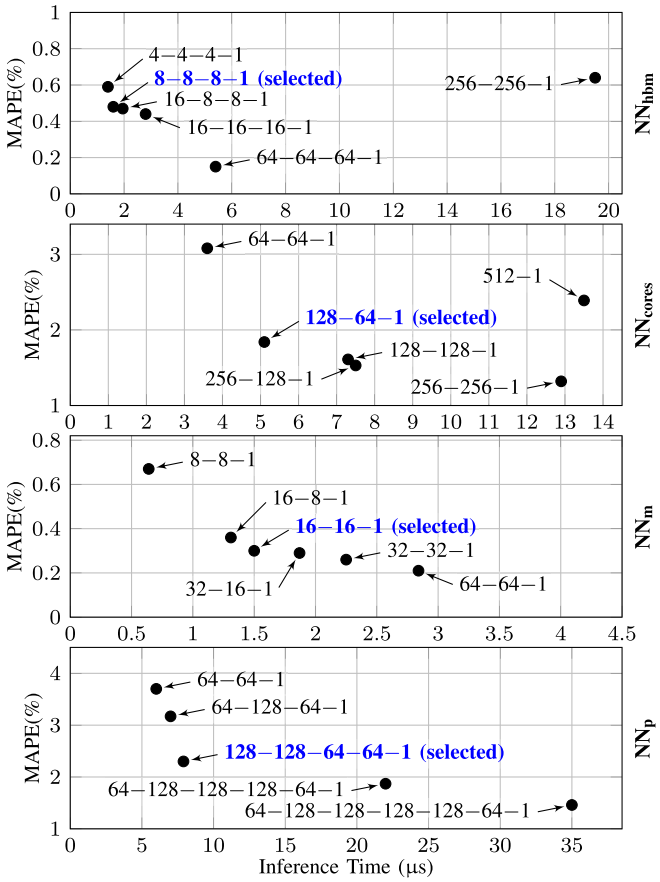

Fig. 4. Initial search using KerasTuner explores various topologies with different layers and neurons per layer. After excluding nonconverging models, a Pareto-front of options is identified. Ultimately, we select the model topology that balances prediction error and inference time on our target platform, thereby enabling our runtime policies to maximize the system performance under thermal constraints with a very negligible runtime overhead.

their corresponding MAPE scores and inference times on our target platform. Since, the data points in the explored space are *Pareto-optimal*, we select the topologies that satisfy our strict timing requirements at runtime as further discussed in Section VI-E. Fig. 4 also shows model topologies with lower MAPE scores than the selected topology, to highlight that our models could deliver a higher prediction accuracy when the strict runtime timing requirements are relaxed.

Ultimately, the selected NN_p has four hidden dense layers with 128, 128, 64, and 64 neurons, in addition to an output layer of one neuron. The selected NN_m has two hidden dense layers with 16 neurons each, and one output layer of one neuron. The inference time on our target platform is 7.9 μs, and 2 μs for NN_p and NN_m, respectively. The achieved MAPE scores are 2.3% and 0.3% for NN_p and NN_m, respectively. These models' marginal prediction errors are incorporated into *MTCM* as detailed in Section V.

B. Temperature Prediction for the Manycore and HBM

As formulated in Section III, the thermal safety must be enforced at runtime, by only applying decisions that would not violate the thermal constraints of either the manycore or the HBM. This requires fast and accurate thermal models for each of the two subsystems, that are able to deliver accurate temperature predictions within the tight time constraints at runtime. To achieve this goal, the training data for our thermal models shall be representative of the runtime scenarios, when the cores on the manycore and the banks on the HBM will exhibit varying levels of power consumption and will be utilized at varying intensities, i.e., core occupation, memory accesses, etc. However, building upon the simulation framework outlined in Section IV-A, with a targeted architecture of a clustered manycore processor consisting of 64 cores and an integrated HBM with eight channels, collectively holding 128 banks (as per the specifications detailed in [12]), the required simulation combinations to reach the intended state of representativeness lead to an exponential explosion of the design space, making this approach not viable and inefficient. Instead, to train our NN-based thermal models, we propose to use synthetically generated training data obtained using the following steps.

1) Generating the Base Datasets: Building upon the simulation framework outlined in Section IV-A, we initiate our study by conducting the simulation experiments, where single multithreaded applications are mapped to one cluster at a time. Each experiment is conducted across all the available VF levels to ensure comprehensive data coverage. During these simulations, we collect per-core and per-bank power consumption data, alongside steady-state temperature measurements from the hotspot simulator [15]. This data is collected with a fine granularity of 1 ms, ensuring a detailed temporal resolution. We then proceed to construct two base datasets: one for training NN_{cores} and the other for NN_{hbm}.

