Fine-grain Temperature Monitoring for Many-Core Systems

Alzemiro Lucas da Silva
PUCRS - Porto Alegre, Brazil
alzemiro.silva@acad.pucrs.br

André Luís del Mestre Martins
IFSUL - Charqueadas, Brazil
almmartins@charqueadas.ifsul.edu.br

Fernando Gehm Moraes
PUCRS - Porto Alegre, Brazil
fernando.moraes@pucrs.br

ABSTRACT

The power density may limit the amount of energy a many-core 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 have been proposed to guarantee that many-core systems run at good performance without compromising reliability. DTM techniques rely on accurate temperature information and estimation, which is a computationally complex problem. However, related works usually abstract the temperature monitoring complexity, assuming available temperature sensors. An issue related to temperature sensors is their granularity, frequently measuring the temperature of a large system area instead of a processing element (PE) area. Therefore, the first goal of this work is to propose a fine-grain (PE level) temperature monitoring for many-core systems. The second one is to present a dedicated hardware accelerator to estimate the system temperature. Results show that software performance can be a limiting factor when applying an accurate model to provide temperature estimation for system management. On the other side, the hardware accelerator connected to the many-core enables the fine-grain temperature estimation at runtime without sacrificing system performance.

CCS CONCEPTS

• **Computer systems organization → System on a chip**;

KEYWORDS

Temperature, monitoring, many-core, peripheral, resource management, dark silicon.

ACM Reference format:
Alzemiro Lucas da Silva, André Luís del Mestre Martins, and Fernando Gehm Moraes. 2019. Fine-grain Temperature Monitoring for Many-Core Systems. In *Proceedings of 32nd Symposium on Integrated Circuits and Systems Design, Sao Paulo, Brazil, August 26–30, 2019 (SBCCI '19),* 6 pages.
https://doi.org/10.1145/3338852.3339841

1 INTRODUCTION

The steady transistor scaling and the increasing demand for performance led to the development of NoC-based many-core systems. A significant challenge related to the design of many-core architectures in recent technology nodes is the increased power density which causes the effect called *Dark Silicon* [4], where parts of the circuit need to be switched off or underclocked to keep the system within the physical limits of power density and safe temperature.

Monitoring the 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. Temperature monitoring is usually done by inserting on-die thermal sensors, which provide accurate measurements at a given point of the chip [3]. The total area required for a basic analog sensor is not significant [8]. However, the temperature distribution within a single processor is non-uniform and requires multiple sensors to provide an accurate measure. Accordingly, the power consumed for thermal sensor data can easily reach tens of watts [8].

Moving to many-core designs, with dozens of PEs, fine-grain temperature estimation is required to guide DTM techniques [7, 12, 20]. DTM proposals adopt at design time estimation tools as HotSopt [5] or MatEx [11] to provide data for simulated scenarios. At runtime, the presence of thermal sensors is assumed. However, even employing thermal sensors, it is very challenging to predict the future state of the system after a change in power consumption due to the lack of information on the thermal dissipation layers [18], evidencing the importance of running an accurate and efficient thermal model in a real system.

Two paths may be followed to estimate at runtime the temperature using tools as HotSpot or MatEx. The first one is to execute the estimation in a given processor of the system, and the second one is to use a hardware accelerator. Hardware accelerators are a trend in commercial SoCs to execute specialized functions due to their energy efficiency compared to software.

This paper has two *goals*. The first one is to present a fine-grain temperature monitoring method for many-core systems, estimating the temperature at the PE level. The second one is to propose a dedicated hardware accelerator to estimate the system temperature.

The main *original contributions* of this work are as follows:

- Adaptation of the MatEx heuristic to reduce its computational complexity and memory requirements without sacrificing accuracy (Section 5);
- A hardware accelerator - Thermal Estimation Accelerator (*TEA*) to be used as a peripheral for many-core systems, which executes the temperature estimation algorithm at runtime in a fast and energy efficient way (Section 6);
- Comparison between software and hardware estimations, showing that *TEA* may be used as a reference for DTM techniques (Section 7).

