Mapping and Migration Strategies for Thermal Management in Many-Core Systems

Alzemiro Lucas da Silva*, André Luís del Mestre Martins†, Fernando G. Moraes*
*PUCRS – School of Technology, Porto Alegre, Brazil – alzemiro.silva@edu.pucrs.br, fernando.moraes@pucrs.br
†Sul-Rio-Grandense Federal Institute, IFSul, Charqueadas, Brazil – almmartins@charqueadas.ifsul.edu.br

***Abstract*—New technology nodes enable the integration of billions of transistors in a small silicon area by replicating identical structures, resulting in many-core systems. However, power density may limit the amount of energy the system can consume. A many-core at its maximum performance may lead to safe temperature violations and, consequently, result in reliability issues. Dynamic Thermal Management (DTM) techniques proposals guarantee that many-core systems run at good performance without compromising reliability. In this paper, we review recent DTM works, discussing their limitations, and propose new heuristics for thermal-aware application mapping and migration, using a hardware accelerator that enables temperature monitoring on systems with a large number of processing elements. Results show that using straightforward heuristics, with reactive actions based on runtime temperature monitoring, reduce the peak temperature in high workload scenarios (6.8%), and improve thermal distribution significantly on a large (8x8) many-core system.**

***Index Terms*—Many-core; resource management; thermal management; dark silicon; temperature estimation.**

I. Introduction

The steady transistor scaling and the increasing demand for performance led to the development of NoC-based many-core systems. Networks-on-chip (NoCs) provide enhanced performance and scalability for communication on systems with thousands of Processing Elements (PEs) [1]. The design of many-core systems takes advantage of the large number of transistors provided by recent CMOS technologies, exploring parallelism at the task-level to increase the overall system performance.

A significant challenge related to the design of many-cores in recent technology nodes is the increased power density, which causes the effect called *Dark Silicon* [2], where parts of the circuit need to be switched-off (dark) or underclocked to keep the system within the physical limits of power density and safe temperature. Although the number of PEs increases in many-core designs, the power consumed by each PE remains constant, since the threshold and supply voltage does not scale down with the size of the transistor as in previous technologies [3, 4].

The increase in power density also generates the effect of localized overheating, called hotspots. The dynamic temperature behavior can create reliability threats, increase power leakage, and add cooling costs [5]. The literature presents various many-core management techniques to deal with the increasing challenges imposed by the dark silicon phenomenon. For example, the management can frequently set PEs as active or dark at different moments to control the system power consumption under a power budget or below a critical temperature [6]. Recent proposals assume the system itself manages the adaptation, to increase its performance while keeping it within the physical limits of operation [7].

Monitoring system temperatures allow the implementation of Dynamic Thermal Management (DTM) techniques to ensure the operation within the specified limits, increasing reliability, reducing energy consumption, and possibly increasing lifetime [5]. Temperature monitoring is also a challenge in many-core designs since the number of sensors required to measure the temperature in a many-core system has several overheads, as their granularity, often measuring the temperature of a large system area instead of a PE area.

Recent DTM works propose thermal management strategies, using both software and hardware actuation. Most works focus on proactive approaches that require extensive knowledge of the applications executing in the system. Other works make use of complex thermal models or require advanced monitoring mechanisms, frequently omitted from the work description. However, system management should be able to deal with dynamic workloads, considering thermal-aware mapping and actuation to keep the system with a uniform thermal distribution.

In this context, the *goal* of this work is to explore mapping and migration heuristics to propose a thermal management solution. The heuristics use power and temperature monitoring, that can manage systems with dynamic workloads, offering the best performance within the physical limits of power density and temperature.

II. Related Work

Concerning thermal management strategies, the following topics are highlighted and discussed in the next paragraphs:

- *application profiling* is a data set to support management decisions derived from the applications' execution at design time (e.g., energy consumption, execution time);
- *workload behavior* is dynamic when applications (profiled or not) enter and leave the system at any time. Otherwise, the static workload is when the management is aware of the starting time of incoming applications at design time;

978-1-7281-9625-1/20/$31.00

- *actuation approach* is reactive when triggered after the occurrence of events, e.g., the temperature reaching a threshold in a given PE. Otherwise, proactive actuation strategies predict the occurrence of events and aim to avoid or mitigate them.