2) Synthetic Data Augmentation: The base datasets are comprised of power and temperature maps generated from the single application executions at varying VF levels. However, these datasets do not adequately represent runtime scenarios where multiple applications typically run concurrently, leading to diverse utilization patterns of cores and memory banks. To enhance the representativeness of these datasets, we generate synthetic periodic power traces for both the manycore processor and the HBM. By mixing power values from the individual executions and distributing them randomly across a larger set of cores and memory banks, we construct synthetic multiapplication power maps. Given the exponential potential for the trace generation, we cap the number combinations at two million traces for each subsystem, the manycore and the HBM. These synthetic traces combined with the original data, yield two augmented datasets that simulate a range of scenarios, including both the single and parallel application executions with diverse VF level mixes manycore occupation and HBM utilization. Nevertheless, the random generation of power distributions leads to an imbalance in the dataset. Specifically, some cores and banks are underrepresented, which might inadvertently suggest to the model a lower likelihood of their occupancy. Furthermore, the variance in power consumption ranges could mislead the model to infer that certain cores and banks are restricted to specific power consumption levels. To correct this imbalance and improve the training data representativeness, we employ *stratified sampling* with designated power bins. This approach ensures balanced representation of each core and bank in terms of appearance

TABLE I
OUR NN MODELS ARE LIGHTWEIGHT AND HIGHLY ACCURATE, ENABLING OUR MIGRATION AND DVFS POLICIES TO EFFECTIVELY ACHIEVE THE GOAL OF PERFORMANCE MAXIMIZATION UNDER TEMPERATURE CONSTRAINTS OF BOTH THE CLUSTERED MANYCORE AND HBM

Model	Topology	MAPE (%)	Inference Time (µs)	Memory Footprint (KB)
$\mathbf{NN_p}$	4 hidden layers (128, 128, 64, 64 neurons)	2.3	7.9	134
$\mathbf{NN_m}$	2 hidden layers (16 neurons each)	0.3	1.5	5
$\mathbf{NN_{cores}}$	2 hidden layers (128, 64 neurons)	1.8	5.1	67
$\mathbf{NN_{hbm}}$	3 hidden layers (8 neurons each)	0.4	1.6	5

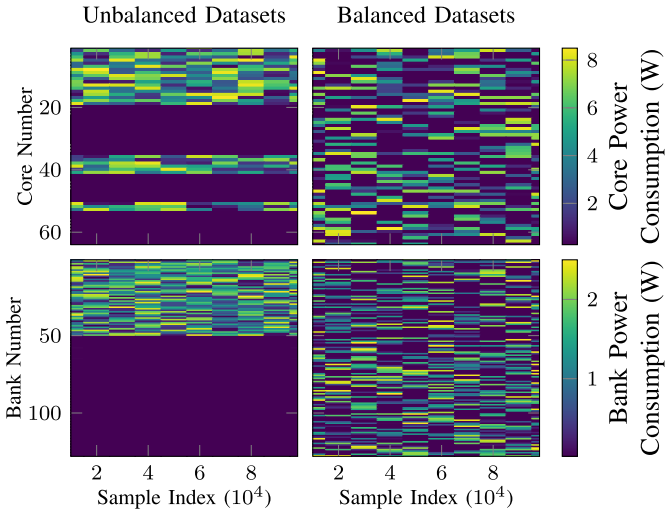

Fig. 5. Examples of skewed distributions of power values in the training dataset with dominant activity in few cores or memory banks, which can limit our thermal models' ability to generalize to unseen power consumption patterns at runtime. Our synthetic data generation approach in contrast ensures a well-balanced distribution across all the cores and memory banks both in terms of frequency and power value ranges.

frequency and power consumption levels as illustrated in Fig. 5. It is important to note that, at this stage, the rows in the training data cannot be traced back to any individual application, as this information is diluted following the mixing, shuffling, and stratified sampling, ensuring that the models will not be trained with a bias toward specific application mixes.

The final step involves using the hotspot simulator [15] to predict the steady-state temperature of each core and bank based on the power traces, serving as ground truth labels for our models. The datasets are then divided, allocating 75% for model training and 25% for testing.

3) Single-Label Model Optimization: Aligned with the goal of ensuring thermal safety, essentially predicting potential thermal violations based on the power traces, we adjust our training data, so that the models only predict the maximum temperature across either the cores or memory banks, instead of all the temperatures. This approach is not only consistent with thermal safety objectives but also serves to minimize the computational overhead during runtime. Accordingly, the datasets are refined to include just the maximum temperature per set, namely the hottest core for $\mathrm{NN_{cores}}$ and the hottest bank for $\mathrm{NN_{hbm}}$. Similar to the neural architecture search for $\mathrm{NN_p}$ and $\mathrm{NN_m}$, and as highlighted on Fig. 4, we select from the Pareto front the topologies that satisfy our strict timing

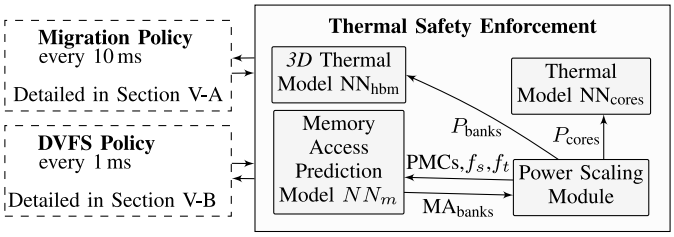

Fig. 6. Our runtime thermal safety enforcement strategy enables our proposed policies to apply thermally safe resource management decisions, thereby better harnessing the performance potential of the system.

requirements at runtime as further discussed in Section VI-E. The final $\mathrm{NN_{cores}}$ model comprises the two hidden layers with 128 and 64 neurons, and the $\mathrm{NN_{hbm}}$ model is built with the three hidden layers of eight neurons each, all employing the ReLU activation. As detailed in Table I, this optimization results in significant reductions in the memory footprint of the models as well as their corresponding inference time on our target platform.

