FLEA - FIT-Aware Heuristic for Application Allocation in Many-Cores based on Q-Learning

Iaçanã Ianiski Weber, Vitor Balbinot Zanini, Fernando Gehm Moraes
School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
iacana.weber@pucrs.br, vitor.balbinot@edu.pucrs.br, fernando.moraes@pucrs.br

2023 XIII Brazilian Symposium on Computing Systems Engineering (SBESC) | 979-8-3503-9410-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/SBESC60926.2023.10324296

***Abstract*—This paper introduces FLEA, a novel allocation technique for many-core systems that uses Q-learning to improve system reliability based on Failure In Time (FIT) monitoring. FLEA considers task energy consumption and thermal behavior between neighbor processing elements (PEs) to determine the most suitable task allocation that minimizes PE FIT. Through a design phase that learns by trials, FLEA creates the Q-table. FLEA allocates and migrates tasks at runtime by consulting the Q-table, bypassing the need for executing complex heuristics. Compared to state-of-the-art allocation techniques, the results show that FLEA, in addition to increasing the MTTF, the mapping and migration heuristics reduce the thermal amplitude, peak temperature, and spatial thermal distribution.**

***Index Terms*—Lifetime Management, Q-Learning, Many-cores, Application Allocation.**

I. Introduction and Related Work

The continually increasing capability of on-chip integration resulted in integrated circuits containing hundreds of processing elements (PEs). Examples of such integrated circuits include Celerity [1], which incorporates 496 RISC-V cores into a single die, and ET-SoC-1 [2], which surpasses one thousand PEs. However, due to the high power density resulting from the numerous PEs, the chip temperature increases, thus reducing the system lifetime reliability caused by aging effects. Examples of such aging effects are Electromigration (EM), Stress Migration (SM), Time-Dependent Dielectric Breakdown (TDDB), Thermal Cycling (TC), and Negative Bias Temperature Instability (NBTI), the effects of which are closely linked to the chip thermal behavior [3, 4].

The key to delivering performance without violating the Thermal Design Power (TDP) is the efficient management of system resources. One solution, *dark silicon*, keeps parts of the chip powered off, thus creating spots where no heat is produced [5]. The patterning allocation heuristic interleaves active and dark PEs to cope with the dark silicon restrictions [6]–[8]. This solution has proven effective, as dark cores placed near active cores can help with heat dissipation through thermal conduction. However, such approaches typically bind a task to one core for the entire task lifetime without mechanisms to avoid overheating. To address this issue, various proposals in the literature adopted task migration techniques [9]–[13], which move tasks to cooler PEs when reaching a given threshold, such as temperature or aging.

The authors would like to thank Imperas Software and Open Virtual Platforms for their support and access to their models and simulator. This work was financed in part by CAPES (Finance Code 001), CNPq (grant 309605/2020-2), and FAPERGS (grant 21/2551-0002047-4).

The development of self-aware dynamic thermal management (DTM) techniques, including the observe-decide-act paradigm (ODA) [14], overcome the limitations of the pattern-based allocation heuristic. Da Silva et al. [15] proposed a Proportional, Integral, and Derivative (PID) technique to periodically generate a score for each PE based on its temperature, which is then used to select suitable candidates for allocation or migration. However, this approach requires the system manager to continually compute the PID score, leading to scalability issues.

The literature presents proposals that adopt machine learning (ML) techniques for Dynamic Thermal Management (DTM), such as Reinforcement Learning (RL). Das et al. [16] employ Q-learning to generate a Q-table indexing system states and actions. However, the action space grows exponentially with the system size, making the proposal unscalable. Rathore et al. [17] proposed a fixed action space technique utilizing average power and temperature to classify applications to address this scalability issue. The RL trains a Q-table to decide if a given application should be executed on a high- or low-frequency core to maximize system health.

The **original contribution** of this work is the **F**IT-aware **L**earning H**e**uristic for application **A**llocation (FLEA), including task mapping and task migration. The proposed heuristic adopts a policy table trained at design time using Q-learning. The policy table learns the best patterns to deploy tasks to maximize system reliability based on the Failure In Time (FIT) monitoring.

The results of this work compare FLEA with a standard mapping heuristic, patterning mapping, and PID DTM. In addition to providing temperature management, FLEA can reduce maximum FIT and the amount of computing power required to manage the system.

This paper is organized as follows. Section II presents the system model and the problem formulation. Section III presents the **original contribution** of this work, the FLEA proposal. Section IV evaluates FLEA in terms of temperature (distribution, maximum values, thermal maps) and FIT. Section V concludes this paper and points out directions for future work.