Profiling based thermal management [8, 9] execute a pre-characterized applications' set (profiling). The management strategies rely on profiling information extracted at design time. Therefore, profiling based thermal management requires a previous step of profiling to execute new applications (out of the original applications' set) and to keep the management strategies working accordingly. Whether an application without profiling has assigned to profiling based thermal management, the application profiling at runtime could enable the application execution. However, there are no evaluations of the impact in the execution time to perform an on-demand application profiling strategy. Furthermore, algorithms to execute profiling have high computational complexity, and the many-core is not in charge of performing the profiling since an external entity provides the profiling data ready to use [10].

Dynamic workload means that an unknown set of applications may enter and leave the system at any time [11]. The application mapping relies on the current resources utilization by other running applications, i.e., the incoming application is unlikely to share the same area of system with a running application. Assuming that mapping of an incoming application does not leverage running applications, mapping becomes more complex concerning thermal hotspots. Further, traffic issues are more likely to appear. Thermal management techniques [8] are not able to deal with the issues induced by dynamic workloads properly because they require a mapping review of running applications. Since most of the thermal managements employ mostly design time strategies [10, 12, 13], remapping all applications may require a strategy to pause the applications' execution and induce the system to a non-negligible overhead. Similarly, when an application leaves the system, it may create an opportunity for a better thermal distribution for the remaining running applications. To take advantage of this opportunity, design time based thermal management needs to remap the running applications.

Reactive actuation strategies, e.g., task migration and DVFS, require online monitoring to guide management decisions. Accurate thermal models present high computational complexity and data dependency. Due to the complexity of thermal models, system characterization and application profiling compose the temperature estimation for replacing thermal monitoring in thermal management [8, 9]. Therefore, thermal management strategies usually do not include reactive actuation because they would impact on temperature estimation [14]. For example, task migration is a management strategy to deal with traffic issues and perform occasional task remapping with low overhead [15]. In the context of the dynamic workload of unknown applications, task migration is an essential strategy in thermal management.

This work stands out from related ones by employing reactive actuation strategies in the dynamic workload of an unknown applications' set. Accurate thermal monitoring supported by a hardware accelerator [14] enables the proposed management strategies. Our management strategies allow more flexibility and adaptability to deal with thermal hotspots by running effective and low-complexity computational algorithms.

III. Reference Architecture

Figure 1 presents the main components of the many-core system. The architecture is derived from a platform with two regions [16]: (*i*) a homogeneous set of PEs - $GPPC$ region; (*ii*) peripherals attached to the $GPPC$ borders. Peripherals may be dedicated hardware to inject new applications into a $GPPC$ (as Application Injector at Figure 1), or hardware accelerators. Each PE (SP, CM, GM) has a processor, a network interface, local memory, and the NoC router.

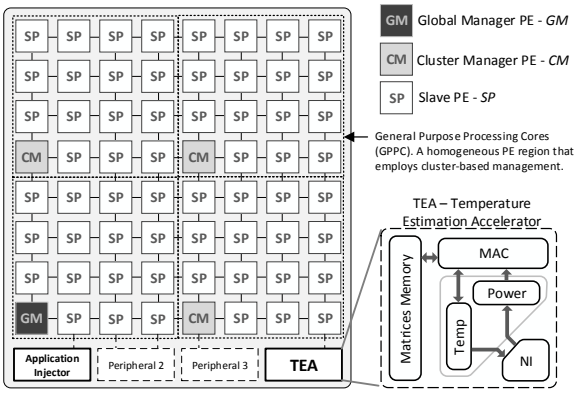

Fig. 1. NoC-based many-core system with peripherals (adapted from [16]).

