A Lightweight Software-based Runtime Temperature Monitoring Model for Multiprocessor Embedded Systems

Guilherme Castilhos, Fernando Gehm Moraes
PUCRS University, Computer Science Department
Porto Alegre, Brazil
guilherme.castilhos@acad.pucrs.br,
fernando.moraes@pucrs.br

Luciano Ost
University of Leicester, Department of Engineering
Leicester, UK
luciano.ost@leicester.ac.uk

***Abstract*—High-thermal variation and temperature operation can have a noteworthy impact on system performance, power consumption and reliability, which is a major and increasingly critical design metric in emerging multiprocessor embedded systems. Existing thermal management techniques rely on physical sensors to provide them with temperature figures to regulate the system's operating temperature and thermal variation at runtime. However, on-chip thermal sensors present limitations (e.g. extra power and area cost), which may restrict their use in large scale systems. In this regard, this paper proposes a lightweight software-based runtime temperature model, enabling to capture detailed temperature distribution information of multiprocessor systems at a negligible overhead. To validate the proposal, the model is embedded in a distributed-memory MPSoC platform described in RTL. Further, results show that the average absolute error of the temperature estimation, compared to HotSpot is smaller than 4% in systems with up to 36 processing elements.**

Keywords—MPSoC; NoC; monitoring; HostSpot; temperature estimation

I. INTRODUCTION

The technology scaling allied with growing processing capability of multiprocessor embedded systems, cause higher on-chip thermal variation and temperature operation. Managing thermal variation and temperature operation is key to accomplish a reliable and efficient system operation [1]. The higher on-chip temperature may lead to overall system performance degradation (e.g. energy efficiency), transient faults (e.g. occurrence of bit-flips) due to timing violations, as well as physical/permeant faults [2][3].

To balance at runtime temperature variation while satisfying power budget constraints of many-core systems, researchers have been considering different techniques like dynamic voltage and frequency scaling (DVFS), task scheduling, mapping, and migration. DVFS reduces high-temperature peaks by lowering the supply voltage and system operating frequency. Effects of application workload allocation on thermal system behavior have been investigated in several works [4][5], showing that load imbalance decisions can generate hotspots zones and consequent thermal implications, which may result in unreliable system operation. While task mapping and migration techniques aim to balance application workloads across multiple processing elements, task scheduling focus on local tasks/threads assignment optimizations to meet the temperature constraints.

Aforementioned techniques rely on, or assume, the existence of various on-chip temperature sensors to dynamic manage system temperature [6]. Although collecting real on-chip temperature values can be considered ideal, this approach presents several limitations. Physical sensors may be located away from hotspot zones, which impacts on both temperature measurement accuracy and response time [7]. To minimize such limitations, the use of numerous sensors is an option. As reported in [8], to achieve an efficient thermal management the IBM microprocessor combines 44 digital thermal sensors on a chip. Incorporating multiple sensors on the same die may reduce temperature underestimation and response delay, but it incurs extra costs regarding power and area, reducing its use in large scale systems that require detailed temperature distribution information at runtime.

The *goal* of the present work is to propose a lightweight software-based runtime temperature monitoring model, which can be used to collect detailed temperature distribution information of multiprocessor embedded systems without penalizing applications' execution time and system operation.

The main *contributions* of this work are the following:

- proposal of a software-based runtime temperature monitoring model that can be used for temperature management of large scale systems;
- validation of the proposed model by integrating it onto a cycle-accurate SystemC NoC-based MPSoC platform, considering several benchmark scenarios;
- comparison of the proposed model with a well-known temperature tool.

This work is organized as follows. Section II presents related works. Section III describes the proposed temperature evaluation flow. Section IV details the temperature heuristic. Experimental results including model accuracy and cost are given in Section V. Section VI concludes the paper and presents directions for the future works.

II. RELATED WORK

Beyond models and tools, like HotSpot [9], which allows thermal analysis at design time, researchers have been

978-1-5090-2736-1/16/$31.00 ©2016 IEEE

investigating techniques to manage on-chip system temperature and power budget constraints at runtime.

