Thermal- and Cache-Aware Resource Management based on ML-Driven Cache Contention Prediction

Mohammed Bakr Sikal, Heba Khdr, Martin Rapp, Jörg Henkel
Karlsruhe Institute of Technology, Karlsruhe, Germany
{bakr.sikal, heba.khdr, martin.rapp, henkel}@kit.edu

***Abstract*—While on-chip many-core systems enable a large number of applications to run in parallel, the increased overall performance may come at the cost of complicating the performance constraints of individual applications due to contention on shared resources. For instance, the competition for last-level cache by concurrently-running applications may lead to slowing down the execution and to potentially violating individual performance constraints. Clustered many-cores reduce cache contention at chip level by sharing caches only at cluster level. To reduce cache contention within a cluster, state-of-the art techniques aim to co-map a memory-intensive application with a compute-intensive application onto one cluster. However, compute-intensive applications typically consume high power, and therefore, executing another application in their nearby cores may lead to high temperatures. Hence, there is a trade-off between cache contention and temperature. This paper is the first to consider this trade-off through a novel thermal- and cache-aware resource management technique. We build a neural network (NN)-based model to predict the slowdown of the application execution induced by cache contention feeding our resource management technique that then optimizes the application mapping and selects the voltage/frequency levels of the clusters to compensate for the potential contention-induced slowdown. Thereby, it meets the performance constraints, while minimizing temperature. Compared to the state of the art, our technique significantly reduces the temperature by 30% on average, while satisfying performance constraints of all individual applications.**

***Index Terms*—thermal optimization, resource management, cache contention, application mapping, DVFS, machine learning**

I. Introduction

The need for continued performance growth in computing systems lead to the integration of many CPU cores on one chip to concurrently run many multi-threaded-applications [1]. Although this ability of massive parallel computations improves the overall system performance [2], it imposes two problems.

The first problem is the heat transfer between simultaneously-running cores, potentially leading to thermal hotspots, which severely impact the reliability [3]. The second problem is the contention on shared resources, which slows the application execution down, potentially leading to performance constraint violations [4]. The main resource thereby is the last-level cache (LLC). Employing a clustered many-core system (e.g., AMD Zen 3 microarchitecture [5]) helps in reducing cache contention at chip level by sharing the LLC only between cores of the same cluster. Recent resource management techniques (e.g., [6], [7]) have started to reduce cluster-level cache contention via application mapping. Specifically, they co-map memory-intensive applications with compute-intensive ones,

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Fndn.) – Proj. Nb. 146371743 – TRR 89 Invasive Computing.
978-1-6654-3274-0/21/$31.00

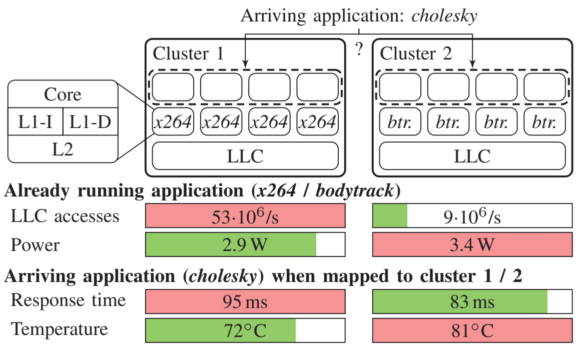

Arriving application: cholesky
Cluster 1
?
Cluster 2
Core
L1-I
L1-D
L2
x264
btr.
LLC
Already running application (x264 / bodytrack)
LLC accesses
53·10^6/s
9·10^6/s
Power
2.9 W
3.4 W
Arriving application (cholesky) when mapped to cluster 1 / 2
Response time
95 ms
83 ms
Temperature
72°C
81°C

Fig. 1. Trade-off between cache contention and temperature.

since the latter put less stress on the LLC, thereby reducing contention. However, this might elevate on-chip temperatures since compute-intensive applications typically lead to higher power consumption, compared to memory-intensive ones. The following motivational example demonstrates this trade-off between cache contention and temperature.

A. Motivational Example

In this illustrative example, a simulation is conducted where two applications[1], *x264* and *bodytrack*, are running on two individual clusters on the chip, as shown in Fig. 1. *x264* is a memory-intensive application, reflected in its high number of LLC accesses per second, compared to *bodytrack*, which is compute-intensive. On the other hand, the power consumption of *bodytrack* is higher than the one of *x264*. A new application, *cholesky*, arrives in the system, and needs to be mapped to one of the clusters. We analyse the implications of the two mappings. As expected, the response time of *cholesky* when co-mapped with *x264* on Cluster 1 is longer than its response time when co-mapped with *bodytrack* on Cluster 2, due to the slowdown induced by LLC contention on Cluster 1. On the other side, co-mapping *cholesky* with *bodytrack* leads to an increase in the average cluster temperature by 9 °C, due to the high power consumption of *bodytrack*.