This paper is organized as follows. Section 2 reviews related works on thermal management and monitoring. Section 3 presents the baseline many-core architecture. Sections 4, 5 and 6 details the

SBCCI '19, August 26–30, 2019, Sao Paulo, Brazil

ACM ISBN 978-1-4503-6844-5/19/08…$15.00
https://doi.org/10.1145/3338852.3339841

main contributions: the fine-grain monitoring, the temperature estimation model and the temperature estimation accelerator. Section 7 presents the results and section 8 concludes the paper.

2 RELATED WORK

According to Sha et al. [16], DTMs can control the temperature of a many-core system by proactive or reactive approaches. Reactive approaches take actions when a given processor reaches a threshold temperature, while proactive strategies rely on a thermal model and previous characterization of applications to ensure that a given mapping decision is thermally safe. Previous works proposed online DTM techniques based on temperature sensor information to cope with dark silicon issues on multi-core processors [2, 14]. Recent works propose proactive DTM techniques using MatEx or similar thermal models to obtain transient and peak temperatures during the system operation to enable application mapping heuristics for large NoC based many-core systems [7, 20]. However, these works do not address the feasibility of running the thermal model at runtime in a real system to manage real-time mapping decisions.

More recent works tend to present higher core count, relying on thermal models to enable thermal monitoring of the system, considering the challenges imposed by the use of thermal sensors to provide fine-grain temperature monitoring for many-core systems with higher core count. The high computational complexity, the accuracy, and the data dependency of the thermal models make proactive approaches the design choice for most many-core DTM proposals (Table 1). Many proactive strategies available on the literature rely on analytical models (as integer linear programming methods) to take task mapping decisions, estimating the temperature based on a previously characterized applications' set and a known task-to-core mapping [6, 7]. Other proactive approaches focus on defining a schedule of high computing power and low energy periods that keep the safe temperature before the release of tasks [16, 17]. Besides, proactive DTM proposals require the computation of a complex thermal model and power consumption information, but details about how to compute the temperature and how to transmit temperature and power information across the system are missing. Accordingly, it is implicit the thermal samplings are generated externally to the system since this computation would negatively impact on the system performance.

Table 1: Related works classification.

Proposal	Thermal Management	Thermal monitoring
2000s [2, 14]	reactive	thermal sensors
Yang [20]	proactive	HotSpot/MatEx
Liu [7]	proactive	MatEx
Pagani [12, 13]	proactive	analytical/math
Sha [16, 17]	proactive	analytical/math
Li [6]	proactive	analytical/math
Castilhos [1]	reactive	simplified HotSpot
Yang [19]	proactive/reactive	MatEx
This work	reactive	software and accelerator

On the other hand, efforts to enable reactive DTM approaches on many-core systems using thermal models are facing the computational complexity problem [1, 11]. With the increase in the number of core, the run-time thermal estimation can be unfeasible due to the computational complexity. Another issue is the communication overhead due to the power sampling traversing the system to a centralized point for temperature calculation. Yang's proposal [19] focus on achieving application performance requirements in the most energy efficient configuration, but also considers a reactive approach if a threshold temperature is reached. However, the monitoring overhead was not analyzed. Despite the open challenges for developing reactive DTM for many-core systems, only a reactive approach can deal with dynamic workloads where unknown applications can enter and leave the system anytime.

We claim that the unpredictability of dynamic workload makes proactive approaches unfeasible for employing DTM. On the other hand, if the DTM knows when applications start and end (i.e., pre-characterized workload), proactive approaches relying on temperature prediction may work together with reactive DTMs as an option for enabling runtime DTMs.

Finally, it is required to detail how to perform thermal monitoring to enable DTM development. Thermal monitoring challenges are related to avoid interferences with the running applications, reduce the execution time of the thermal model, minimize the communication overhead, and keep the thermal model accuracy. This paper is the first work to focus on all these challenges related to thermal monitoring for enabling reactive DTMs.

3 REFERENCE ARCHITECTURE

Figure 1(a) presents the main components of the many-core system. The system contains two regions [15]: (*i*) a homogeneous set of processing elements (PEs) - *GPPC* region; (*ii*) peripherals attached to the *GPPC* borders. Peripherals may be dedicated hardware to inject new applications into the *GPPC* (as the Application Injector in the Figure), or hardware accelerators. Figure 1(b) presents the PE internal modules: a processor (CPU), a network interface (DMNI), local memory, and the NoC router (PS).