For instance in [4][5], tasks are mapped according to thermal condition of cores, which are collected at runtime by physical sensors. In [10], sensors are also employed to monitor the system temperature and based on collected information task migrations may be triggered, aiming to balance system temperature. A similar task migration scheme is proposed in [11], aiming to reduce hotspots on multi-core systems. Another work that targets multi-core thermal balance is presented in [12]. Wu et al. [13] present a temperature sensor-based DVFS, which allows to fine-tuning the system frequency operation according to its temperature condition. Such techniques rely on monitoring approaches, which provide necessary power/temperature information that is used to invoke a thermal management technique (e.g. task migration) when necessary. Underlying approaches consider the presence of on-chip sensors. Despite increasing chip power and area cost, physical sensors are vulnerable to noise and process variation.

In an attempt to replace or at least reduce the number of on-chip thermal sensors, a software-based approach is described in [14]. This approach extends the HotSpot model and uses physical performance counters available in the Pentium 4 to collect its power activity at runtime. The main limitation of this approach is the performance overhead related to the HotSpot temperature model computation. Results show that the performance overhead of this approach is more than 50% depending on the SPEC2000 benchmark profile.

Different from reviewed works, this paper proposes a purely software-based monitoring model, which has been integrated into a cycle-accurate NoC-based multiprocessor platform. The proposed model enables gathering detailed temperature distribution information of large scale multiprocessor systems with little runtime performance (less than 5%) and power (less than 4%) overheads. Further, due the low memory usage (less than 2 KB), the proposed approach can be easily integrated into other operating systems, which opens up new possibilities for enhancing dynamic temperature management of other multiprocessor platforms.

III. TEMPERATURE ESTIMATION FLOW

This Section describes the two-phase flow used to estimate the MPSoC temperature, at the processing element (PE) level. In this work, each PE includes a processor, a router, and a local memory. Fig. 1 illustrates the design-time and the runtime phases.

The *design time* phase consists in two steps:

(1) Energy calibration (‘1’ in Fig. 1). This step generates an *Energy Table*, which includes the energy cost of the processor and the NoC routers, for a given technology. Each PE stores this table. Section III.A details this step.

(2) Temperature calibration (‘2’ in Fig. 1). This step creates a temperature model using the Hotspot tool as the reference. As output, a *Thermal Table* with the thermal influences is generated. Section III.B details this step.

The *runtime phase* also contains two steps:

(1) Periodic power monitoring (‘3’ in Fig. 1). Each PE monitors its processor and router energy according to a parameterizable monitoring window. Monitored power values are sent to a manager processor (*M* in Figure 1).

(2) Runtime temperature estimation (‘4’ in Fig. 1). The manager processor receives the power dissipation of each PE and computes at runtime the current temperature of all PEs. Section IV details this step.

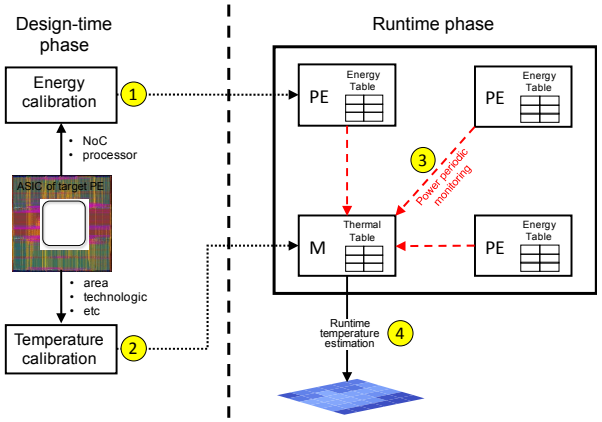

Fig. 1. Temperature Estimation Flow.

A. Processor and NoC Power Estimation

The energy consumption in a NoC-based multiprocessor system is mainly due to three components: processors, NoC (routers and links), and the memory. The current work does not consider the memory energy cost. This paper assumes an MPSoC that adopts scratchpad as local storage memory. In this case, the memory energy consumption per PE is similar, representing an offset of the consumed energy per PE.

As described in the literature [15], the energy consumption (EC) of a processor is defined by its static and dynamic consumption. The processor EC related to the execution of a given task is a function of the number of executed instructions. In our model, the energy cost of each instruction is determined from a gate-level implementation of the processor, as proposed by Rosa et al. [16]. In this model is accounted both static and dynamic consumptions.