In summary, application-to-cluster mapping exposes a trade-off between cache contention and temperature. Particularly, co-mapping an application with a memory-intensive application increases cache contention and results in an execution slowdown, but might decrease temperature due to the decreased computation on CPU cores. On the other hand, co-mapping with a compute-intensive application decreases cache contention but might increase temperature, since intensive computation

[1]The applications used in this paper are from *PARSEC* [1] and *SPLASH-2* [8]

typically results in higher power consumption. This trade-off has not yet been observed nor exploited by the state of the art.

B. Challenges and Contributions

The state of the art has tackled each of the aforementioned problems, i.e., temperature increase and cache contention, separately. This paper introduces a resource management technique, *TcRM*, *that leverages, for the first time, the trade-off between cache contention and temperature* in clustered many-core systems, via the joint selection of application mapping and voltage/frequency (V/f) levels. *TcRM* aims at minimizing temperature under performance constraints while considering the execution slowdown induced by cache contention. To achieve this, two key challenges need to be tackled:

1) The slowdown induced by cache contention between potential *co-mapped* applications, i.e., concurrently-running on a cluster, needs to be estimated at arrival times of applications in order to select the suitable application-to-cluster mapping. Tackling this challenge through design-time profiling of parallel execution of applications is not feasible for two reasons. First, the number of all possible combinations of applications is exponential. Secondly, cache contention between co-mapped applications depends on their memory intensiveness within the period of time when they are running concurrently, because application behavior may differ throughout the execution time. This shared time period depends on the relative arrival times of the co-mapped applications, which cannot be known a priori in an open system [9]. Moreover, cache contention originates from several underlying effects (limited capacity / bandwidth), which are affected differently by different applications, and have also diverse impact on the slowdown in the execution time of different applications. That makes cache contention behavior too complex to be analytically modeled. We tackle this challenge by employing a machine learning (ML) model, specifically an NN, that is able to cope with this complexity and learn from limited training data at design time then generalize to unseen data at runtime [10].

2) How to find the required V/f level to compensate for the contention-induced slowdown, while still minimizing the temperature. To tackle this challenge, we consider the V/f level as a feature in the NN model. This enables *TcRM* to *jointly* find the application-to-cluster mappings and cluster V/f levels that compensate for the contention-induced slowdown in the execution of the co-mapped applications and thereby satisfying their performance constraints, while reducing the chip temperature. Importantly, *TcRM* does not only determine application-to-cluster mapping, but also selects thread-to-core mappings to exploit all potentials for thermal reduction.

In summary, our novel contributions are:

- We design, train, and employ an NN model to predict the slowdown in the application execution induced by cache contention between co-mapped applications in a cluster at a given V/f level.
- We introduce a thermal- and contention-aware resource management technique to determine application-to-cluster and thread-to-core mapping, as well as the required V/f levels of the clusters that compensate for the contention-induced slowdown predicted by the NN model, such that the performance constraints of all applications are satisfied, while minimizing temperature.

II. Related Work

The related research can be categorized as follows: thermal-aware techniques and cache-contention-aware techniques.

Thermal-aware techniques: Different means have been used by thermal management techniques, including task migration, V/f level selection, and application mapping. A recent thermal management technique that uses task migration is proposed in [11]. The technique decreases the chip temperature by migrating tasks from cores with high temperature to colder ones. Minimizing temperature through dynamic voltage and frequency scaling (DVFS) [12] has been adopted by most thermal management techniques due to its large impact on power and thereby temperature. DVFS is used at core level, e.g., [13], if the architecture supports it, or at cluster level, e.g., [14], for clustered many-core systems. Application mapping is also used by thermal management techniques, especially in many-core systems to determine where to map the application so that its active cores are surrounded by idle cores to absorb the generated heat [15]. The technique in [16] first selects the region of the application to map, then the cores inside this region to map the application threads. This selection attempts to place inactive cores nearby active cores of the applications to reduce temperature. This technique shares with us the goal of selecting an application mapping to reduce temperature, but it does not consider the cache contention problem, nor does any of the state-of-the-art techniques in this category.

Cache-contention-aware techniques: Cache contention is a well-known problem in the era of many-core systems. Several works have proposed models [4], [17] to estimate the cache contention-induced slowdown in the application execution. Such models enable scheduling policies to account for the estimated slowdown through scheduling decisions [18]. The scheduling policy proposed in [19] considers cache contention to satisfy the performance constraints of applications. The work proposed in [6] demonstrated that co-mapping memory-intensive and compute-intensive applications reduces cache contention, since the latter have less memory accesses. A recent technique [20] has employed ML to select the minimum V/f level for each cluster that satisfies the performance constraints of applications, considering the predicted contention-induced slowdown, while minimizing power. This technique shares with us the consideration of cache contention in selecting the V/f levels of the clusters that satisfy the performance constraints of applications. However, it does not consider the thermal problem.