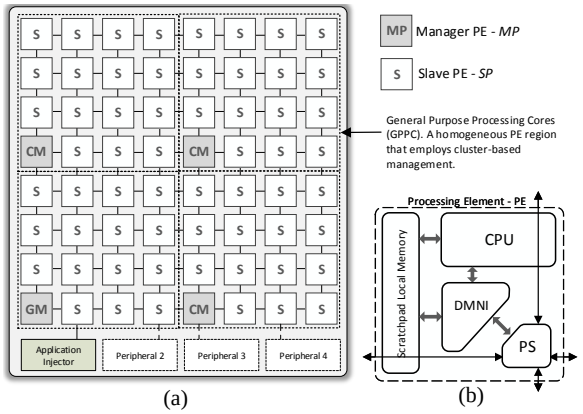

Figure 1: NoC-based many-core system with peripherals [15].

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, being the differentiation made in software. Each PE may assume the following roles:

- Slave PE – *SP*: execute applications' tasks.
- Manager PE – *MP*: manage the SPs of a given cluster, executing functions such as application admission, mapping, remapping, DVFS control. Manager PEs only execute management functions.

Manager PEs may be local to a given cluster (CM) or execute global actions besides the cluster management (GM).

As shown in Figure 1(a), peripherals are connected to the boundaries of the GPPC. Standard mesh topologies do not enable to send/receive packets to/from the NoC borders. Thus, to allow communication between PEs and peripherals, it is necessary to add hardware and software support. At the software level, a dedicated communication API creates packets with a flag in the header flit notifying the NoC that the packet should go to a peripheral (IO packet). The target of an IO packet is the router address, not the peripheral itself. At the hardware level, when the IO packet reaches the target router, it goes to the border port and not to the local port.

4 FINE-GRAIN MONITORING

The thermal information estimation for each PE requires a centralized computation due to the data dependency inherent of neighborhood temperature influence on the thermal models. Therefore, the power monitoring samplings from all PEs must be ready together before the temperature estimation. To avoid a penalty in the applications' traffic, the design of the power monitoring method is hierarchical [9] to induce little traffic in the NoC according to the architecture model previously presented.

Figure 2 overviews the hierarchical monitoring scheme, organized in three levels.

- SPs implement the lowest level of the monitoring scheme. Each SP monitors its power consumption by observing the CPU activity (power per instructions [9]), the number of memory accesses, and the number of flits transmitted by the router [10]. Periodically, SPs send the observed data, *power samples*, to its manager processor – MP (GM or CM).
- At the cluster level, MPs receive the power samples of its corresponding SPs and updates look-up tables. This procedure reduces the NoC traffic, by distributing the monitoring packets, and the processing load in the MPs, due to the smaller number of packets to treat. The MP creates a packet with the power information related to the cluster after receiving the monitored data from all SPs of its cluster and transmits this packet to the hardware accelerator - *TEA*.
- *TEA* receives the power consumed by all PEs through the packets sent by the MPs and executes the temperature estimation procedure. After computing the estimated temperature for each SP, *TEA* sends a packet with the temperatures of all SPs to the GM. The GM receives this "temperature message", storing the current temperature data, and transmits to the CMs the temperature of the SPs belonging to each cluster.

It is important to highlight that the temperature estimation must consider the power consumption in all PEs due to the thermal influence between PEs, since the power consumed by a given PE may affect the temperature of all other PEs due to the thermal conductance effect. For this reason, this work adopts a centralized approach to compute the temperature using a dedicated IP – *TEA*. If each cluster manager estimates the temperature of its SPs locally, it should exchange messages with other MPs to obtain the temperature and power consumption of the SPs in all other clusters, increasing the NoC traffic and making the process more complex.

The software implementation used to compare the performance with *TEA* (section 7), uses a similar monitoring method to the process previously presented. The difference is that the temperature estimation is executed in the GM. Instead of developing a hardware IP, software is in charge to estimate the temperature from the power samples.