II. System Model and Problem Formulation

Our many-core architecture comprises a homogeneous set of PEs surrounded by peripherals such as shared memories and hardware accelerators. These PEs are interconnected via a 2-dimensional mesh Network-on-Chip (NoC). Each PE has its

processor, scratchpad memory, network interface with direct memory access (DMA) capability [18], and NoC router.

Each PE is identified by a unique address – $C_{x,y}$. Taking into account a $X \times Y$ mesh, the bottom-left PE address is $C_{0,0}$ where the System Manager (***MPE***) is allocated and the top-right PE, $C_{X-1,Y-1}$. The peripherals share the same address as the PE to which it is connected. The packet header has a field that differentiates packets that go to the local PE or the attached peripheral.

A. Management

All PEs execute the FreeRTOS kernel with a message-passing communication API. The ***MPE*** executes the task-to-core allocation (mapping and migration) heuristics and manages the applications running in the system. We adopt an observe-decide-act (ODA) management approach. Temperature monitoring (*observation*) is enabled through the process where each PE counts the number of flits traversing its local router ($F_{x,y}$), the number of memory accesses ($M_{x,y}$), and the number of executed instructions ($I_{x,y}$). We classify the instructions into K categories with similar energy consumption to simplify the processor energy estimation. Based on these counters, each PE periodically estimates its dynamic energy consumption ($E_{x,y}$) using Equation 1.

$$E_{x,y} = \sum_{k=0}^{K} (I_{x,y}[k] E_{inst}[k]) + M_{x,y} E_{mem} + F_{x,y} E_{flit} \quad (1)$$

where: $E_{inst}[k]$: average energy to execute a instruction from category k; E_{mem}: average energy cost to make a memory access; E_{flit}: average energy cost to transmit one flit.

Periodically, PEs send their energy consumption estimation to a hardware accelerator, TEA (Temperature Estimator Accelerator) [19], designed to estimate the temperature at runtime based on the MatEx [20] heuristic. We extended TEA by adding a Failure In Time (FIT) estimator, which indicates the number of failures that can occur within 10^9 hours of execution. The FIT estimator adopts the RAMP [21] model to compute the aging effects, which provides the individual failure rate for five different aging mechanisms, namely, Electromigration (λ_{EM}), Stress Migration (λ_{SM}), Time-Dependent Dielectric Breakdown (λ_{TDDB}), Thermal Cycling (λ_{TC}) and Negative Bias Temperature Instability (λ_{NBTI}). To model the failure of PEs, we consider the sum of these failure rates, since a single mechanism can cause a failure that compromises the PE behavior. Equation 2 gives the PE FIT.

$$FIT_{x,y} = \lambda_{EM_{x,y}} + \lambda_{SM_{x,y}} + \lambda_{TDDB_{x,y}} + \lambda_{TC_{x,y}} + \lambda_{NBTI_{x,y}} \quad (2)$$

The temperature and FIT of each PE computed by TEA are sent to the ***MPE***, which takes *decisions* related to the system management. The ***MPE*** then sends decision messages to the PEs. Upon receiving these messages, the PEs execute the requested *actions* to meet the ODA constraints, such as migration or DVFS actuation.

B. Application Model

The workload is modeled as a set of multitask applications. Each application is characterized by a deadline, starting time, number of iterations, and period. The mapping function assigns every arriving application task to a PE. The application task is sent through the NoC from a peripheral called Application Repository to the selected PE. The migration function uses the temperature of each PE and current system usage to evaluate whether any allocated application task needs to migrate from its current location to another PE to maintain the system within temperature constraints.

C. Problem Statement

The *Mean Time to Failure* (MTTF) of an entity is the expected time of occurring the first system failure, measured in hours. Equation 3 computes the MTTF of each PE.

$$MTTF_{x,y} = \frac{10^9}{FIT_{x,y}} \quad (3)$$

The many-core system fails when any PE presents some failure and does not have fault-tolerant mechanisms. Equation 4 expresses the system MTTF.

$$MTTF_{sys} = \min_{\forall x \in [0,X-1], y \in [0,Y-1]} \{MTTF_{x,y}\} \quad (4)$$

The objective of this work is to develop mapping and migration heuristics that maximize $MTTF_{sys}$ while satisfying the temperature constraint $T_{x,y} \leq T_{th}\ \forall\ x \in [0, X-1], y \in [0, Y-1]$, where the threshold temperature (T_{th}) is defined by the system designer.