In summary, none of the state-of-the-art techniques has jointly considered both problems, i.e., cache contention and temperature increase, as they have not yet observed the trade-off between cache contention and temperature.

III. Problem Formulation

Our targeted system is a clustered many-core system with N cores divided into L clusters. A cluster ℓ has $n{=}N/L$

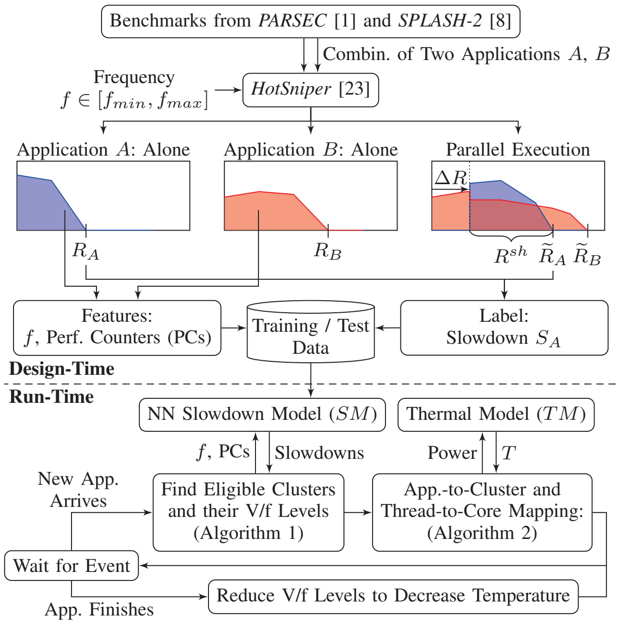
homogeneous cores that share one LLC, and the same V/f level, denoted as $f_\ell \in [f_{min}, f_{max}]$. To specify which cores belong to which cluster, we define a binary matrix $\mathbf{Q} = [q_{i,\ell}]_{N\times L}$, where $q_{i,\ell} = 1$ iff core i belongs to cluster ℓ. The total power (dynamic and leakage) consumption of core i is p_i. The standard well-known RC-thermal model [21], denoted as TM, is used to estimate the steady-state temperature of all cores, $T{=}[t_i]_N$ based on the power. Our application model comprises K multi-threaded applications, which arrive at a-priori unknown times (open system [9]). Each application k has h_k parallel threads, where only one thread is mapped to a core at any given time; a common model for multi-core systems [22]. The response time of application k is denoted as R_k, and its performance constraint is $\hat{R}_k$, which indicates the maximum period of time to finish the application execution. Application-to-cluster mappings are identified by a binary matrix $G{=}[g_{\ell,k}]_{L\times K}$. Each application can be mapped to one cluster. However, multiple applications can be mapped to the same cluster. A binary matrix $V{=}[v_{i,k}]_{N\times K}$ is defined to represent the mapping of the application threads to the cores. The objective in this paper is to find for each arriving application the application-to-cluster and thread-to-core mapping, and to select the V/f levels of all clusters so that the peak temperature of the chip $\max T$ is minimized while the performance constraints of all applications are satisfied, considering the cache-contention-induced slowdown. Mathematically, we can express this problem as finding matrices G, V, and $f_\ell\ \forall\ell$, in order to:

$$\text{Minimize } \max T \text{ s.t.: } R_k \le \hat{R}_k \forall k$$

Our proposed approach to solve this problem is illustrated in Fig. 2. At design time, we train an NN model (details in Section IV) on some application scenarios at different V/f levels. At runtime, our proposed runtime resource management technique (details in Section V) employs this model to predict contention-induced slowdowns in application executions even for unseen scenarios, and uses this prediction to select application-to-cluster, thread-to-core mappings, and the V/f levels so that the aforementioned goal/constraint are satisfied.

IV. Slowdown Prediction Model

Predicting the cache contention-induced slowdown is a challenging task. Firstly, cache-contention behavior is too complex to be described by a simple analytical model. An example of this complexity is shown in Table I, where each cell reports the resulting slowdown of application A when co-mapped with application B. The first observation is that *streamcluster* is not slowed down by any application while it slows down all applications it is co-mapped with. Secondly, *lu.cont* suffers a slowdown of 20.8% and 27.2% from *streamcluster* and *cholesky*, respectively. However, co-mapping *lu.ncont* with the same two applications shows the opposite trend: *lu.ncont* suffers a slowdown of 22.0% and 17.4% from *streamcluster* and *cholesky*, respectively. Secondly, cache-contention behavior varies throughout the execution of co-mapped applications. In fact, the same applications exhibit different slowdowns if arrived to the system at different times. Hence, predicting the slowdown cannot be tackled by design-time profiling due to