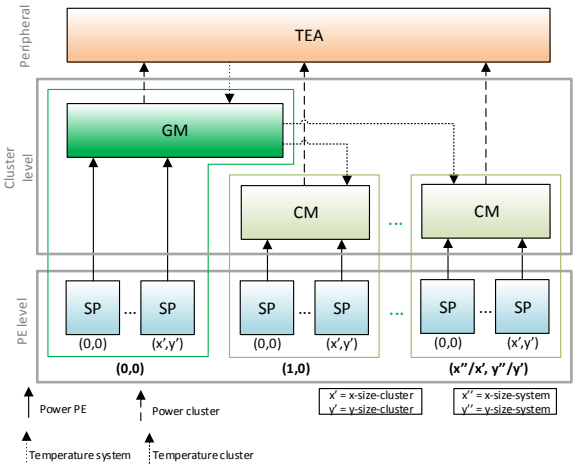

Figure 2: Hierarchical monitoring scheme.

5 TEMPERATURE ESTIMATION MODEL

MatEx [11] is the reference algorithm used by this work to build an accelerator that enables temperature estimation at runtime. MatEx uses a thermal model based on RC thermal networks, similar to HotSpot, relating temperatures in different areas of the chip with their power consumption. MatEx differentiates itself from HotSpot in the method used to solve first-order differential equations from RC networks. While HotSpot uses conventional numerical methods, MatEx solves the differential equations by using matrix exponentials and linear algebra. This method results in a polynomial time algorithm based on matrix multiplications that, similarly to graphics rendering, is executed faster and more efficiently with dedicated structures.

The system characterization process involves two main steps executed at design time: floorplanning and power consumption evaluation. The floorplanning of the system, described in Section 3, is a function of the technology node used to synthesize the PEs. The complete PE was synthesized using a 65nm technology, resulting in a 1 mm^2 area. The power consumption evaluation method [9] considers the CPU, memory and router consumption, main modules of the PE.

5.1 Model Adaptation

The open-source tool provided by MatEx authors' can estimate transient and peak temperatures after a power state change of the system, and the duration of power states can be variable. In this work, we compute transient temperatures at run-time, in a fixed interval of time, named *monitoring window*. To evaluate the adaptation of the MatEx algorithm to be executed by a hardware accelerator to estimate all transient temperatures of the system at run-time, we first present how the algorithm works for a generic floorplan and power state changes.

The RC model used to estimate the thermal behavior of the system contains N thermal nodes interconnected through thermal conductances. Each thermal node also has a thermal capacitance to consider the behavior of transient temperatures of each node. The ambient temperature (T_{amb}) is considered to be constant, and the power consumption of the active nodes are considered as heat sources. Equation 1 defines the thermal behavior of the system.

$$AT' + BT = P + T_{amb}G \tag{1}$$

where:

- $A = [a_{i,j}]_{N\times N}$, thermal capacitances between nodes i and j,
- $B = [b_{i,j}]_{N\times N}$, thermal conductances between nodes i and j,
- $T = [T_i(t)]_{N\times 1}$ vector with the temperature on every thermal node,
- $T' = [T'_i(t)]_{N\times 1}$ vector with the first order derivative of the temperature at time t,
- $P = [p_i]_{N\times 1}$ vector with the power consumption on every thermal node,
- $G = [g_i]_{N\times 1}$ vector with the thermal conductance between every thermal node and the ambient temperature.

The thermal model used in HotSpot and replicated in MatEx is that the number of thermal nodes (N) is greater than the number of processing elements of the system, as the heat dissipation structure is also modeled. Each PE results in 4 thermal nodes: the first is the circuit itself that considers its power consumption as a heat source, the second is the silicon substrate and interface material, the third is the heat sink, and the fourth is the heat spreader[5]. The model also considers that the ambient temperature interacts with the three top layers of the system in their four sides, adding 12 more thermal nodes to the model. Therefore, a system modeled with 16 PEs results in a model with 76 thermal nodes, a system with 36 PEs generates 156 thermal nodes and so on, increasing significantly the amount of memory required to hold the matrices used by the algorithm.

The first step executed by the model is the computation of steady-state temperatures for a given power vector of the active thermal nodes. The steady state temperatures depend only on the thermal conductances, once the capacitances are used to model the transient behavior of temperatures.

Once the steady-state temperatures (T_{steady}) are found, it is possible to calculate the transient temperatures using Equation 1. With a known initial temperature (T_{init}) and a steady state temperature previously estimated, the temperature estimation after t seconds can be found using the following equation:

$$T = T_{\text{steady}} + e^{Ct}(T_{\text{init}} - T_{\text{steady}}) \tag{2}$$

where: e^{Ct} is the exponential of matrix C at time interval t, and the matrix $C = -A^{-1}B$ is a derived from equation (1).