III. FIT-Aware Allocation

This section presents the **original contribution** of this work, named **F**IT-aware **L**earning **He**uristic for application **A**llocation (FLEA). We adopt the Q-learning [22, 23] algorithm as the Reinforcement Learning method due to its simplicity and the model-free feature, i.e., it does not require a model of the environment. FLEA adds a lookup table in the ***MPE***, named Q-table. The Q-learning populates the Q-table. The mapping and migration heuristics use the Q-table values (named Q-values).

The Q-table is a matrix with r rows and c columns used to select the location of a given task for the mapping and migration heuristics. Each row is addressed by the power consumed by the task, or "task power category" – TPC, according to Definition 1. Each column of the Q-table is indexed by a value defined by the thermal influence of the neighbors of a given PE, named PE bin-state – PBS, according to Definition 2. The Q-learning training phase determines the $r \times c$ Q-values.

When a task arrives for allocation or migration, during the inference phase, the ***MPE*** traverses the many-core, calculating the PBS for each $PE_{x,y}$. The ***MPE*** accesses the Q-table using the task TPC as row index and the $PE_{x,y}$ PBS as column index, adding the tuple {Q-value, $PE_{x,y}$ address} to a list. After the PEs have been traversed, the list is sorted, selecting the $PE_{x,y}$.

Definition 1. *Task Power Category* (**TPC**) - Tasks are grouped into discrete classes according to their average power profile. The power profile is obtained by running each application multiple times in the many-core without any other applications running in the system and without multitasking.

Figure 1 exemplifies the classification of a set of applications. The x-axis has tasks $\tau_{a,t}$, where a is the application index (this example has 7 applications), and t is the task index (each application has 2 to 8 tasks). The y-axis represents the average power consumption. The graph illustrates the mean power consumption in 38 tasks. These tasks are categorized according to their average power consumption into three TPCs: p_0, p_1, and p_2. Specifically: p_0: tasks that consume up to 0.2 W; p_1, tasks with consumption between 0.2 and 0.4 W; p_2 tasks consuming more than 0.4 W. These TPC groups were established using the k-means clustering technique. The reasoning behind this categorization is to cluster tasks with analogous power needs, thereby simplifying the overall model.

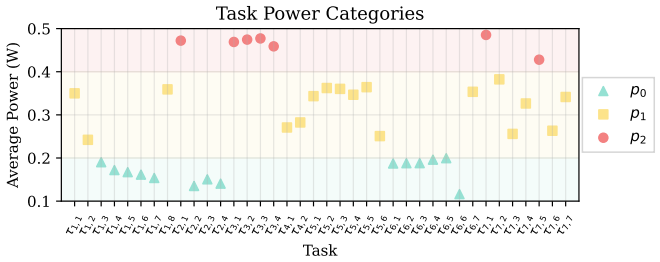

Fig. 1. Average power consumption of tasks $\tau_{a,t}$ (a: application index , t: task index). Tasks are classified into three categories, p_0 comprises tasks with average power consumption up to $0.2W$, p_1 tasks range from $0.2W$ up to $0.4W$ and p_2 includes tasks that consume over $0.4W$.

Definition 2. *PE bin state* (**PBS**) – index in the *bin state table* that represents the distribution of tasks within the sphere of influence around a given PE. The sphere of influence comprises the neighboring PEs surrounding a given $PE_{x,y}$: $PE_{x+1,y}$, $PE_{x,y+1}$, $PE_{x-1,y}$, and $PE_{x,y-1}$. These PEs represent the dominant source of heat that affects $PE_{x,y}$ [24].

The bin state table is defined at design time. It is constructed by generating the set of all possible combinations of TPCs within the sphere of influence of a given PE, discarding equivalent arrangements. With a sphere of influence considering four neighbor PEs and three TPCs, we have 35 possible PBS.

Table I presents the *PE bin state table*. The first three columns contain the sum of neighboring PEs for TPCs p_0, p_1, p_2. Given that we consider a maximum of 4 neighbors per PE, the maximum sum of these three columns is equal to 4. The fourth column is the PE bin state index, or PBS.

Figure 2(a) presents a many-core with 25 cores arranged in a 5x5 mesh. Each PE is colored gray if it is idle (not executing any task) or with a color according to its TPC. Each PE contains its address (x, y) and current PBS. Consider $PE_{2,2}$ in Figure 2(a). The PBS of $PE_{2,2}$ is 10 because $\{\sum p_0, \sum p_1, \sum p_2\} = \{0, 2, 1\}$. In Figure 2(b), we can observe the effect in PBS after the allocation of a new task with TPC $p1$ at $PE_{1,2}$, changing the $PE_{2,2}$ PBS from 10 to 13 (now $\{\sum p_0, \sum p_1, \sum p_2\} = \{0, 3, 1\}$).