Benchmarks from *PARSEC* [1] and *SPLASH-2* [8]
Combin. of Two Applications A, B
Frequency $f \in [f_{min}, f_{max}]$
HotSniper [23]
Application A: Alone
Application B: Alone
Parallel Execution
ΔR
R_A
R_B
R^{sh} $\widetilde{R}_A$ $\widetilde{R}_B$
Features: f, Perf. Counters (PCs)
Training / Test Data
Label: Slowdown S_A
Design-Time
Run-Time
NN Slowdown Model (SM)
Thermal Model (TM)
f, PCs
Slowdowns
Power
T
New App. Arrives
Find Eligible Clusters and their V/f Levels (Algorithm 1)
App.-to-Cluster and Thread-to-Core Mapping: (Algorithm 2)
Wait for Event
Reduce V/f Levels to Decrease Temperature
App. Finishes

Fig. 2. *TcRM* flow. At design time, training data is generated to train the prediction NN model. At runtime, application mapping to the clusters, and their V/f levels are determined to achieve *TcRM* goal.

TABLE I
Different application combinations show different slowdowns

App B \ App A	*streamclus.*	*lu.cont*	*cholesky*	*lu.ncont*
streamclus.	0.0%	20.8%	3.6%	22.0%
lu.cont	0.0%	28.9%	12.5%	20.8%
cholesky	0.0%	27.2%	8.3%	17.4%
lu.ncont	0.0%	26.7%	18.8%	18.8%

the exponential scenarios in two dimensions: the combinations of co-mapped applications and their relative arrival times. To address this complexity, *TcRM* employs an NN model to predict the cache contention-induced slowdown between co-mapped applications. The following subsections describe the training data generation, feature selection and the model architecture.

A. Training Data Generation

The upper part of Fig. 2 illustrates our methodology for generating training data. We first identify representative execution scenarios to generate traces, from which the model features will be extracted. As previously mentioned, the number of application combinations is exponential. Therefore, we choose to analyze combinations of two applications only, A and B. As shown in Fig. 2, the shift in the arrival times, ΔR, determines the shared execution time R^{sh} of the co-mapped applications. Hence, different ΔR values lead to different cache-contention behaviors. For our training data, we select two arbitrary values for ΔR. Importantly, our model generalizes to any number of applications and to any ΔR, as their impacts on cache contention behavior are reflected in our selected features. Moreover, *TcRM* is evaluated on random arrival times, as described in Section VI. Besides these *parallel executions* of applications, we also run each of them alone in a

Algorithm 1 Find Eligible Clusters and their V/f Levels

```
Input: arriving app. $A$, slowdown model $SM$ with error $\varepsilon$
Output: set of eligible clusters $C^E$ and V/f levels
  $C^E \leftarrow \{\ell : \forall k\ g_{\ell,k} = 0, f_{min}\}$        ▷ completely free clusters
  if $C^E \neq \{\}$ then return
  for each $\ell \in \{1, \ldots, L\}$ do                 ▷ iterate over clusters
      $X \leftarrow \{i : q_{i,\ell} \wedge \forall k\ v_{i,k} = 0\}$        ▷ free cores on cluster $\ell$
      if $|X| < h_A$ then continue        ▷ not enough free cores on $\ell$
      $f \leftarrow$ BinarySearch($[f_{min}, f_{max}], \lambda f.$SATISFIES$(f, \ell)$)
      $C^E \leftarrow C^E \cup \{(\ell, f)\}$
  procedure SATISFIES($f, \ell$)                  ▷ deadline violation check
      $\Gamma \leftarrow \{A\} \cup \{B_1, \ldots, B_Z\}$          ▷ set of $A$ and other apps
      for each $\gamma \in \Gamma$ do                  ▷ violation check for $\gamma$
          $R^{sh} \leftarrow$ shared time of $\gamma$ and other apps $\Gamma \setminus \{\gamma\}$
          $S_\gamma \leftarrow SM(f, \text{PC}(\gamma, R^{sh}), \sum_{\omega \in \Gamma \setminus \{\gamma\}} \text{PC}(\omega, R^{sh}))$
          $R'_\gamma \leftarrow (S_\gamma + 1 + \varepsilon) \cdot R^{sh} + (R_\gamma - R^{sh})$
          if $R'_\gamma > \hat{R}_\gamma$ then return false
      return true
```