MatEx tool solves matrix exponentials using eigenvalues and eigenvectors and applying linear algebra. This procedure has high complexity ($O(n^3)$) but only needs to be executed once for a given chip. The matrix exponential e^{Ct} is constant for a fixed interval of t, and matrix C is the relationship between the capacitances and conductances of the system. In our implementation, the temperature monitoring is done at a fixed time interval, so the same matrix e^{Ct} can be used to compute all transient temperatures for each predefined time interval.

The resultant equations for temperature estimation in our approach are the following:

$$T_{\text{steady}k} = \sum_{j=1}^{N} b^{-1}_{k,j} \cdot p_j + T_{\text{amb}} \tag{3}$$

$$T_k(t) = T_{\text{steady}k} + \sum_{j=1}^{N} cexp_{k,j} \cdot (T_{\text{init}j} - T_{\text{steady}j}) \tag{4}$$

To apply this model it is necessary to extract the matrices $B^{-1} = [b^{-1}_{i,j}]_{N\times P}$ and $e^{Ct} = [cexp_{i,j}]_{N\times N}$ with a fixed monitoring period t from MatEx.

5.2 Integer Implementation

MatEx and HotSpot use double precision floating point variables to represent matrix values, power vectors, and temperatures. While double precision variables provide accuracy, they consume more memory and require more processing cycles or more complex hardware to execute.

A dedicated IP to estimate the temperature should have the following characteristics: fast memory access, small area footprint, and low power consumption. Memory is required to store B^{-1} and e^{Ct} matrices and should be local to the IP to enable fast memory accesses. The second and third characteristics come from the use of simple arithmetic functions. Thus, the IP implementation adopts integer variables instead of floating point representation. The execution time with integers had better performance in our experiments. On average, the execution time reduced by 46.6%.

Table 2 presents the temperature for the two representations in a 3x3 system, and the resulting error, for a monitoring period (epoch) equal to 1 ms, after 25 and 100 epochs. The power values applied to each PE are constant and do not vary along the time. As observed in Table 2, the maximum error introduced by the integer computation is 0.12 (0.24%) and 0.31 (0.58%) degrees for 25 and 100 epochs respectively.

Table 2: Temperature comparison using floating-point and integer representations.

	25 Epochs			100 Epochs		
PE#	double	integer	error%	double	integer	error%
1	55.2401	5534	0.18	56.8183	5703	0.37
2	52.0189	5212	0.19	53.9088	5416	0.47
3	59.5845	5970	0.19	61.6697	6194	0.44
4	51.5776	5169	0.22	53.6994	5396	0.49
5	56.3297	5644	0.20	58.3198	5857	0.43
6	49.8902	4998	0.18	51.5945	5181	0.42
7	48.3237	4843	0.22	50.1743	5042	0.49
8	58.6833	5880	0.20	60.9040	6119	0.47
9	51.1853	5131	0.24	53.6407	5395	0.58

These results show that an implementation using integers has a good cost-benefit ratio, considering the performance increase and the low error introduced. An important observation is that the relative error reduces after many epochs. The reason is that the power consumption in PEs changes along the time (variable workload), and the effect of the thermal capacitance is computed based on the steady temperature, which varies when the power consumption of a PE changes.

6 TEA - TEMPERATURE ESTIMATION ACCELERATOR

The *TEA* IP executes the following actions: (*i*) receives the power samples from the manager PEs (GM/CMs) periodically; (*ii*) at the end of an epoch, the IP computes equations 3 and 4 to obtain the current temperature of each PE; (*iii*) after updating the PEs' temperature, the IP sends a packet with the temperature values in the payload. Figure 3 presents a block diagram of *TEA*.

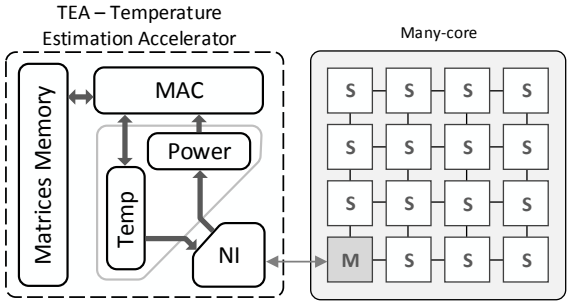

Figure 3: *TEA* architecture overview.