During the training phase, the Q-learning algorithm updates its Q-values. Upon completing this phase, the Q-table contains the scores for the most favorable places to allocate tasks. It is worth noting that during the Q-table training, updating a specific score indexed by a {TPC, PBS} pair provides no information about other pairs, emphasizing the crucial role of exploring a large set of different mappings (the trained population) [25]. Equation 5 corresponds to the update function used during the training phase.

TABLE I
PE BIN STATE (PBS) TABLE

$\sum p0$	$\sum p1$	$\sum p2$	***PBS***	$\sum p0$	$\sum p1$	$\sum p2$	***PBS***
0	0	0	**0**	1	0	3	**18**
0	0	1	**1**	1	1	0	**19**
0	0	2	**2**	1	1	1	**20**
0	0	3	**3**	1	1	2	**21**
0	0	4	**4**	1	2	0	**22**
0	1	0	**5**	1	2	1	**23**
0	1	1	**6**	1	3	0	**24**
0	1	2	**7**	2	0	0	**25**
0	1	3	**8**	2	0	1	**26**
0	2	0	**9**	2	0	2	**27**
0	2	1	**10**	2	1	0	**28**
0	2	2	**11**	2	1	1	**29**
0	3	0	**12**	2	2	0	**30**
0	3	1	**13**	3	0	0	**31**
0	4	0	**14**	3	0	1	**32**
1	0	0	**15**	3	1	0	**33**
1	0	1	**16**	4	0	0	**34**
1	0	2	**17**	-	-	-	-

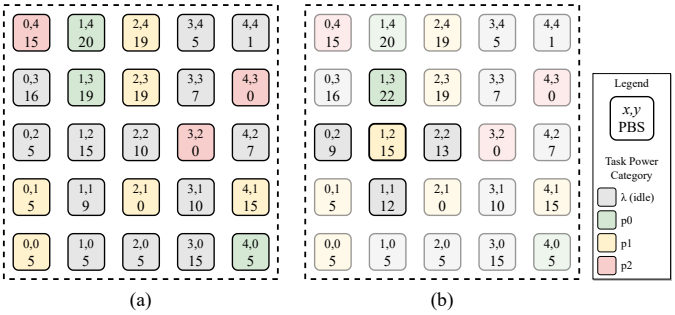

Fig. 2. PE bin state (PBS) examples. The PE color indicates the task power category (TPC) that is executing. (a) presents the PBS of each PE; (b) highlights the PBS change after the admission of a new application task with TPC $p1$ in the $PE_{1,2}$.

$$Q^{new}(p,s) \leftarrow (1-\alpha)Q(p,s) + \alpha[r(\Delta FIT_{x,y}) + \gamma \max_{P} Q(P,s)] \quad (5)$$

where the new value ($Q^{new}(p,s)$) for the pair TPC (p) and PBS (s) is given by the sum of three factors:

- $(1-\alpha)Q(p,s)$: represents the current value for the pair (p,s);
- $\alpha r(\Delta FIT_{x,y})$: the reward is a function of the FIT variation in the last evaluated period (weighted by the learning rate α).
- $\alpha\gamma Q(p,s)$: the maximum value of Q for tasks with the same TPC ($p = P$) (weighted by the learning rate α and discount factor γ).

Figure 3 illustrates the reward function, denoted as $r(\Delta FIT_{x,y})$. It is designed to boost the selection of bin states that lower the PE FIT. Specifically, bin states causing a decrease in FIT (x-axis) receive higher rewards (y-axis), promoting their corresponding Q-values. On the contrary, those who cause an increase in FIT get reduced rewards, maintaining their associated Q-values at lower levels.

Training occurs in the design phase using an epsilon-greedy learning policy [25], as depicted in Figure 4. As applications

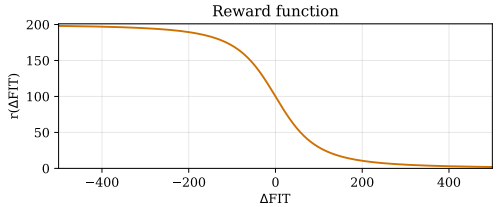

Fig. 3. The reward function $r(\Delta FIT)$ used in the Q-table training.

enter the system randomly, the ***MPE*** determines a mapping policy for each task. With an ϵ probability, the ***MPE*** allocates the task to a random PE; with a $1-\epsilon$ probability, the ***MPE*** accesses the Q-table to select the optimal PE. This involves computing the PBS for every available PE and then scanning the Q-table for the highest Q-value based on the TPC and PBS. Using the ϵ factor ensures comprehensive exploration of the state space. Once the location has been chosen, tasks are assigned to the designated PE. Every 1 ms, the platform evaluates each PE's temperature and FIT as outlined in Section II. Concurrently, the manager initiates the Q-table update. A PE is considered stable if its PBS remains unaltered over the preceding 50 ms, indicating that no tasks were placed within its sphere of influence. If stability persists over this duration, rewards are allocated based on FIT variation, leading to the computation of $Q^{new}(p, s)$, where s represents the PBS and p denotes the TPC executing in the PE (Equation 5).