Algorithm 2 App-To-Cluster and Thread-to-Core Mapping

```
Input: eligible clusters $C^E$, arriving app. $A$
Output: mapping and corresponding frequency of app. $A$
  for each $(\ell, f) \in C^E$ do
      $P^o \leftarrow \{1, \ldots, \ell{-}1, \ell{+}1, \ldots, L\}$ ▷ powers of other clusters
      $X \leftarrow \{i : q_{i,\ell} \wedge \forall k\ v_{i,k} = 0\}$        ▷ free cores on cluster $\ell$
      $P^B \leftarrow$ power of already running applications on $\ell$ at $f$
      $M_\ell \leftarrow \{\}$                      ▷ start with empty mapping on $\ell$
      while $|M_\ell| < h_A$ do
          for each core $i \in X$ do
              $P \leftarrow P^o + P^B + P^A(M_\ell \cup \{i\}, f)$ ▷ full chip power
              $T \leftarrow TM(P)$         ▷ steady-state temperature with $p$
              $\hat{T}_i \leftarrow \max T$   ▷ peak temperature when using core $i$
          $i_{\text{opt}} \leftarrow \arg\min_i \hat{T}_i$     ▷ select core with min. peak temp.
          $M_\ell \leftarrow M_\ell \cup \{i_{\text{opt}}\}$               ▷ add core to mapping
          $X \leftarrow X \setminus \{i_{\text{opt}}\}$          ▷ core is not available any more
          if $|M_\ell| = h_A$ then
              $\hat{T}_\ell \leftarrow \min_i \hat{T}_i$       ▷ temp. when mapping on cluster $\ell$
              $f'_\ell \leftarrow f$            ▷ remember frequency for cluster $\ell$
  $\ell_{\text{opt}} \leftarrow \arg\min_\ell \hat{T}_\ell$   ▷ select cluster with min. peak temperature
  $g_{\ell_{\text{opt}},A} \leftarrow 1$, $v_{i,k} \leftarrow 1\ \forall i \in M_{\ell_{\text{opt}}}$, $f_{\ell_{\text{opt}}} \leftarrow f'_\ell$  ▷ map $A$, set V/f level
```

cluster, i.e., *single executions*. These executions are needed to extract features and to calculate the resulting slowdown, i.e., the model label. The selected scenarios are run at different V/f levels, since the latter is a feature as motivated in Section I.

B. Feature selection and model architecture

The features of our model are the relevant cache-/memory-related Performance Counters (PCs) extracted from the single execution traces of applications, specifically from the trace parts that are corresponding to R^{sh} in the parallel execution trace. To find the smallest subset of PCs that accurately capture the cache-contention behavior of applications, while yielding the lowest prediction error, we use *Lasso Regression* to identify the features with the highest predictability of S_A, and the Pearson Product-Moment Correlation matrix to analyze their correlations. After investigating a large number of traces, we shortlisted the following features: LLC accesses and misses, DRAM reads and DRAM accesses. Thus, each row of the training data will consist of this 4-tuple of PCs of application A and the ones of application B. The model label is the resulting slowdown suffered by application A due to co-mapping with application B and calculated as: $S_A = \frac{\widetilde{R}_A}{R_A} - 1$, where R_A denotes the response time of application A when running alone, while $\widetilde{R}_A$ denotes its response time when co-mapped. As aforementioned, application A is co-mapped with one application, i.e., application B. In case, it will be co-mapped with more than one application at run-time, the features that correspond to application B are filled with the accumulated PCs of these applications.

We build a lightweight NN model consisting of 4 hidden dense layers (64, 32, 16 and 8 neurons) with ReLU activation and one output layer of 1 neuron with linear activation. Our NN achieves a 2% RMSE with a 0.99-quantile of the prediction errors ε =5% and a low inference overhead of 3 µs.

V. Contention-Aware Mapping

As shown in the bottom part of Fig. 2, *TcRM* takes runtime decisions at two events; when a new application arrives in the system, and when an application finishes its execution. Upon arrival of an application A, *TcRM* executes two steps: 1) Finding eligible clusters for hosting application A and obtaining the minimum V/f level that satisfies the performance constraints of the application A and the already-running applications B on the cluster. 2) Selecting the application-to-cluster and thread-to-core mapping that minimizes the temperature.

The first step is illustrated in Algorithm 1. First, *TcRM* looks for free clusters on the chip. If found, they will be passed to the second step. Otherwise, the search for eligible clusters starts. An eligible cluster for hosting application A must have enough free cores to execute it and there must be at least one V/f level, f_ℓ, that can compensate for the predicted slowdowns of all potential co-mapped applications on that cluster, denoted as Γ. To find f_ℓ, a binary search is performed. At each iteration, the slowdown suffered by each application, $\gamma \in \Gamma$, is estimated, considering the tested V/f level in that iteration. To do that, we pass to the model the relevant PCs of the application γ, and the accumulated PCs of all other applications in $\Gamma \setminus \{\gamma\}$. The PCs are extracted from the parts of the single execution traces that correspond to the shared period R^{sh}. Considering the predicted slowdown, the response time of γ, i.e., R'_γ, is estimated[2]. If $\hat{R}_\gamma$ of each application is satisfied on cluster ℓ at the current frequency f_ℓ, then, (ℓ,f) is added to the set of eligible clusters C^E and their frequencies.