V. ML-BASED RESOURCE MANAGEMENT FOR CLUSTERED MANYCORES WITH HBM

Our proposed *MTCM* periodically executes two resource management policies at runtime: task migration and cluster-level DVFS, as shown in Fig. 6. In the following, both policies are presented.

A. ML-Based Application Migration

Similar to the Linux *migration epoch* for the task migration [16], our *MTCM*'s migration policy executes the following steps each 10 ms.

1) Identifying Applications to Migrate: Applications executing alone on a cluster are expected to achieve their maximum performance at the set VF level and are therefore not considered for migration. On the other hand, any application App_a that is running in parallel with the other applications App_O on a cluster is considered for migration, as it may be suffering performance degradation due to cache contention. Given the parallelism level of App_a, the possible target clusters that have a sufficient number of cores to host it are also identified. Following the migration, App_a would be running with zero or many applications App'_O at f_t on the target cluster. The premigration, i.e., over the past 10 ms period, PMCs of the involved App_a, App_O, and App'_O, their observed IPS values in addition to f_s and f_t are saved for all the running applications that satisfy the aforementioned requirements.

2) Evaluating Performance Potentials: MTCM considers that migrating App_a would only benefit the overall system performance if the postmigration IPS improves compared to the premigration for the applications App_a, App_O, and App'_O. Therefore, for each possible migration identified in the previous step, *MTCM* invokes NN_p to predict the potential postmigration IPS. If an improvement by more than the MAPE of our NN_p is observed, the migration is saved but awaits the validation of the thermal safety enforcement step.

3) Enforcing Thermal Safety: The performance benefits predicted in the previous step may be suppressed if the migration leads to thermal violations on the manycore or on the HBM as demonstrated in Section I-A. To estimate the impact of the migration on the temperature of manycore, the postmigration power distribution map P_{cores} is constructed according to the postmigration application-to-cluster mapping, where App_a and App'_O would run in parallel on the target cluster t at the VF level f_t. The current temperature of all the cores and the postmigration P_{cores} are passed to our NN_{cores} model to predict the new steady-state temperatures of the cores on the manycore. Estimating the impact of the migration on the temperature of the HBM, on the other hand, is more complex. Before the migration, App_a runs on the cluster s and its memory requests are serviced by the channel ch_s assigned to the memory controller MC_s at the VF level f_s. After the migration, App_a will be running on the cluster t and its memory requests will be serviced by the channel ch_t assigned to the memory controller MC_t at the VF level f_t. Therefore, the postmigration P_{banks} power map depends on the postmigration *memory access map* denoted as MA_{banks}. To construct the postmigration MA_{banks} memory access map, we first predict the number of memory accesses App_a would issue at f_t using our NN_m model. Second, we deduct the premigration accesses uniformly from the banks of the channel ch_s in MA_{banks}. Third, we add the predicted postmigration memory accesses uniformly to the banks of the channel ch_t in MA_{banks}. The obtained MA_{banks} now depicts the distribution of memory accesses across the banks of all the HBM channels if App_a is migrated from the clusters s to t. Given the energy per access of the modeled HBM obtained from the state-of-the-art CACTI-3DD simulator [33], P_{banks} can then be constructed, then passed to our NN_{hbm} in order to predict the new per-bank steady-state temperatures. At the end of this step, the migration is considered thermally safe if none of the cores on the manycore and none of the banks on the HBM exceeds the defined T_{thresh} temperature. Finally, *MTCM* performs the thermally safe migration that would result in the highest overall performance improvement.

B. Cluster-Level DVFS

At each DVFS epoch, i.e., 1 ms, our *MTCM*'s cluster-level DVFS policy attempts the following.

1) It estimates steady-state temperatures of both the subsystems using their corresponding thermal model based on their current corresponding power consumption P_{cores} and P_{banks}. If a potential thermal violation is detected, *MTCM* needs to throttle down the clusters with the higher temperatures. To thermally rank the clusters, we compute a *thermal score* given the highest core temperature in each cluster k and the highest bank temperature in its assigned channel ch_k. In decreasing order of thermal score, the clusters are then throttled to the next lower VF level and the new steady-state temperatures of both the subsystems are predicted given the constructed P_{cores} and P_{banks}. Throttling continues until temperatures of both the manycore and the HBM are below T_{thresh}. The DVFS policy then downgrades the VF levels of selected clusters.
2) If no violation is predicted and thermal headroom exists in both the subsystems, *MTCM* safely boosts clusters to the next higher VF level. It is important to note that constructing the HBM's P_{banks} requires the invocation of our NN_m to predict the number of memory accesses at the next potential VF levels.