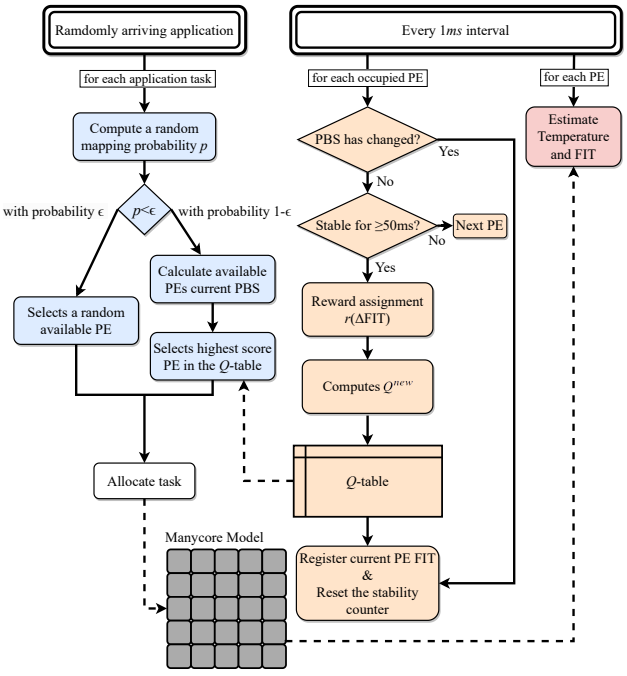

Fig. 4. Training flow diagram. The diagram is divided into three main processes; the Blue path presents the task-to-core selection process; the Orange path is the Q-table update process; and the Red presents the temperature and FIT estimation process.

The training is executed until the Q-table is stable. After completing the training, the Q-table is saved, and the system is ready to execute workloads using the learned Q-table in the mapping and migration heuristics.

A. Mapping

The mapping heuristic starts when an application arrives for execution on the many-core. The mapping heuristic has two main steps: (*i*) cluster creation and (*ii*) task-to-PE allocation.

Cluster creation seeks a continuous region with an amount of idle PEs (i.e., not executing any task) at least equal to two times the number of application tasks. This step ensures that the heuristic maintains tasks from the same application close to each other, thus reducing spatial fragmentation, which can cause performance degradation due to increased hop count and traffic congestion [26]. The cluster is not exclusively allocated to a single application, meaning that other tasks may already be running on the cluster. The selection of the cluster is based on which has the lowest temperature. However, in a many-core system where this information is unavailable at runtime, the heuristic may select a cluster based on a different criterion, such as the number of tasks already occupying the cluster.

After cluster selection, the ***MPE*** initiates the second step, repeated for each application task. Each available PE within the selected cluster has its PBS calculated, then a Q-value of each available PE is obtained from the Q-table, generating a score for each available PE. Finally, the heuristic allocates the task to the PE with the highest score. If more than one PE presents the same highest Q-value, the ***MPE*** selects the PE that minimizes the hop count between communicating tasks.

B. Migration

At each monitoring window (1 ms), the ***MPE*** receives temperature and FIT estimates for each PE from TEA. The migration heuristic executes periodically, actively looking for a thermal violation considering a pre-defined thermal threshold T_{th}. Upon detecting a thermal violation, the ***MPE*** identifies the PE with the temperature above T_{th}, selecting the task running on it for migration.

The algorithm initially expands the application cluster size, creating additional migration opportunities. Subsequently, the task is migrated to the PE with the highest Q-value within the coolest subset. Each migration implies an execution overhead for the application because the migration protocol requires multiple synchronization packets to transfer the task memory content to its new PE. Thus, a mapping heuristic minimizing the total number of migrations is essential to optimize the application performance.

IV. Results

We modeled the many-core using OVP [27], with 8x8 PEs (RISC-V cores) [28]. To obtain accurate energy values, we used RTL-level calibration. We evaluated the system with seven applications, ranging from two to eight tasks. The set of applications comprises parallel implementations of AES, Dijkstra, DTW, MPEG, and three synthetic applications. Figure 1 displays the TPC classification of the tasks. To determine whether the technique could accurately infer the behavior of an application not used in the training phase, we trained the Q-table at design time with a subset of six applications.