In the second step, illustrated in Algorithm 2, the goal is to find a hosting cluster ℓ_{opt} and the set of cores $M_{\ell_{\text{opt}}}$ for application A, so that the peak steady-state temperature $\max T$ on the chip is minimized. Testing all thread-to-core mapping combinations has an exponential complexity. Instead, we employ a greedy search with a reduced complexity, that traverses all eligible clusters to select the mapping that reduces

[2]The 0.99-quantile of the model error ε is added to slowdown predictions, to avoid underestimations which may result in performance constraint violations.

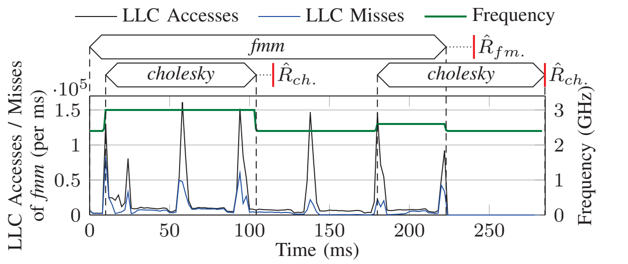


Fig. 3. *TcRM* DVFS decisions depend on the execution phases of applications.

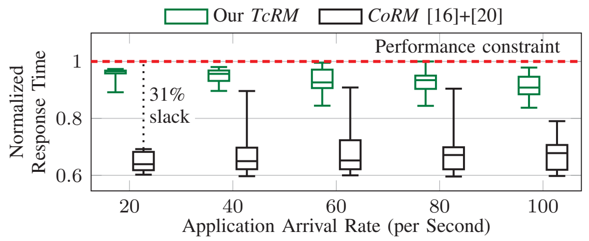


Fig. 4. *TcRM* avoids large slack time in the application execution time to reduce the temperature. Therefore, all applications finish close to their deadlines. In contrast, *CoRM* shows large slack.

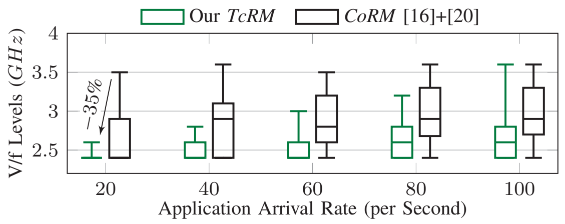


Fig. 5. *TcRM* sets cores to significantly lower V/f levels compared to *CoRM*.

the temperature. On each eligible cluster, the thread mapping to each free core is tested, and the estimated peak chip temperature is stored. After testing all thread-to-core mapping options on a cluster, we select the mapping that results in the lowest $\max T$. The best thread-to-core mapping M_ℓ on this cluster and its corresponding estimated T_ℓ are stored. Finally, we select ℓ_{opt}, the eligible cluster with the lowest $T_{\ell_{\text{opt}}}$, and we map the application threads to $M_{\ell_{\text{opt}}}$. Upon departure of an application hosted by a cluster ℓ, f_ℓ is reset to the minimum V/f level that satisfies the performance constraint of the remaining applications.

Illustrative example: To illustrate the work of *TcRM*, we run an experiment on one cluster only, where two instances of *cholesky* are co-mapped with *fmm* at different execution times. We consider that an application can meet its performance constraint while running alone on the cluster at 2.4 GHz. Fig. 3 shows the arrival times of the three applications, the LLC accesses and misses of *fmm* and the selected frequencies by *TcRM*. At first, *fmm* is running alone on the cluster at 2.4 GHz. Then, the first instance of *cholesky* arrives, while *fmm* is heavily accessing the LLC with a high miss rate. As a response, *TcRM* chooses 3 GHz as the minimum frequency that will meet the performance constraints of both applications. At t=104ms, *cholesky* finishes its execution and *TcRM* reduces the frequency of the cluster to 2.4 GHz. After a while, a new instance of *cholesky* arrives, but this time, *fmm* is accessing the LLC less heavily. Thus, *TcRM* chooses 2.6 GHz. At t=223ms, *fmm* finishes its execution, and the cluster is throttled down to 2.4 GHz as *cholesky* is running alone. Hence, *TcRM* has selected the required minimum V/f level to meet the performance constraints of all applications, while considering the variation in cache-contention behavior throughout execution time.