VI. Experimental Result

A. Setup

We run our experiments using the CoMeT [31] simulator, an integrated toolchain combining the well-known Sniper [35] and Hotspot [15] simulators for performance and thermal simulation, respectively. The target system comprises a clustered 64-core processor and an eight-channel HBM. Each core has private 32 KB L1 instruction and 32 KB L1 data caches, and a private L2 256 KB cache. Each cluster on the manycore groups eight cores, sharing one 8 MB LLC and one memory controller, mapped to one of the eight channels of the HBM. The HBM main memory is modeled based on the 16 GB HBM2E in [12], having a 6.73 ns latency and 9.51 nJ per access energy, obtained using Cacti-3DD [33]. Cluster-level DVFS sets all the cores within a cluster to the same VF level ranging from 1 to 4 GHz with boosting and throttling steps of 200 MHz. The temperature constraint on both the manycore and the HBM is set to 80 °C. A thermal violation on any core of the manycore triggers the DTM, which transitions all the cores to the minimum VF level. A thermal violation on any bank of the HBM triggers the corresponding channel to a low-power state until thermally safe operation is restored. In our experiments, we use applications from the *PARSEC* [32] and *SPLASH-2* [14] benchmark suites, including: *blackscholes*, *bodytrack*, *canneal*, *streamcluster*, *fluidanimate*, *swaptions*, *x264*, *barnes*, *cholesky*, *fft*, *fmm*, *lu.cont*, *lu.ncont*, *radix*, *raytrace*, *water.nsq*, and *water.sp*. These applications cover different application domains., e.g., financial analytics, computer vision, image processing, etc., and therefore have different memory-/compute-intensity characteristics. All the applications are using the large input size and cover a variety of compute and memory intensity characteristics. Each of these applications can be executed at 2, 3, 4, or 5 parallel threads, leading to a total of 68 unique applications. Using these applications, we construct three workloads, each comprising 40 randomly selected multithreaded applications. In addition, we sample the arrival times of applications in each workload from a Poisson distribution at the four arrival rates: 120, 140, 160, and 180 applications per second, and each workload