The system management adopts a hierarchical organization, by partitioning the GPCC region in clusters, where each cluster has its manager PE. All PEs have the same hardware while the software in each PE may assume the following roles:

- Slave PE – *SP*: execute applications' tasks.
- Manager PE – *MP*: manage the SPs of a given cluster. MPs may be local to a given cluster (CM) or execute global actions besides the CM (GM). In this work, CM is in charge of mapping and migrating tasks and GM accumulates the application admission to the CM role.

This work uses the TEA (Temperature Estimation Accelerator) IP [14]. TEA implements in hardware a simplified version of the *MatEx* temperature estimation method [17]. TEA uses: (*i*) three single port memory blocks, responsible for storing the matrices derived from MatEx and temporary temperature values; (*ii*) a register bank to store the power values of each PE; (*iii*) a single 32-bit multiplier-accumulator (MAC); (*iv*) a network interface (NI).

The temperature monitoring scheme adopts a hierarchical scheme to minimize the control traffic in the NoC. At the lower level, SPs monitor their power consumption by observing the PE activity (executed instructions, transmitted flits, accesses to the local memory). At the cluster level, MPs receive the power samples of its corresponding SPs and update lookup tables.

Next, each MP creates a packet with the power information related to the cluster and transmits this packet to TEA. At the upper level of temperature monitoring, TEA receives the power consumed by all PEs and executes the temperature estimation procedure. Following, TEA sends a packet with the temperatures of all PEs to the GM. Finally, the GM receives this temperature message, stores the current temperature data, and transmits to the CMs the temperature of the SPs belonging to each cluster. This monitoring schemes adopt a fixed monitoring window for temperature estimation.

IV. DTM Actuation Policies

This Section presents three actuation policies used by the heuristics proposed in Section V: (*i*) application admission (Algorithm 1), (*ii*) task mapping (Algorithm 2), and (*iii*) task migration (Algorithm 3). These actuation policies avoid the presence of hotspots during the applications' execution.

When the Application Injector (Figure 1) interrupts the GM requesting the admission of a new application into the system, the first action is to select a cluster to execute the application, using a procedure called *application admission*. The GM selects a cluster based on the temperatures of all PEs, obtained from TEA, and the availability of resources in the cluster. Algorithm 1 details the application admission procedure.

Algorithm 1 Application Admission (executed at GM)

```
1: Inputs: app, cl_set
2: Outputs: cl_output
3: cl_output ← ∅,
4: coolestTemp ← ∞
5: if cl_set.freeResrc ≥ app.#tasks then
6:     for each cl ∈ cl_set do
7:         if cl.temp < coolestTemp then
8:             coolestTemp ← cl.temp
9:             cl_output ← cl
10:        end if
11:    end for
12: end if
13: return cl_output
```

Algorithm 1 receives as inputs: (*i*) the application task graph description – *app*; (*ii*) the set of clusters – cl_{set}. The algorithm starts by initializing the selected cluster as empty, cl_{output}, and sets the coolest temperature as infinite (line 3-4). Next, all clusters are verified to check the availability of *resources* to receive the application (line 5). A *resource* is a memory page assuming a paged memory system. If there are no clusters with enough resources, the algorithm returns an empty value (line 13). Otherwise, the cluster with the lowest temperature is assigned as the output of the algorithm (lines 6-11).

If it is not possible to execute the application with the available resources in the cluster ($cl_{output} = \emptyset$), the MP may borrow resources from another cluster (a process named reclustering), increasing the cluster size, re-executing the application admission. In the last case, the application is enqueued for later execution.

The next step is to map the application's tasks in the select cluster. Algorithm 2 receives as inputs: (*i*) *app*; and (*ii*) $mapping\_prio$ – a list of SPs belonging to a given cluster sorted according to one of the prioritizing algorithms (Section V).

At line 2, Algorithm 2 sorts the application's tasks according to the communication dependence, i.e., traversing the CTG (communicating task graph) from initial tasks up to final ones. The main loop (lines 3-13) iterates on every task of the application. Line 6 selects the SP with the highest mapping priority. If the SP has resources to receive the task (line 7), the task is mapped into the SP (line 8). Otherwise, the search for resource goes to the next PE in $mapping\_prio$ (line 11). Note that Algorithm 2 always finds an SP for task mapping, because the Application Admission (Algorithm 1) ensured resources availability.