VI. Experimental Evaluation

We run simulations on *HotSniper* [23], an augmented version of the *Sniper* [24] simulator with *McPAT* [25] and *HotSpot* [21] integration for power and temperature estimations, respectively. We model a many-core processor organized in 8 clusters of 8 cores, similar to the AMD Zen 3 [5] microarchitecture, found in many commercial processors. Each core has 64 KB of L1 cache and 256 KB of L2 cache. All cores within the same cluster share 8 MB of LLC and a memory controller. Per-cluster DVFS is supported, with a frequency range from 1 GHz to 4 GHz with steps of 0.2 GHz. The default *HotSpot* [21] cooling parameters are used with an ambient temperature of 45 °C. We conduct our experiments using benchmarks that show cache-contention behavior at the design profiling phase, to guarantee real testing for our prediction model. These applications include: *bodytrack*, *streamcluster*, *x264*, *cholesky*, *raytrace*, *radix*, *fmm*, *fft*, *lu.cont*, *lu.ncont*, *water.nsq* and *water.sp*, each with 4 parallel threads. We use a mixed workload of 20 applications randomly selected from this set of benchmarks with medium/large input sizes. The arrival times of the applications are sampled from a Poisson distribution with varying arrival rates, to consider various system utilization values and importantly, to allow testing our technique for unseen scenarios. The performance constraint of an application is defined as its execution time at 2.4 GHz.

Comparison Technique: Our *TcRM* is the first to *jointly* consider the temperature increase problem and the cache-contention one, while the state of the art tackles each problem separately. Hence, we cannot compare *TcRM* directly against any state-of-the-art technique. Therefore, we develop a baseline, *CoRM*, as a combination of the two state-of-the-art works: the thermal-aware application mapping proposed in [16] and the contention-aware V/f level selection proposed in [20]. At runtime, when a new application arrives, *CoRM* first selects the cluster with the lowest average temperature of all its cores, and then finds the task-to-core mapping that reduces the temperature (using the approach in [16]). This is done by positioning idle cores between active ones. Moreover, temperature will be reduced further, as the NN-based approach in [20] selects the minimum V/f level of the cluster that satisfies performance constraints of the co-mapped applications, based on the current frequency and readings from the PCs. Their approach selects the maximum V/f level as the starting operating point for the applications, then call the model to select the minimum V/f level that satisfies the constraints. The model will not be called again during the application execution, unless a violation of performance constraint is predicted.

Experimental Results: We compare *TcRM* and *CoRM* in terms of the response times of the applications relative to their performance constraints and the resulting temperatures. Fig. 4 shows that both techniques satisfy all performance constraints.

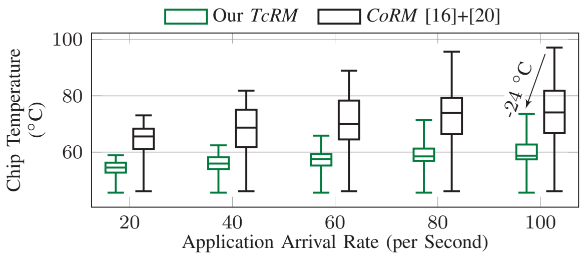


Fig. 6. *TcRM* significantly reduces the chip temperature over *CoRM*.

However, *CoRM* results in a significant slack time, up to 38%. These results are explained by the selected frequencies by both techniques illustrated in Fig. 5. *CoRM* always selects higher frequencies, since it starts the application execution at f_{max} as mentioned in [20]. Then, the model is called based on the current performance counter readings, which reflect only the instantaneous cache-contention behavior of the running applications, but this might vary throughout the execution, as shown in Fig. 3. In contrast, our *TcRM* considers the cache-contention behavior during the whole shared execution time of the co-mapped applications. This allows our model to provide a more accurate prediction of the slowdown, thus selecting lower V/f levels, ultimately yielding minimal slack times. Fig. 6 shows the resulting temperatures in our comparison experiments. *TcRM* reduces the temperature by 30% on average compared to *CoRM*. There are two reasons; 1) *TcRM* selects lower V/f levels as shown in Fig. 5. 2) our mapping Algorithm 2 tests much more mapping possibilities compared to the ones tested by the mapping policy [16] used in *CoRM*. Specifically, *CoRM* only selects the coldest cluster, while *TcRM* first estimates the resulting temperatures of mapping the application to each available cluster, then selects the one that results in the minimum temperature.

Runtime Overhead: The execution overhead of *TcRM* depends on the utilization of the chip as new applications arrive. We measure the overhead induced by *TcRM* in the extreme scenario, i.e., when none of the clusters is neither free nor fully utilized. In this scenario, *TcRM* traverses all clusters to find the eligible ones. This requires a binary search to find the minimum V/f level that satisfies the performance constraints of the co-mapped applications. This scenario is the worst-case scenario for our technique, and yields an overhead of 1% of the average execution time of the applications.