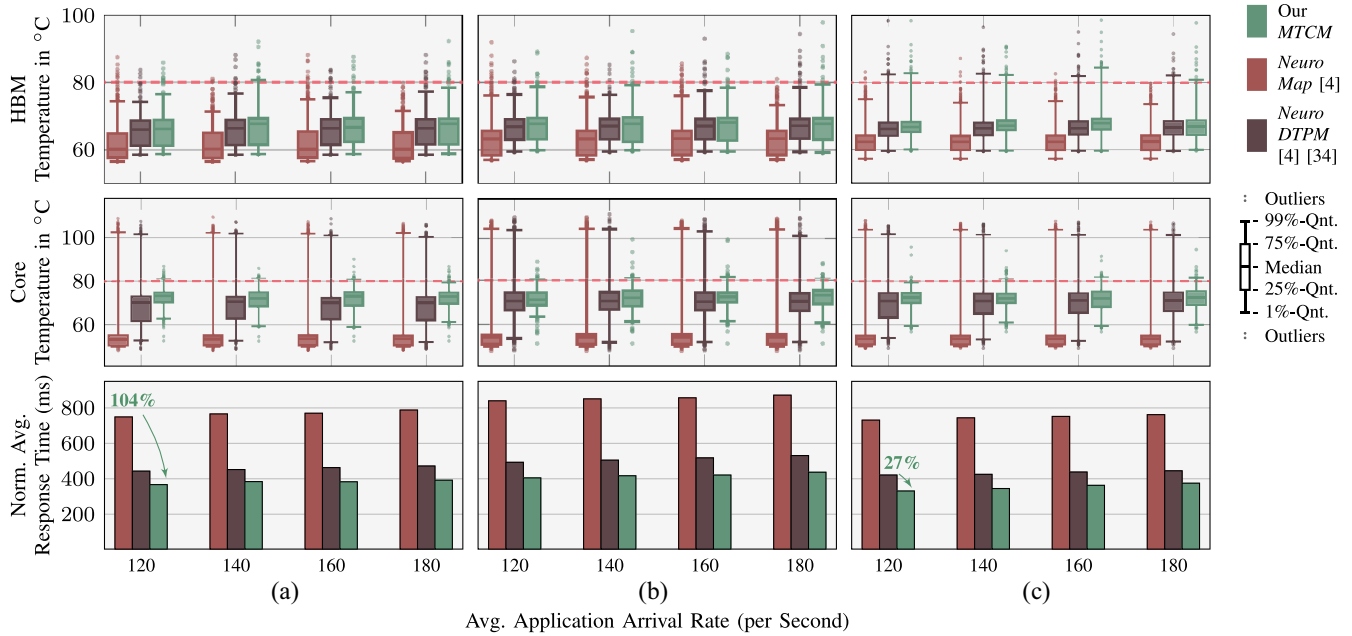

Fig. 7. our *mtcm* demonstrates substantial performance gains by up to 1× for multiple workloads compared to the state-of-the-art comparison techniques, thereby underscoring the importance of jointly considering cache and thermal interference while optimizing performance of modern systems with clustered manycores and **hbm!**. in addition, our *mtcm* successfully operates both the manycore and the **hbm!** closer to the thermal constraint of the system compared to the state-of-the-art comparison techniques, thereby better harnessing the performance potentials of the system. (a) workload 1. (b) workload 2. (c) workload 3.

is executed at each of the four arrival rates. The different arrival rates in combination with the application mixes in each workload lead to different CPU and HBM utilization values, i.e., core and cluster occupation, cache accesses, memory bank accesses, etc. In these experiments, our models are exposed to scenarios that are unseen at training in terms of cache contention behavior, number of applications per cluster, number of threads per application, core occupation, and HBM channel utilization.

B. Comparison Techniques

Our proposed *MTCM* is compared against the following two techniques.

NeuroMap [4]: It is the closest state-of-the-art technique to our *MTCM*, which proposes a resource management policy that uses task migration and DVFS. It aims at maximizing the overall system performance under a memory channel-temperature constraint without considering the temperature of the manycore processor. Their proposed technique first assigns memory channels to groups of cores, a strategy that can be applied on our target platform where the cores are also grouped into clusters and share a memory channel. Then, periodically, *NeuroMap* adjust the application-to-cluster mappings depending on the characteristics of the running applications in a rule-based manner. For instance, if an application is in a memory-intense phase, *NeuroMap* attempts to migrate it to the cluster that is mapped to the HBM channel closest to the cooling system (the upper channels). Similarly, a compute-intense application with very few memory accesses can be mapped to clusters that use channels in the lower end of the HBM stack. When *NeuroMap* fails to find such an ideal application-to-cluster match, it employs DVFS to either throttle down or boost the clusters. However, none of these decisions consider the temperature of the cores.

NeuroDTPM [4], [34]: As highlighted in Section II, none of the state-of-the-art techniques has targeted the goal of performance maximization under temperature constraints of both the manycore and HBM subsystems. Therefore, we further construct *NeuroDTPM*, a second comparison technique that combines *NeuroMap* with *DTPM* [34], a state-of-the-art DVFS technique. *DTPM* aggressively boosts the cores under a processor temperature constraint. With this addition, *NeuroDTPM* now aims at maximizing the overall system performance under the temperature constraints of both the manycore and the HBM similar to our *MTCM*.

In the following evaluation experiments, temperature violations on the HBM trigger the low-power mode on the impacted memory channels, while temperature violations on the manycore lead to throttling down the processor to the minimum VF level, i.e., 1 GHz.

C. Evaluation Results

We present our evaluation experiments, where we compare our *MTCM* against the two state-of-the-art techniques in terms of performance, temperature, and thermal efficiency. Fig. 7 shows the performance and temperature results.

Compared Against NeuroMap: Our *MTCM* shows a superior performance maximization ability compared to the *NeuroMap*, scoring a significant improvement of 1x% on average across all the workloads and arrival times, which can be explained as follows. First, *NeuroMap* periodically sets the clusters to either a low, medium, or high VF level depending on the memory intensity of the running applications at their current execution phase. This coarse-grained approach of setting the VF

levels of the clusters misses on the performance optimization opportunities that are exploited by our fine-granularity DVFS policy. This can also be seen in the box plots on Fig. 7, our *MTCM* better exploits the thermal headroom available on both the manycore and the HBM at runtime. Second, although the decisions of *NeuroMap* lead to a thermally safe operation of the HBM more than 99.5% of the execution time, their technique does not monitor the impact of such decisions on the manycore. Consequently, as expected, the temperature constraint of the manycore is frequently violated, triggering the TCC which throttles down all the clusters to the minimum VF level to restore the thermally safe operation. Although it has guaranteed a thermally safe operation of the HBM, the DVFS decisions of *NeuroMap* lead to frequent triggers of the TCC which suppress the performance gains of their task migration policy. This confirms the initial observation we have motivated in Section I-A, to consider the thermal effects on both the subsystems jointly.