Algorithm 2 Task Mapping (executed at GM or CMs)

```
1: Inputs: app, mapping_prio
2: sortTasks(app)      ▷ sort app tasks according to the comm. dependence
3: for each task ∈ app do
4:     mapped ← false
5:     while task is not mapped do
6:         SP ← mapping_prio.top()
7:         if SP.#free_resources > 0 then
8:             map_task(task, SP)
9:             mapped ← true
10:        end if
11:        mapping_prio.pop()                ▷ next PE in mapping_prio
12:    end while
13: end for
```

Each time an MP receives a packet with temperature data from thermal monitoring, Algorithm 3 evaluates the need to migrate tasks. In general, a task migration occurs when a given SP is above a temperature threshold. If the cluster presented a recent task event (as task mapping or task migration), migration is temporarily blocked in such a way to wait for a steady state temperature (currently set to 20 monitoring windows).

Algorithm 3 Task Migration (executed at GM or CMs)

```
1: Inputs: mapping_prio, cl
2: for each SP_src ∈ cl do
3:     if SP_src.temp > Mig_Threshold then
4:         for each task ∈ SP_src.tasks do
5:             migrated ← false
6:             while task is not migrated do
7:                 SP_tgt ← mapping_prio.top()
8:                 if SP_tgt.#free_resources > 0 then
9:                     migrate_task(task, SP_src, SP_tgt)
10:                    migrated ← true
11:                end if
12:                mapping_prio.pop()        ▷ next PE in mapping_prio
13:            end while
14:        end for
15:    end if
16: end for
```

The inputs of Task Migration, Algorithm 3, are: (*i*) mapping priority – $mapping\_prio$; (*ii*) the monitoring data of SPs belonging to the cluster – cl, obtained from TEA. Algorithm 3 starts a search in all SPs of the cluster (line 2). If there are SPs with temperature above the $Mig\_Threshold$ (line 3), all tasks running on the SPs, denoted as SP_{src}, must migrate to

another SP, denoted as SP_{tgt} (lines 4-15). The search of SP_{tgt} to receive the migrated task from SP_{src} (lines 6-13) follows *mapping_priority* order, similarly to the mapping algorithm. If SP_{tgt} has resources to receive a new task (line 8), the task is migrated from SP_{src} to SP_{tgt} (line 9). Otherwise, the search for a SP_{tgt} goes to the next PE in $mapping\_prio$ (line 12).

V. Thermal Management Strategies

Previous Section described algorithms to map and migrate tasks relying on an input called $mapping\_prio$. We define three strategies to prioritize the PEs of a cluster ($mapping\_prio$) described in the following Subsections.

A. Temperature PE Selection With Migration

Temperature PE Selection With Migration (TPESM) algorithm is a straightforward strategy that prioritizes the coldest PEs for mapping and migration. Algorithm 4 presents the main steps executed in this heuristic. Each time an MP receives a packet with temperature data (line 1), the procedure $generateMapPrio()$ creates a vector that sorts PEs by its temperatures (line 2). The procedure $generateMapPrio()$ is a generic function that receives scores and generates a vector of PE indexes ordered by the received scores, used by all proposed heuristics.

Algorithm 4 TPESM

1: $cluster\_temps \leftarrow$ **receive_packet**$(TEMP\_PERIF)$
2: $mapping\_prio \leftarrow$ **generateMapPrio**$(cluster\_temps)$
3: **migration**$(mapping\_prio, cluster\_temps)$ ▷ Algorithm 3

The execution of algorithm 4 is triggered by the reception of a temperature packet by a manager PE and the migration procedure runs every time a new temperature data arrives (line 3). However, the application admission and the task mapping procedures (Algorithms 1 and 2) are triggered by the injection of a new application in the system. In this case, the priority used in algorithm 2 is the last one generated when receiving a temperature packet.