After the training phase, we performed experiments varying the system load. For example, a system load equal to 50%

means that half of PEs continuously execute tasks. We evaluated six allocation strategies:

1) *grouping*, heuristic that maps tasks close to each other, reducing the hop count between the same application tasks. Most mapping heuristics use this cost function;
2) *patterning*, interleaves a PE running a task with an idle PE, a dark-silicon approach that spreads the heat generation, avoiding thermal hotspot.
3) *PID* thermal management without task migration [15]. The PID technique periodically calculates a score for each PE. The score is a function of the PE temperature and is used by the mapping and migration heuristics to minimize the system temperature;
4) *PID+M* , PID with task migration;
5) *FLEA* without task migration;
6) *FLEA+M* with task migration.

Aiming for a thermal limit of 80°C we experimentally defined a threshold temperature (T_{th}) of 75°C to trigger the migration heuristic. Whenever a PE violates T_{th}, the ***MPE*** starts the migration heuristic, selectively migrating a particular application task from the area that presented the T_{th} violation.

Figure 5 presents the thermal distribution results with two system loads: 50% and 70%. From the figure, we observe:

- *Grouping*: mapping heuristics that do not consider the temperature in their cost function generates a large thermal amplitude (over 25°C) due to hotspot formation;
- *Patterning*, does not generate high temperatures, with peak temperatures reaching 74.60°C and 77.04°C for 50% and 70% load, respectively;
- *Patterning* and *PID* generate similar thermal amplitude (∼17°C) for both loads.
- *FLEA* presents a lower thermal amplitude, with 16.71ºC at 70% and 13,73ºC at 50%, which represent a 0.86°C and 3.85°C reduction, respectively;
- *PID+M* produced a reduction in peak temperature of 2.84°C and 3.39°C compared to PID.
- *FLEA+M* shows benefits mainly in the 70% load, reducing the peak temperature in 4.15°C. Expected result because in the 50% load, the *FLEA+M* average peak temperature was already below T_{th}.

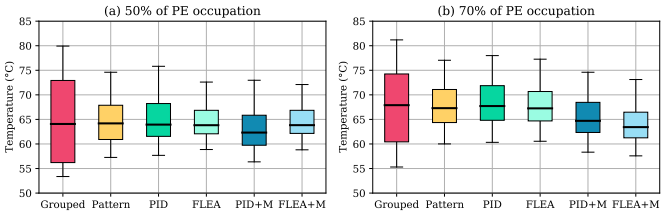

Fig. 5. Thermal distribution for two workload loads (50% and 70%), considering four mapping heuristics, two with and without migration.

Figure 6 shows the peak temperature for the 70% load for one second of simulation. The average peak temperatures are 81.19, 77.04, 74.60, and 73.12°C for *grouping*, *patterning*, *PID+M*, and *FLEA+M*, respectively. *PID+M* and *FLEA+M* successfully met the temperature constraint, with 20 and 41 task migrations, respectively. These migrations effectively prevented temperature violations. Notably, the FLEA+M heuristic sustained a system temperature of around 73°C.

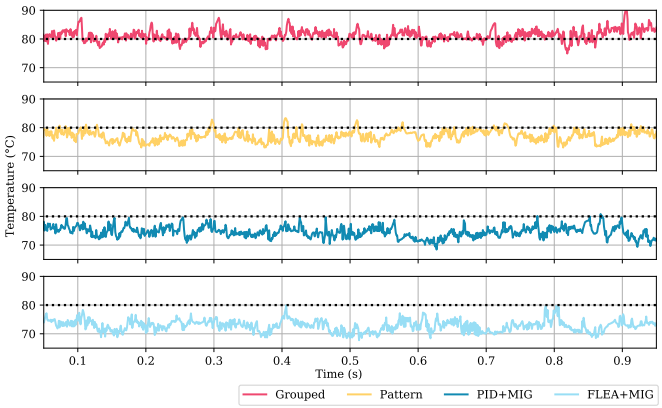

Fig. 6. Peak temperature behavior of many-core executing at 70% load.

Figure 7 illustrates the heat distribution in the many-core. The *grouping* method has the worst thermal distribution since tasks are allocated close to each other, leading to hotspots. The *patterning* is efficient in terms of thermal distribution, but a region in the center of the system is observed with higher temperatures. *PID* and *FLEA* without migration have similar behavior to *patterning*. However, when using task migration, the hottest regions do not occur. The figure also shows an advantage of *FLEA* over *PID* - the lower allocation complexity based on querying the Q-table. Note that the ***MPE***, at position (0,0), of the PID is always at a higher temperature, compromising the system reliability.