Compared Against NeuroDTPM: Tackling the weaknesses of *NeuroMap*, *NeuroDTPM* employs the more advanced *DTPM* DVFS policy, which considers the temperature of the cores in its optimization. As expected, the performance is significantly improved compared to the *NeuroMap*, as now a thermally safe operation of both the subsystems is maintained 99.1% of the execution time. As can bee seen in the box plots in Fig. 7, *NeuroDTPM* manages to better utilize the thermal headroom available on the manycore thanks to its advanced DVFS policy from *DTPM* [34], with the exception of minor violations in less than 1% throughout the execution. The occasional thermal violations are only seen less than 1% of the time, on the 99th quantile and some upper outliers, which is expected due to sudden bursts in power consumption of either the manycore of the HBM. This also implies that the TCC is triggered less frequently, thus sustaining the performance improvements over time. Nevertheless, our *MTCM* still outperforms *NeuroDTPM* with an average performance improvement of 25.4% across all the experiments. This is due to the different application-to-cluster mappings, both the task migration policies apply at runtime. While *NeuroMap* does not consider the cache contention effects, our *MTCM*'s migration policy selects mappings that lead to the least contention between the corunning applications on a cluster, reducing the slowdown observed in their execution time, thereby further harnessing the performance potentials of the system. These results highlight the initial observation in our work that ignoring cache contention effects in such architectures leads to suboptimal resource management decisions.

To further analyse the results of the experiments, we present in Fig. 8 the thermal efficiency achieved by the three techniques at runtime, computed as the average number of millions of executed IPS per unit of temperature. The previously observed trends are maintained, where *MTCM* demonstrates a higher level of thermal efficiency compared to the other two techniques. *NeuroMap* shows the lowest thermal efficiency across all the workloads and arrival rates, mainly due to its long execution time, low average IPS and high number of thermal violations on the manycore side. *NeuroDTPM* shows a

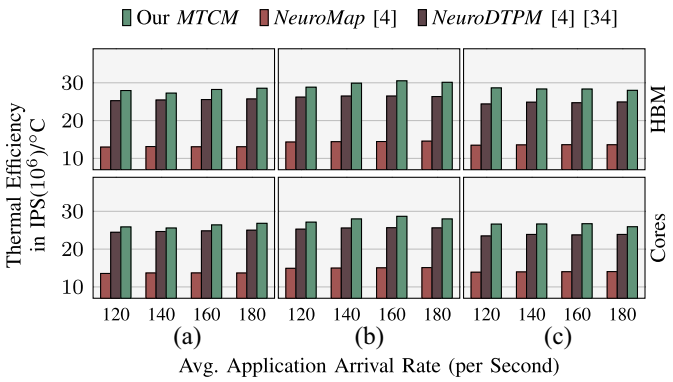

Fig. 8. Compared to the two state-of-the-art comparison techniques, our *MTCM* consistently executes more IPS per unit of temperature, thereby achieving an improved thermal efficiency across all the workloads and arrival rates. (a) Workload 1. (b) Workload 2. (c) Workload 3.

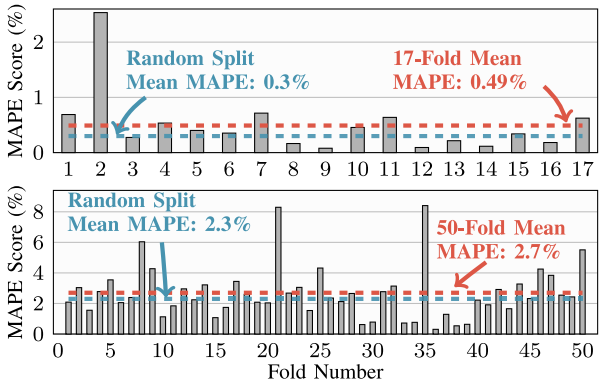

Fig. 9. With 17 unique applications in the dataset, running a leave-group-out cross-validation with a 17-fold for NN_m and a 50-fold for NN_p, where random groups of applications are excluded in each iteration, shows that the performance of our models on unseen applications is minimal across the folds with a mean MAPE that is only marginally higher than the MAPE obtained with a random split of the training/test dataset. This highlights the application-independence of our models and their ability to generalize to unseen traces, which is further demonstrated when the model is used on larger workloads and unseen scenarios.

significantly better thermal efficiency compared to *NeuroMap*, thanks to the fine-granularity *DTPM* DVFS and its thermally safety operation on both the manycore and the HBM. These results demonstrate that the joint consideration of cache contention and temperature of both the manycore and the HBM can lead to a better harnessing of the performance potentials of the system without violating its thermal constraints.

D. Generalization Analysis

As presented in the previous section, *MTCM* has demonstrated substantial performance gains over *NeuroMap* and *NeuroDTPM*, while maintaining a thermally safe operation of both the manycore and the HBM, even when our NN models are faced with unseen workloads and scenarios. We further analyse this generalization aspect by conducting a set of *Leave-Group-Out (GroupKFold)* cross-validation experiments. In K iterations, our dataset is shuffled and split such that each fold has an unique set of groups of applications. For each fold, a specific portion of the groups is used as the validation set, while the remaining groups form the training set. This

TABLE II
LIGHTWEIGHTNESS, HIGH ACCURACY, AND LOW MEMORY FOOTPRINT OF OUR FOUR NN MODEL ALLOW BOTH OUR POLICIES TO PERFORM EFFECTIVE RESOURCE MANAGEMENT AT RUNTIME WITHIN THEIR ALLOCATED EPOCH LENGTHS AT A NEGLIGIBLE OVERHEAD