The IP has six main modules:

- Network interface (NI): handles the packet reception, identifying the sender and saving the power data in the correct location for the algorithm execution. The NI assembles a packet with temperature results, sending it to the GM;
- Temperature registers (Temp): store the steady and current temperature of the N thermal nodes;
- Power registers (Power): store the received power samples from the P PEs;
- Multiply-Accumulate (MAC): arithmetic module responsible for computing equations 3 and 4;
- Matrices Memory: stores matrices B^{-1} and e^{Ct};
- Two finite state machines (FSMs) to interact with the NI and MAC (not presented in the Figure).

The *TEA* area is a function of the number of PEs in the system. As the number of PEs increases, the size of the registers and the matrices memory increases. Table 3 presents the memory requirements according to the system size. The MAC and FSMs area are constant.

Table 3: *TEA* memory and register sizes.

System size	3x3	6x6	9x9	12x12
Number of PEs (P)	9	36	81	144
Thermal Elements (N)	48	156	336	588
Power registers (P)	9	36	81	144
Temperature registers ($2N$)	96	312	672	1,176
Total registers (32 bits)	105	348	753	1,320
Registers' size (bytes)	420	1,392	3,012	5,280
Matrices values ($N^2 + NP$)	2,736	29,952	140,112	430,416
Mem. size 32-bit words (KB)	10.69	117.00	547.31	1,681.31
Mem. size 7-bit words (KB)	2.00	25.59	119.72	367.79

The number of power registers is equal to the number of PEs since only active thermal nodes consume power. The number of temperature registers is a function of the number of thermal nodes since the temperature conduction in all layers of packaging and cooling needs to be calculated to provide an accurate model. The number of registers increases linearly with the system size ($R\# = 36 * P + 96$) .

However, the memory size increases quadratically due to the B^{-1} and e^{Ct} matrices. It is possible to adopt simple techniques to reduce memory size. For example, let's consider the discretization of the matrices' values into 128 ranges, resulting in a 512-byte vector (1 KB for both matrices). The last line of the Table presents the memory requirements for the matrices, considering the discretization methods. For a large system, e.g., 9x9, the memory requirement is smaller than the local memory of a PE (typically 128 KB).

As described in section 5, the temperature estimation algorithm respects a predefined monitoring window. *TEA* uses the same monitoring window, starting the temperature computation at the end of this period, being not necessary to wait for the reception of all power samples. As these packets are assembled in software, they can present different delays and arrive out of order due to congestion in the NoC. This procedure generates packets to the GM with a fixed periodicity.

Although the monitoring approach gives precise power data, *TEA* operation is independent of the measurement method. Other power estimations methods may be used, not discarding the presence of physical sensors.

7 RESULTS

The first set of results compare the performance of *TEA* against a software implementation on the same platform. This evaluation is important to justify the use of a dedicated IP for temperature estimation rather than a software implementation. For this purpose, we implemented two software versions of the algorithm: (*i*) centralized version, which runs on a processor of the platform (MP); (*ii*) distributed version, where each processor runs a service to compute its temperature. Table 4 shows the performance results, considering three system sizes.

Results show that *TEA* consumes 0.422 ms for a 10x10 system to estimate the temperatures, using a single MAC. As the monitoring window is set to 1 ms, this dedicated IP has enough performance to complete its task. As the algorithm is based on matrix-vector products, its complexity is quadratic. Thus, for larger systems, *TEA* may employ parallelism to execute the estimation to respect the monitoring window.

Table 4: Software and hardware performance comparison.