B. Proportional, Integral and Derivative Temperature Management

TPESM is a strategy to balance the temperature distribution in a system and has been used by multi-core architectures with available temperature sensors [18, 19]. However, some mapping decisions using only the instant temperature value can impact the thermal distribution of the system. For example, if a task is mapped in a PE with the lowest temperature but it is increasing, the resulting peak temperature can be higher than if mapped in other PE with decreasing temperature. For this reason, we proposed the Proportional, Integral, and Derivative Temperature Management (PIDTM). The Proportional, Integral and Derivative temperature management should not be confused with PID from control theory, where the objective is to set the value of a signal equal to a reference value. In our algorithm, the objective is to avoid hotspots.

The *Proportional* value is related to the instantaneous temperature of a cluster or a PE, the *Integral* value is the average temperature value of a predefined number of monitoring windows, and the *Derivative* value is the tendency of the temperature value, and means how fast the temperature is changing.

Algorithm 5 shows the PIDTM computation score to generate the mapping priority vector. The *time_idx* entry corresponds to the number of times the function was called, increased after its execution. For each SP, there is a circular buffer, named *integral_buf*, which stores the last INT_WINDOW temperature samples. Every time a new temperature sample arrives, the algorithm computes the PID score for each SP in the cluster. Lines 4 and 5 replace the last temperature value in the circular buffer by the current SP temperature, while line 6 computes the average SP temperature in the last INT_WINDOW intervals. The derivative value is the subtraction of the previous temperature from the new temperature value (line 7). The final PIDTM score is the sum of the proportional value, which is the current temperature, the integral value and the derivative value, each one multiplied by a constant that defines the weight of each component of the sum in the final score (line 8). KP, KI, and KD are proportional, integral, and derivative constants, respectively. It is possible to tune those constants according to the control objectives. Finally, line 11 generates the mapping priority, used in task mapping and task migration procedures, as shown in algorithms 2 and 3.

Algorithm 5 PIDTM

1: **Inputs**: $time\_idx$
2: $cluster\_temps \leftarrow$ **receive_packet**$(TEMP\_PERIF)$
3: **for each** $SP_i \in cluster\_temps$ **do**
4: $last \leftarrow time\_idx$ **mod** INT_WINDOW
5: $integral\_buf_i(last) \leftarrow SP_i.temp$
6: $integral_i \leftarrow \frac{\sum_{n=1}^{INT\_WINDOW} integral\_buf_i(n)}{INT\_WINDOW}$
7: $derivative_i \leftarrow SP_i.temp - temperature\_prev_i$
8: $pid\_score_i \leftarrow KP * SP_i.temp + KI * integral_i + KD * derivative_i$
9: $temperature\_prev_i \leftarrow SP_i.temp$
10: **end for**
11: $mapping\_prio \leftarrow$ **generateMapPrio**(pid_score)
12: **migration**$(mapping\_prio, cluster\_temps)$ ▷ Algorithm 3

C. Temperature Management With Energy Constraint

The main drawback of TPESM and PIDTM strategies is the risk of always selecting the same PEs to provide a balanced temperature distribution in the system. Thus, a balanced temperature distribution may not always generate a balanced workload across all PEs in a cluster, which may stress some PEs while others are left unused. For this reason, we proposed the Temperature Management With Energy Constraint (TMEC) strategy, which takes the energy consumption of each PE to generate the score of mapping priority.

In this strategy, we add the energy consumption to the PIDTM score (Algorithm 5, line 8), since PIDTM tends to generate a better temperature distribution when compared with TPESM. The same procedures of task mapping and migration are used (Algorithms 2 and 3), based on the calculated

TMEC score, to keep a uniform thermal distribution during the execution of applications while balancing the workload between all PEs.

VI. Experimental Results

We compare our three thermal management strategies with a patterning mapping, and with a Multi-Objective Resource Management (MORM) approach [7]. Recent works on DTM use the patterning approach [12, 13] in a known applications' set, without the support of remapping by migrating tasks. MORM manages applications without taking temperature into account (it uses power consumption and performance instead) using task migration dynamically.