Migration Policy				**DVFS Policy**		
Performance	**Memory Access**	**Core TM**	**HBM TM**	**Memory Access**	**Core TM Model**	**HBM TM**
$\mathbf{NN_p}$	$\mathbf{NN_m}$	$\mathbf{NN_{cores}}$	$\mathbf{NN_{hbm}}$	$\mathbf{NN_m}$	$\mathbf{NN_{cores}}$	$\mathbf{NN_{hbm}}$
64.5 µs	7.5 µs	25.5 µs	8 µs	4.4 µs	15.3 µs	4.8 µs

guarantees that the validation set for each fold contains groups not seen during training in that fold. It is important to note that our NN-based thermal models, i.e., NN_{cores} and NN_{hbm} cannot be considered in this generalization analysis through cross-validation as their training data is synthetically generated and does not contain application-specific per-core and per-bank power maps. For NN_m and NN_p, we use *LeaveOneGroupOut* from the *sklearn* library to conduct the cross-validation experiments. Since, the training data of our NN_m is limited to single applications, we run a 17-fold exploration, corresponding to the number of unique applications in the dataset. For NN_p, we cap the folding to 50 groups with each group having up to three applications, corresponding to the number of applications involved in a migration scenario, in addition to cases where the destination cluster is empty. This leads to the exploration of scenarios where up to 24% of the applications are unseen during training. In all these experiments, we use the final models described in Table I. Fig. 9 shows the results of these experiments. For the NN_m, a steady error distribution is observed across the validation set in all the folds, except for one scenario, $K = 2$, where we observe a slightly higher deviation from the mean. This is due to the fact that excluding *canneal* leads to the exclusion of a significant number of data slices from the dataset as *canneal*'s execution time is significantly longer than most applications in the two benchmark suites. Still, the mean MAPE is 0.49%, only marginally higher than the 0.3% observed when training with the randomly split dataset in Section IV.

A similar behavior is observed in the training of NN_p. In folds like $K = 36$, we observe a significantly lower MAPE compared to the mean MAPE, as that fold comprises less complex migration scenarios, where the destination cluster is empty. On the other extreme, folds like $K = 21$ with *canneal* excluded from the training set, show a significantly higher MAPE compared to the mean. Nevertheless, the mean MAPE is 2.7%, only 0.4% higher than the MAPE observed when training with the randomly split dataset in Section IV.

E. Overhead Analysis

To analyse the overhead of *MTCM*, we run additional experiments where the technique is bundled as a single-threaded application that is mapped to a single core on our target platform, and executed at the maximum supported VF level. In these additional experiments, the randomly generated workloads and arrival rates from the evaluation experiments in Section VI-A are reused. Table II shows the average time spent per epoch in each phase of our proposed technique across all the experiments. *MTCM*'s DVFS policy takes 24.6 μs on average to boost or throttle clusters, representing 2.46% of the adopted 1 ms DVFS epoch on a single core. *MTCM*'s migration policy takes 105.5 μs on average to find a migration option and apply it, equivalent to 1.05% of the 10 ms migration epoch on a single core. Therefore, the benefits of our proposed *MTCM* can be obtained with a negligible overhead on our target 64-core processor.

VII. CONCLUSION

We presented *MTCM*, the first resource management technique that considers cache contention in maximizing the system performance, while maintaining thermal safety on the modern systems with a clustered manycore and HBM. Enabled by our fast, yet accurate, lightweight NN models for performance, memory access, and temperature prediction, our task migration and DVFS policies can perform thermally safe resource management at runtime, even when exposed to scenarios that are unseen at design time. As a result, our *MTCM* achieves substantial performance improvements compared to the state-of-the-art, validating the significance of jointly considering the contention and temperature problems within performance maximization in the systems with many-cores and HBM.

REFERENCES

[1] "Intel® xeon® CPU max series." 2023. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2023-01/xeon-cpu-max-series-product-brief.pdf
[2] K. Son et al., "Thermal and signal integrity co-design and verification of embedded cooling structure with thermal transmission line for high bandwidth memory module," *IEEE Trans. Compon., Packag. Manuf. Technol.*, vol. 12, no. 9, pp. 1542–1556, Sep. 2022.
[3] Y. Shen, L. Schreuders, A. Pathania, and A. D. Pimentel, "Thermal management for 3D-stacked systems via unified core-memory power regulation," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 5S, p. 120, Sep. 2023. [Online]. Available: https://doi.org/10.1145/3608040
[4] S. Pandey and P. R. Panda, "NeuroMap: Efficient task mapping of deep neural networks for dynamic thermal management in high-bandwidth memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 3602–3613, Nov. 2022.
[5] *Intel 64 and IA-32 Architectures Software Developer's Manual*, Intel Corp., Santa Clara, CA, USA, 2016.
[6] B. Pourmohseni, S. Wildermann, F. Smirnov, P. E. Meyer, and J. Teich, "Task migration policy for thermal-aware dynamic performance optimization in many-core systems," *IEEE Access*, vol. 10, pp. 33787–33802, 2022.
[7] M. B. Sikal, H. Khdr, M. Rapp, and J. Henkel, "Machine learning-based thermally-safe cache contention mitigation in clustered manycores," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.
[8] T. Marinakis, S. Kundan, and I. Anagnostopoulos, "Meeting power constraints while mitigating contention on clustered multiprocessor system," *IEEE Embedded Syst. Lett.*, vol. 12, no. 3, pp. 99–102, Sep. 2020.