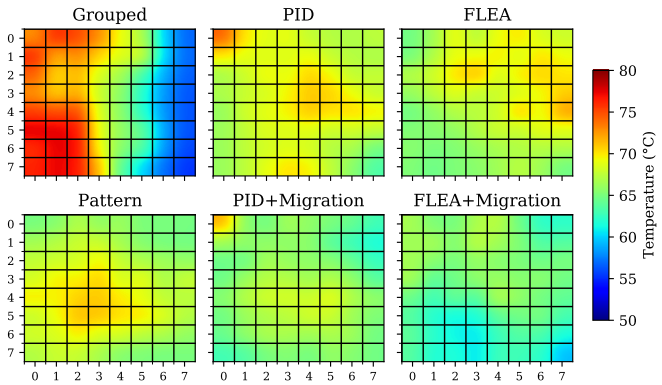

Fig. 7. Thermal distribution with a 70% load.

Figure 8 presented the FIT distribution of every PE for both load scenarios, 50%, and 70%. We may observe:

- In both scenarios, the *grouping* strategy presented the worst FIT distribution, with the highest FIT of $\sim$ 4000 representing an MTTF of 29 (50%,) and 27 years (70%).
- 50% load. The *pattern* algorithm presents the best FIT distribution, with a maximum FIT of 2169 representing an MTTF of 52 years. All other techniques present similar behavior, with a maximum FIT of 2913, 2786, 2812, and 2683 for PID, FLEA, PID+M, and FLEA+M, representing an MTTF of 39, 41, 41 and 43 years respectively.
- 70% load. *Patterning* no longer seamlessly allocates tasks, with a maximum FIT of 2763 (MTTF=41 years), a reduction of 11 years compared to the 50% load). With

this higher load (70%), FLEA takes advantage of learned patterns. The benefits are reflected in the maximum FIT for FLEA+M, which is 2343 and represents an MTTF of 49 years. The FLEA+M heuristic made 41 migrations, compared to 20 by PID+M, which presented a maximum FIT of 2946, representing 39 years of MTTF, a decade of difference.

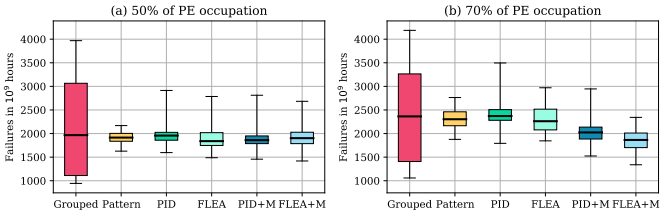

Fig. 8. PE's FIT after executing two workloads (50% and 70%), considering the allocation strategies.

Figures 5-7 reveal the effectiveness of runtime actuation, in this work achieved through task migration. PID and FLEA approaches reduced thermal amplitude, peak temperature, and spatial thermal distribution by using task migration triggered by monitoring temperature threshold violations. The patterning technique exhibits the highest MTTF for system utilizations up to 50% since it employs a technique that prevents heat propagation between neighboring PEs. However, this technique compromises application performance as the number of hops between communicating tasks increases [29]. Migration becomes essential to improve MTTF for higher system workloads, such as 70% (Figure 8). In this context, FLEA, specifically trained to optimize MTTF, achieves the expected result. It is also worth noting that FLEA allocates and migrates tasks by consulting a table (Q-table), bypassing the need for complex heuristics such as the PID approach. This shows that using Q-learning for task allocation in a many-core system is feasible and delivers superior results compared to other heuristics.

V. Conclusion and Future Works

This paper proposed and evaluated a new allocation technique, named FLEA, for managing task allocation and migration in many-core systems. FLEA uses Q-learning to optimize the allocation process by considering task energy consumption and thermal behavior between neighboring PEs. Our experiments demonstrated that the FLEA effectively reduces the average peak temperature of many-core systems, achieving temperature reduction compared to alternative mapping techniques. Moreover, FLEA decreased the many-core MTTF under higher system loads compared to pattern, PID, and grouping approaches. These results support our hypothesis that the management of the system considering the FIT improves system lifetime. Overall, this research contributes with a novel allocation technique that efficiently manages both energy consumption and thermal behavior in many-core systems.

Future works include applying and evaluating FLEA in systems with hundreds of processing elements (PEs) to identify limitations in the algorithm's scalability. Additionally, it is worth exploring DVFS for temperature control, applying and evaluating FLEA in multi-tasking scenarios, incorporating online learning, and assessing the impact of introducing a not-characterized application to the system.

References

[1] S. Davidson *et al.*, "The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips," *IEEE Micro*, vol. 38, no. 2, pp. 30–41, 2018.