System size	3x3	6x6	10x10
Number of PEs	9	36	100
Thermal Elements	48	156	412
TEA (clock cycles)	2784	30108	211356
Time (μs@500MHz)	5.568	60.216	422.712
Centralized (clock cycles)	53681	661829	4805736
Time (μs@500MHz)	107.362	1323.66	9611.47
Distributed (clock cycles)	6867	19648	50368
Time (μs@500MHz)	13.734	39.296	100.736

The computation time for the centralized software implementation, in a 6x6 system size, requires 1.32 ms. As the monitoring window is 1 ms, it is not possible to use the centralized software implementation even for small system sizes. As the algorithm can be easily parallelized, each processor can compute its temperature, and the results show that the time required for this computation

is enough considering the monitoring window of 1 millisecond. However, to distribute the temperature computation, the communication overhead may be the limiting factor since each PE would need the transmit its energy consumption to all other PEs. Another issue related to the distributed implementation is that all processors of the system would spend some time running the algorithm, impacting on the performance of the applications running in the PEs. Considering a 10x10 system, each processor would spend about 10% of the execution time running the temperature estimation algorithm and an unpredictable amount of time to receive the power samples from all other PEs. *For those reasons, we argue that the dedicated IP is the best choice to perform a runtime temperature estimation.*

The second set of results was performed to analyze the *TEA* precision against the HotSpot model in the temperature estimation computation. Figure 4 shows the simulation results for a 6x6 system, where each square represents one temperature. The simulation consists of a typical utilization scenario, with tasks having a long execution time mixed with tasks that start and end in several moments. Five temperature snapshots were taken during the simulation, represented in the *z* axis in the figure. The power vectors of the simulation have been extracted and used as inputs for the HotSpot tool to produce figure 4(a). Figure 4(b) presents the results calculated by *TEA*.

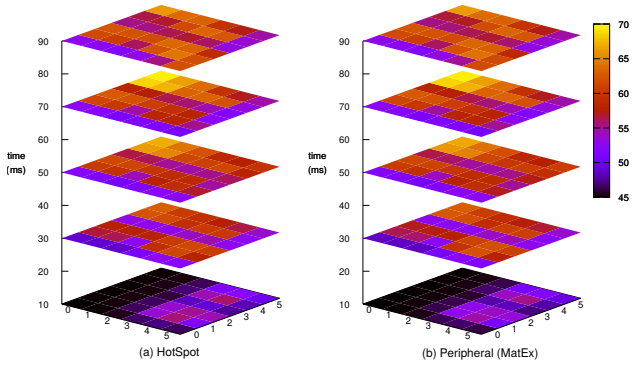

Figure 4: Typical scenario temperature results for (a) HotSpot and (b) proposed peripheral.

Analyzing the results graphically it is not possible to observe the error introduced by the proposed method. In fact, the average error is about 0.02 degrees and the maximum error observed in this simulation was 0.2 degrees.

8 CONCLUSIONS

This work addressed the four challenges related to temperature estimation addressed at the end of the related works Section. The first challenge regards thermal monitoring to enable DTM development. MatEx was adapted to reduce its computational complexity considering a fixed monitoring window and integer representation. The second challenge is related to intrusiveness, i.e., avoid interferences with the running applications. The solution to this issue was to move the complexity of the thermal estimation computation to a dedicated hardware accelerator. The third challenge, reduce the execution time of the thermal model is linked to the previous ones, simplification of the thermal model without sacrificing accuracy and the proposal of a hardware accelerator. Finally, the last challenge regards the minimizing of the communication overhead. This challenge was addressed by a hierarchical monitoring scheme, which reduces the monitoring traffic transmitted through the NoC.

Thus, the proposal of a hardware accelerator for temperature estimation, linked to a hierarchical power monitoring, is an important step towards reactive DTMs in large NoC-based many-core systems.