VII. Conclusion

This paper has investigated for the first time the trade-off between temperature and cache contention that arises on clustered many-core processors where applications are subject to performance constraints. We exploited this trade-off by determining the application-to-cluster, thread-to-core mapping, and the V/f levels of the clusters. To capture the complex characteristics of shared resource contention, we trained an NN model at design time that can predict slowdown of co-mapped applications at runtime. This model enabled *TcRM* to meet the performance constraints of all applications, while significantly reducing the chip temperature compared to the state of the art.

References

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

[2] J. L. Manferdelli, N. K. Govindaraju, and C. Crall, "Challenges and Opportunities in Many-Core Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 808–815, 2008.

[3] H. Khdr, H. Amrouch, and J. Henkel, "Aging-Constrained Performance Optimization for Multi Cores," in *Design Automation Conference (DAC)*, 2018, pp. 1–6.

[4] X. E. Chen and T. Aamodt, "Modeling Cache Contention and Throughput of Multiprogrammed Manycore Processors," *IEEE Trans. on Computers (TC)*, vol. 61, no. 7, pp. 913–927, 2011.

[5] (2020) AMD "Zen 3" Core Architecture. [Online]. Available: https://www.amd.com/en/technologies/zen-core-3

[6] T. Marinakis, S. Kundan, and I. Anagnostopoulos, "Meeting Power Constraints While Mitigating Contention on Clustered Multiprocessor System," *IEEE Embedded Systems Letters (ESL)*, vol. 12, no. 3, 2019.

[7] J.-H. Liao, H.-R. Chen, and Y.-S. Chen, "A Cache Contention-aware Run-time Scheduling for Power-constrained Asymmetric Multicore Processors," in *Research in Adaptive and Convergent Systems (RACS)*, 2020.

[8] S. C. Woo, M. Ohara, E. Torrie *et al.*, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Int. Symp. Computer Architecture (ISCA)*, 1995.

[9] D. G. Feitelson and L. Rudolph, "Metrics and Benchmarking for Parallel Job Scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1998.

[10] M. Rapp, H. Amrouch, Y. Lin *et al.*, "MLCAD: A Survey of Research in Machine Learning for CAD Keynote Paper," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

[11] Y. G. Kim, M. Kim, J. Kong, and S. W. Chung, "An Adaptive Thermal Management Framework for Heterogeneous Multi-Core Processors," *IEEE Trans. on Computers (TC)*, vol. 69, no. 6, pp. 894–906, 2020.

[12] D. Hackenberg, R. Schöne, T. Ilsche *et al.*, "An Energy Efficiency Feature Survey of the Intel Haswell Processor," in *Int. Parallel and Distributed Processing Symp. Workshop (IPDPSW)*. IEEE, 2015, pp. 896–904.

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

[14] S. Dey, E. Z. Guajardo, K. R. Basireddy *et al.*, "EdgeCoolingMode: An Agent Based Thermal Management Mechanism for DVFS Enabled Heterogeneous MPSoCs," in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*. IEEE, 2019, pp. 19–24.

[15] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," in *Design Automation Conference (DAC)*, 2015, pp. 1–6.

[16] X. Wang, A. K. Singh, B. Li *et al.*, "Bubble Budgeting: Throughput Optimization for Dynamic Workloads by Exploiting Dark Cores in Many Core Systems," *IEEE Trans. on Computers (TC)*, vol. 67, no. 2, 2017.

[17] L. Subramanian, V. Seshadri, A. Ghosh *et al.*, "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory," in *International Symposium on Microarchitecture (MICRO)*, 2015.

[18] N. Mishra, J. D. Lafferty, and H. Hoffmann, "ESP: A Machine Learning Approach to Predicting Application Interference," in *International Conference on Autonomic Computing (ICAC)*, 2017, pp. 125–134.

[19] H. Usui, L. Subramanian, K. K.-W. Chang, and O. Mutlu, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," *ACM Trans. on Architecture and Code Optimization (TACO)*, vol. 12, no. 4, 2016.

[20] S. Kundan and I. Anagnostopoulos, "A Machine Learning Approach for Improving Power Efficiency on Clustered Multi-Processor System," in *International Symposium on Circuits and Systems (ISCAS)*, 2020.

[21] W. Huang, S. Ghosh, S. Velusamy *et al.*, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[22] S. Boyd-Wickizer, H. Chen, R. Chen *et al.*, "Corey: An Operating System for Many Cores," in *Symp. Operating System Design and Implementation (OSDI)*, 2008.

[23] A. Pathania and J. Henkel, "HotSniper: Sniper-Based Toolchain for Many-Core Thermal Simulations in Open Systems," *IEEE Embedded Systems Letters (ESL)*, 2018.

[24] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *High Performance Computing, Networking, Storage and Analysis (SC)*. ACM, 2011.

[25] S. Li, J. H. Ahn, R. D. Strong *et al.*, "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing," *ACM Trans. Arch. and Code Opt. (TACO)*, 2013.