[2] D. R. Ditzel and Esperanto team, "Accelerating ML Recommendation with over a Thousand RISC-V/Tensor Processors on Esperanto's ET-SoC-1 Chip," in *HCS*, 2021, pp. 1–23.

[3] J. Srinivasan *et al.*, "The Impact of Technology Scaling on Lifetime Reliability," in *DSN*, 2004, pp. 177–186.

[4] Jörg Henkel *et al.*, "Reliable on-chip systems in the nano-era: lessons learnt and future trends," in *DAC*, 2013, pp. 1–10.

[5] ——, "New trends in dark silicon," in *DAC*, 2015, pp. 1–6.

[6] W. Liu *et al.*, "Thermal-aware Task Mapping on Dynamically Reconfigurable Network-on-Chip based Multiprocessor System-on-Chip," *IEEE Transactions on Computers*, vol. 67, no. 12, pp. 1818 – 1834, 2018.

[7] H. Khdr *et al.*, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," in *DAC*, 2015, pp. 1–6.

[8] A. Kanduri *et al.*, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *ICCD*, 2015, pp. 573–580.

[9] S. Wen *et al.*, "Performance Optimization of Many-Core Systems by Exploiting Task Migration and Dark Core Allocation," *IEEE Trans. Computers*, vol. 71, no. 1, pp. 92–106, 2022.

[10] A. L. da Silva *et al.*, "Mapping and Migration Strategies for Thermal Management in Many-Core Systems," in *SBCCI*, 2020, pp. 1–6.

[11] V. Rathore *et al.*, "Performance Constraint-Aware Task Mapping to Optimize Lifetime Reliability of Manycore Systems," in *GLVLSI*, 2016, pp. 377–380.

[12] Z. Liu *et al.*, "Task Migrations for Distributed Thermal Management Considering Transient Effects," *IEEE Trans. Very Large Scale Integr. Syst*, vol. 23, no. 2, pp. 397–401, 2015.

[13] H. Mizunuma *et al.*, "Thermal coupling aware task migration using neighboring core search for many-core systems," in *VLSI-DAT*, 2013, pp. 1–4.

[14] N. D. Dutt *et al.*, "Toward Smart Embedded Systems: A Self-aware System-on-Chip (SoC) Perspective," *ACM Trans. Embed. Comput. Syst*, vol. 15, no. 2, pp. 22:1–22:27, 2016.

[15] A. Silva *et al.*, "Reliability Assessment of Many-Core Dynamic Thermal Management," in *ISCAS*, 2022, pp. 1590–1594.

[16] A. Das *et al.*, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *DAC*, 2014, pp. 170:1–170:6.

[17] V. Rathore *et al.*, "LifeGuard: A Reinforcement Learning-Based Task Mapping Strategy for Performance-Centric Aging Management," in *DAC*, 2019, p. 179.

[18] M. Ruaro *et al.*, "DMNI: A specialized network interface for NoC-based MPSoCs," in *ISCAS*, 2016, pp. 1202–1205.

[19] A. H. L. da Silva *et al.*, "Fine-grain temperature monitoring for many-core systems," in *SBCCI*, 2019, pp. 1–6.

[20] S. Pagani *et al.*, "MatEx: efficient transient and peak temperature computation for compact thermal models," in *DATE*, 2015, pp. 1515–1520.

[21] J. Srinivasan *et al.*, "Exploiting Structural Duplication for Lifetime Reliability Enhancement," in *ISCA*, 2005, pp. 520–531.

[22] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Hing's College, United Kingdom, 1989.

[23] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[24] G. M. Castilhos *et al.*, "A lightweight software-based runtime temperature monitoring model for multiprocessor embedded systems," in *SBCCI*, 2016, pp. 1–6.

[25] W. B. Powell, *Reinforcement Learning and Stochastic Optimization*. John Wiley & Sons, Hoboken, NJ, 2021.

[26] B. Li *et al.*, "On Runtime Communication and Thermal-Aware Application Mapping and Defragmentation in 3D NoC Systems," *IEEE Trans. on Parallel and Dist. Systems*, vol. 30, no. 12, pp. 2775–2789, 2019.

[27] "Open Virtual Platforms - the source of Fast Processor Models & Platforms," 2023. [Online]. Available: www.ovpworld.org

[28] I. Weber *et al.*, "A High-level Model to Leverage NoC-based Many-core Research," in *SBCCI*, 2022, pp. 1–6.

[29] X. Wang *et al.*, "Exploiting dark cores for performance optimization via patterning for many-core chips in the dark silicon era," in *NOCS*, Z. Lu *et al.*, Eds., 2018, pp. 17:1–17:8.