[9] M. Rapp, M. B. Sikal, H. Khdr, and J. Henkel, "SmartBoost: Lightweight ML-driven boosting for thermally-constrained many-core processors," in *Proc. Design Autom. Conf. (DAC)*, 2021, pp. 265–270.

[10] J. Henkel, H. Khdr, and M. Rapp, "Smart thermal management for heterogeneous multicores," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2019, pp. 132–137.

[11] M. B. Sikal, H. Khdr, M. Rapp, and J. Henkel, "Thermal- and cache-aware resource management based on ML-driven cache contention prediction," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2022, pp. 1384–1388.

[12] C.-S. Oh et al., "22.1 a 1.1V 16GB 640GB/s HBM2E DRAM with a data-bus window-extension technique and a synergetic on-die ECC scheme," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 330–332.

[13] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 374–385.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 1995, pp. 24–36.

[15] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.

[16] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. Linux Symp.*, 2006, pp. 1–18.

[17] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2010, pp. 463–470.

[18] H. Sultan and S. R. Sarangi, "A fast leakage aware thermal simulator for 3D chips," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 1733–1738.

[19] H. Sultan, A. Chauhan, and S. R. Sarangi, "A survey of chip-level thermal simulators," *ACM Comput. Surv.*, vol. 52, no. 2, p. 42, Apr. 2019. [Online]. Available: https://doi.org/10.1145/3309544

[20] Y. G. Kim, M. Kim, J. Kong, and S. W. Chung, "An adaptive thermal management framework for heterogeneous multi-core processors," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 894–906, Jun. 2020.

[21] S. Kundan and I. Anagnostopoulos, "A machine learning approach for improving power efficiency on clustered multi-processor system," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, 2020, pp. 1–5.

[22] N. Mishra, J. D. Lafferty, and H. Hoffmann, "ESP: A machine learning approach to predicting application interference," in *Proc. Int. Conf. Auton. Comput. (ICAC)*, 2017, pp. 125–134.

[23] S. Dey, A. K. Singh, X. Wang, and K. D. McDonald-Maier, "DeadPool: Performance deadline based frequency pooling and thermal management agent in DVFS enabled MPSoCs," in *Proc. 6th IEEE Int. Conf. Cyber Security Cloud Comput. (CSCloud) 5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, 2019, pp. 190–195.

[24] D. Liu, S.-G. Yang, Z. He, M. Zhao, and W. Liu, "CARTAD: Compiler-assisted reinforcement learning for thermal-aware task scheduling and DVFS on multicores," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1813–1826, Jun. 2022.

[25] M. Rapp, H. Khdr, N. Krohmer, and J. Henkel, "NPU-accelerated imitation learning for thermal optimization of QoS-constrained heterogeneous multi-cores," *ACM Trans. Design Autom. Electr. Syst.*, vol. 29, no. 1, p. 16, 2024.

[26] H. Kim et al., "Signal integrity analysis of through-silicon via (TSV) with a silicon dioxide well to reduce leakage current for high-bandwidth memory interface," *IEEE Trans. Compon., Packag. Manuf. Technol.*, vol. 13, no. 5, pp. 700–714, May 2023.

[27] W.-H. Lo, K.-Z. Liang, and T. Hwang, "Thermal-aware dynamic page allocation policy by future access patterns for hybrid memory cube (HMC)," in *Proc. Design, Autom. Test Europe Conf. (DATE)*, Mar. 2016, pp. 1084–1089.

[28] *High Bandwidth Memory DRAM (HBM3)*, JEDEC Standard JESD238A, 2022.

[29] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *Proc. Workshop Job Scheduling Strategies Parallel Process.*, 1998, pp. 1–24.

[30] S. Boyd-Wickizer et al., "Corey: An operating system for many cores," in *Proc. Symp. Oper. Syst. Design Implement. (OSDI)*, 2008, pp. 1–15.

[31] L. Siddhu et al., "CoMeT: An integrated interval thermal simulation toolchain for 2D, 2.5D, and 3D processor-memory systems," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 3, p. 44, Aug. 2022. [Online]. Available: https://doi.org/10.1145/3532185

[32] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Parallel Archit. Compilation Techn. (PACT)*, 2008, pp. 72–81.

[33] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2012, pp. 33–38.

[34] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 544–557, Mar. 2018.

[35] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. High Perform. Comput., Netw., Storage Anal. (SC)*, 2011, pp. 1–12.