Directed acyclic task graphs model the applications, $A = (T, E)$, where the vertex $t_i \in T$ is a task and the directed edge $e_{ij} \in E$ is the communication between tasks t_i and t_j. The temperature monitoring window is set to 1 ms.

To create the evaluation scenarios, we used 7 applications: (*i*) DIJ (7 tasks), Dijkstra algorithm; (*ii*) AV (7 tasks), audio and video application that implements a video and audio decoding pipeline in parallel; (*iii*) AES (5 tasks), encryption algorithm; (*iv*) Sort (5 tasks), parallel quick sort algorithm; (*v*) MPEG (5 tasks), image decoder; (*vi*) DTW (6 tasks), Digital Time Warping algorithm; (*vii*) SYN (6 tasks), a communication intensive synthetic benchmark. Applications use MPI-like communication and are described in C language.

We generated the results based on four different scenarios in an 8x8 many-core system with 64 PEs divided into 4 4x4 clusters. Table I presents the applications used to create each scenario. The first scenario consists of a typical workload scenario, with a mixed set of long and short execution time applications, entering the system at different moments. The second scenario presents a high workload period, with more than one application executing simultaneously at each cluster. The third scenario is a regular workload with one application per cluster. The fourth scenario presents a low workload, considering one free cluster. Although the larger number of tasks in the dynamic workload than the remaining ones, the number of running tasks varies in the dynamic workload to present peaks and valleys of system utilization.

TABLE I
Thermal evaluation scenarios.

Scenario		Applications	#Tasks
1	Dynamic	MPEG, DTW, AV, AES, 2xSYN, 2xDIJ, 2xSort	57
2	High Workload	MPEG, DTW, AV, SYN, DIJ, Sort	35
3	Reg. Workload	DTW, AV, SYN, DIJ	26
4	Low Workload	MPEG, DTW, DIJ	18

Figure 2 compares the peak temperature (obtained with TEA) for the four scenarios, using the five heuristics. The peak temperature is the highest temperature that one core has achieved during the execution. MORM algorithm is the reference to compare other heuristics because, in MORM, the temperature is not an input of the mapping algorithm. In the patterning approach, the temperature is also not an input of the algorithm, but this approach is usually a reference to produce a better thermal distribution.

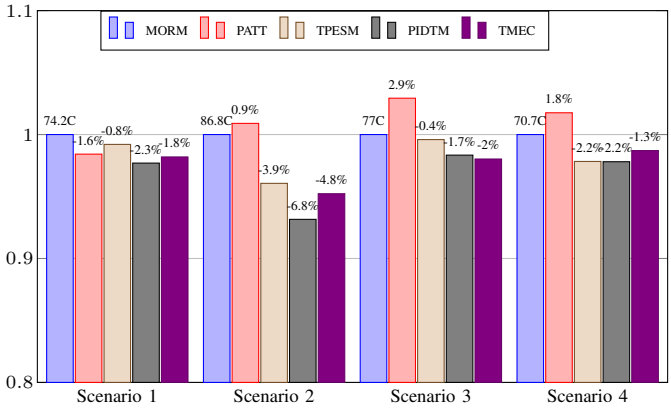

Fig. 2. MORM, Patterning, TPESM, PIDTM, TMEC peak temperature results.

Analyzing the results, we observed that the patterning approach produces peak temperature higher than MORM while achieving negligible improvements in overall average temperature. The reason for these results is mainly due to the hierarchical management used in MORM, which forces some physical separation between different applications among the PEs.

TPESM, which is a simple heuristic that always chooses the coldest PE, and the proposed PIDTM and TMEC, presented improvements in the thermal distribution when compared with MORM and patterning approaches. The PIDTM approach presented the best results in terms of peak temperature, losing to TMEC only on Scenario 3. In Scenario 2, high workload, the PIDTM got an improvement of 6.8% in the peak temperature relative to MORM algorithm, while TMEC got 4.8%. The reason that PIDTM gets better results than TPESM, is that this heuristic takes into consideration the change rate of the temperature, i.e., if the temperature is rising or decreasing during mapping or migration, and also the history of the temperature, which tends to balance the workload.