ACKNOWLEDGMENTS

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] G. Castilhos, F. G. Moraes, and L. Ost. 2016. A Lightweight Software-based Runtime Temperature Monitoring Model for Multiprocessor Embedded Systems. In *SBCCI*. IEEE, 1–6.
[2] Ayse Kivilcim Coskun, T. T. Rosing, Keith Whisnant, and Kenny C. Gross. 2008. Static and Dynamic Temperature-aware Scheduling for Multiprocessor SoCs. *IEEE Transactions on Very Large Scale Integrated Systems* 16, 9 (2008), 1127–1140.
[3] Mohamad El Ahmad, Mohamad Najem, Pascal Benoit, Gilles Sassatelli, and Lionel Torres. 2018. PoETE: A Method to Design Temperature-Aware Integrated Systems. *Journal of Low Power Electronics* 14, 1 (2018), 1–7.
[4] H. Esmaeilzadeh, E. Blem, E. St Amant, K. Sankaralingam, and D. Burger. 2011. Dark Silicon and the End of Multicore Scaling. In *ISCA*. IEEE, 365–376.
[5] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. 2006. Hotspot: A Compact Thermal Modeling Methodology for Early-stage VLSI Design. *IEEE Transactions on Very Large Scale Integration Systems* 14, 5 (2006), 501–513.
[6] Mengquan Li, Weichen Liu, Lei Yang, Peng Chen, and Chao Chen. 2018. Chip Temperature Optimization for Dark Silicon Many-core Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 5 (2018), 941–953.
[7] Weichen Liu, Lei Yang, Weiwen Jiang, Liang Feng, Nan Guan, Wei Zhang, and Nikil D Dutt. 2018. Thermal-aware Task Mapping on Dynamically Reconfigurable Network-on-Chip based Multiprocessor System-on-Chip. *IEEE Trans. Comput.* 67, 12 (2018), 1818–1834.
[8] Jieyi Long, Seda Ogrenci Memik, Gokhan Memik, and Rajarshi Mukherjee. 2008. Thermal Monitoring Mechanisms for Chip Multiprocessors. *ACM Transactions on Architecture and Code Optimization* 5, 2 (2008), 9:1–9:33.
[9] André LM Martins, Marcelo Ruaro, and Fernando G Moraes. 2015. Hierarchical Energy Monitoring for Many-core Systems. In *ICECS*. IEEE, 657–660.
[10] Luciano Ost, Guilherme M. Guindani, Fernando Gehm Moraes, Leandro S. Indrusiak, and Sanna Määttä. 2011. Exploring NoC-Based MPSoC Design Space with Power Estimation Models. *IEEE Design & Test of Computers* 28, 2 (2011), 16–29.
[11] Santiago Pagani, Jian-Jia Chen, Muhammad Shafique, and Jørg Henkel. 2015. MatEx: Efficient Transient and Peak Temperature Computation for Compact Thermal Models. In *DATE*. 1515–1520.
[12] Santiago Pagani, Jian-Jia Chen, Muhammad Shafique, and Jørg Henkel. 2015. Thermal-aware power budgeting for dark silicon chips. In *IGSC*. 1–6.
[13] S. Pagani, H. Khdr, W. Munawar, J. Chen, M. Shafique, M. Li, and J. Henkel. 2014. TSP: Thermal Safe Power - Efficient Power Budgeting for Many-core Systems in Dark Silicon. In *CODES+ISSS*. 1–10.
[14] R. Rao and S. Vrudhula. 2008. Efficient Online Computation of Core Speeds to Maximize the Throughput of Thermally Constrained Multi-core Processors. In *ICCAD*. 537–542.
[15] M. Ruaro, L. L. Caimi, V. Fochi, and F. G. Moraes. 2019. A Framework for Heterogeneous Many-core SoCs Generation. In *LASCAS*. IEEE, 89–92.
[16] Shi Sha, Wujie Wen, Ming Fan, Shaolei Ren, and Gang Quan. 2016. Performance Maximization Via Frequency Oscillation on Temperature Constrained Multi-core Processors. In *ICPP*. 526–535.
[17] Shi Sha, Wujie Wen, Shaolei Ren, and Gang Quan. 2018. M-Oscillating: Performance Maximization on Temperature-Constrained Multi-Core Processors. *IEEE Transactions on Parallel and Distributed Systems* 29, 11 (2018), 2528–2539.
[18] Shervin Sharifi, Dilip Krishnaswamy, and Tajana Simunic Rosing. 2013. PROMETHEUS: a Proactive Method for Thermal Management of Heterogeneous MPSoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 7 (2013), 1110–1123.
[19] Lei Yang, Weichen Liu, Nan Guan, Mengquan Li, Peng Chen, and HM Edwin. 2017. Dark Silicon-aware Hardware-software Collaborated Design for Heterogeneous Many-core Systems. In *ASP-DAC*. IEEE, 494–499.
[20] Lei Yang, Weichen Liu, Weiwen Jiang, Mengquan Li, Peng Chen, and Edwin Hsing-Mean Sha. 2017. 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* 28, 7 (2017), 1905–1918.