Although PIDTM tends to present better results than TMEC, we observed that in clusters with a low workload, some PEs were left unused during the simulation when using this algorithm. Then, to increase the workload distribution between all PEs, we argue that is also important to use energy consumption as an input of the algorithm, not only the temperature. For this reason, we consider that the TMEC algorithm is a good candidate when the focus is to reduce the probability of failures caused by the temperature stress in the same PE.

Figure 3 presents a visual comparison between the mapping heuristics, at different time snapshots, using Scenario 3, regular workload. Each slice represents one snapshot of temperatures in five different moments of the simulation, and each colored square represents one temperature. We observed that the different spacing obtained with MORM and patterning mapping approaches still produces hotspots. The proposed heuristics can react dynamically to the produced hotspots, producing a better thermal distribution, as observed in the fifth

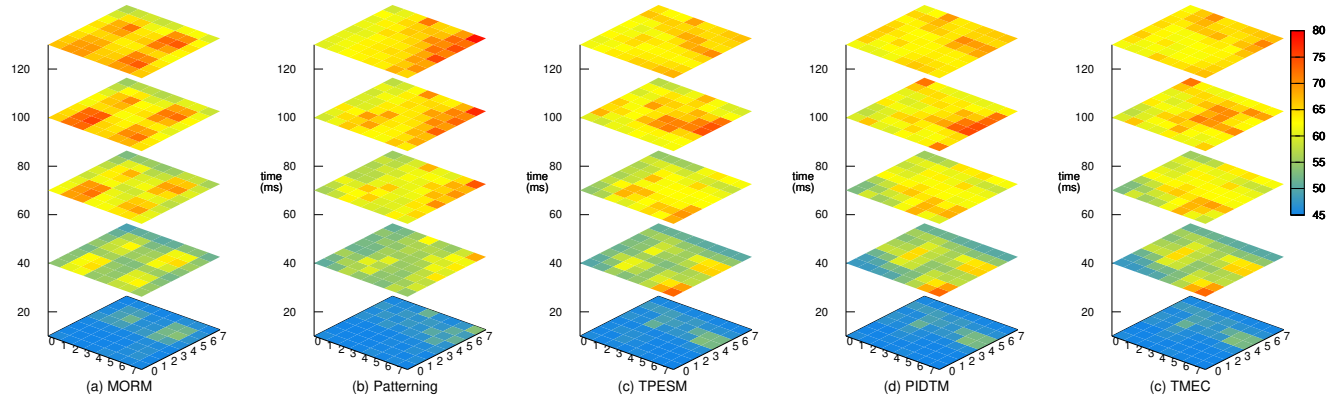

Fig. 3. Heatmaps comparison between different mapping and migration strategies.

slice of the figure. The figure also shows that TMEC produced a better result for this scenario.

VII. Conclusions

In this paper, we presented reactive mapping and migration strategies to provide a better thermal distribution in a many-core system supported by a runtime thermal estimation, considering a dynamic workload. Using simple thermal models to predict the future state of the system before a task mapping enables the use of proactive approaches but does not allow quick reactions to avoid hotspots during system execution. The proposed heuristics produced an improvement of up to 6.8% in the peak temperature of the system in a high workload scenario, which is a worst-case scenario for temperature management. The results show that a better thermal distribution provided by the proposed strategies also reduces the occurrence of hotspots when compared with state-of-the-art approaches.

Despite using a single hardware accelerator for temperature estimation, the proposed thermal management strategies use a hierarchical structure. Manager processors execute the proposed algorithms, which present a low computational complexity, do not affecting the overall system performance.

Future works include the application of DVFS to control the temperature distribution even further and the study of the performance benefits obtained with a better thermal distribution of the system.

Acknowledgment

Alzemiro Lucas da Silva is supported by CAPES (88887.184847/2018-00). Fernando Gehm Moraes is supported by FAPERGS (17/2551- 0001196-1) and CNPq (302531/2016-5).

References

[1] S. Borkar, "Thousand Core Chips: A Technology Perspective," in *DAC*, 2007, pp. 746–749.

[2] H. Esmaeilzadeh, E. Blem, E. S. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *ISCA*. IEEE, 2011, pp. 365–376.

[3] T. Iizuka, "CMOS technology scaling and its implications," in *Digitally-Assisted Analog and Analog-Assisted Digital IC Design*. Cambridge University Press, 2015, pp. 1–10.

[4] M. Bohr, "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007.

[5] M. El Ahmad, M. Najem, P. Benoit, G. Sassatelli, and L. Torres, "PoETE: A Method to Design Temperature-Aware Integrated Systems," *Journal of Low Power Electronics*, vol. 14, no. 1, pp. 1–7, 2018.

[6] H. Khdr, M. Shafique, S. Pagani, A. Herkersdorf, and J. Henkel, "Combinatorial Auctions for Temperature-Constrained Resource Management in Manycores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1605–1620, 2020.

[7] A. L. Martins, A. H. L. da Silva, A. M. Rahmani, N. Dutt, and F. G. Moraes, "Hierarchical adaptive Multi-objective resource management for many-core systems," *Journal of Systems Architecture*, vol. 97, pp. 416–427, 2019.

[8] M. Li, W. Liu, L. Yang, P. Chen, and C. Chen, "Chip Temperature Optimization for Dark Silicon Many-core Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 941–953, 2018.

[9] S. Sha, W. Wen, S. Ren, and G. Quan, "M-Oscillating: Performance Maximization on Temperature-Constrained Multi-Core Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 11, pp. 2528–2539, 2018.

[10] S. Pagani, H. Khdr, W. Munawar, J. Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal Safe Power - Efficient Power Budgeting for Many-core Systems in Dark Silicon," in *CODES+ISSS*, 2014, pp. 1–10.

[11] E. Carvalho, C. A. M. Marcon, N. Calazans, and F. Moraes, "Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs," in *SOC*, 2009, pp. 87–90.

[12] L. Yang, W. Liu, W. Jiang, M. Li, P. Chen, and E. H.-M. Sha, "Fotonoc: A Folded Torus-like Network-on-chip Based Many-core Systems-on-chip In The Dark Silicon Era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1905–1918, 2017.

[13] W. Liu, L. Yang, W. Jiang, L. Feng, N. Guan, W. Zhang, and N. D. Dutt, "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.

[14] A. L. da Silva, A. Martins, , and F. G. Moraes, "Fine-grain Temperature Monitoring for Many-Core Systems," in *SBCCI*, 2019, pp. 1–6.

[15] M. Ruaro and F. G. Moraes, "Demystifying the Cost of Task Migration in Distributed Memory Many-Core Systems," in *ISCAS*, 2017, pp. 148–151.

[16] M. Ruaro, L. Caimi, V. Fochi, and F. G. Moraes, "Memphis: a framework for heterogeneous many-core socs generation and validation," *Design Automation for Embedded Systems*, vol. 23, no. 3, p. 103–122, Aug 2019.

[17] S. Pagani, J.-J. Chen, M. Shafique, and J. Henkel, "MatEx: Efficient Transient and Peak Temperature Computation for Compact Thermal Models," in *DATE*, 2015, pp. 1515–1520.

[18] R. Rao and S. Vrudhula, "Efficient Online Computation of Core Speeds to Maximize the Throughput of Thermally Constrained Multi-core Processors," in *ICCAD*, 2008, pp. 537–542.

[19] A. K. Coskun, T. T. Rosing, K. Whisnant, and K. C. Gross, "Static and Dynamic Temperature-aware Scheduling for Multiprocessor SoCs," *IEEE Trans. on VLSI Systems*, vol. 16, no. 9, pp. 1127–1140, 2008.