Each processor contains an instruction analyzer module, which counts the number of executed instructions for different classes at runtime. Equation 1 computes the processor energy consumption for a given monitoring period.

$$E_{processor} = \sum_{i=0}^{8} energy(c_i) * total\_instructions(c_i) \quad (1)$$

where: $energy(c_i)$, energy to execute a given instruction belonging to the class c_i; $total\_instructions(c_i)$, number of executed instructions belonging to the class c_i.

The NoC EC is proportional to the number of transmitted flits at each router port [17]. A gate level description of the NoC is used to determine the energy consumption of the main router components: buffers, internal crossbar and control logic. Equation 2 gives the energy consumption for a given monitoring period.

$$E_{router} = nb\_flits * E_{buffer} + E_{crossbar} + E_{control\_logic} \quad (2)$$

where: nb_flits corresponds to the number of flits transferred by the router; E_{buffer}, $E_{crossbar}$, and $E_{control\_logic}$ to the energy consumption of the main router components.

Each PE monitors the processor and router energy according to a parameterizable *monitoring period*. Equation 3 computes the PE power dissipation (processor and NoC) for a given monitoring period.

$$P_{PE} = \frac{E_{router} + E_{processor}}{monitoring\ period} \quad (3)$$

B. Temperature Calibration

The temperature calibration uses as the reference the HotSpot tool [9], which provides an accurate thermal model widely used in the computer architecture research community. The worst case error values of HotSpot model for steady-state temperatures and transient temperatures is less than 5% and 7%, respectively.

HotSpot thermal characterization is produced according to a floorplan that represents the target circuit, considering physical properties and power dissipation of components (e.g. processors), modeled as an RC pair, on the die. In our case, HotSpot is used to capture the heat flow into and within the thermal package from each active PE, as well as the lateral heat flow between the PEs.

Using the HotSpot tool, it is applied the maximum power dissipation in a given PE, with a set of surrounding PEs. The goal of executing this simulation is twofold. First, to evaluate how power dissipation affects temperature over time. Secondly, to evaluate how the temperature of the PE under evaluation affects the neighbor PEs.

Fig. 2 illustrates the impact of the *target* PE temperature (i.e. PE under evaluation) on its direct neighbor PEs (*lateral* and *diagonal*), and on PEs far from the target PE (*other neighbors* PEs).

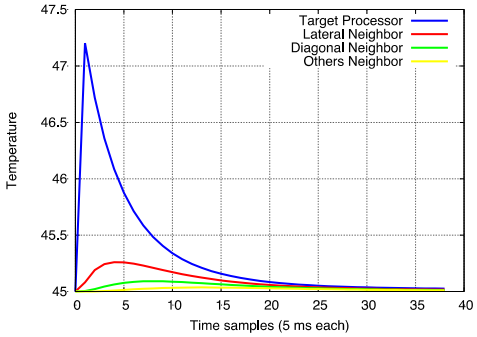

Fig. 2. Effect of temperature (0C) over time in a given PE and its neighbor PEs.

To create a temperature model, a set of assumptions are established to reduce the complexity of the model.

- Assumption 1. It is considered that the thermal influence of a processor affects only its direct neighbors. As Fig. 2 shows, the temperature of a given processor has a small impact in distant PEs.
- Assumption 2. The effect of the temperature decay (*thermal inertia*), as modeled by HotSpot, is too long. This work assumes that at the end of a given period (INERTIA_SIZE) the effect of the applied power ends. In our current model, this period is assumed as 100 ms, corresponding to 20 sampling windows of 5 ms.
- Assumption 3. As the temperature is linear with the power, it is possible to discretize the power values in intervals, assigning a temperature for each power interval.
- Assumption 4. Only integer values are used instead of floating point numbers. This assumption enables faster computation than using floating point numbers.

These assumptions may induce an error on the temperature estimation. The error of the model is evaluated in the Results section.

Using the above assumptions, it is created an LUT-based model. This LUT model comprises 3 *thermal matrices*. Each *thermal matrix* corresponds to the effect of the power over time for PEs in the boundary of the system (*lateral*), in the corners of the system (*diagonal*), or in the center of the system (*central*). Fig. 3 presents the placement of these PEs in a 6x6 system.

diagonal	lateral	lateral	lateral	lateral	diagonal
lateral	central	central	central	central	lateral
lateral	central	central	central	central	lateral
lateral	central	central	central	central	lateral
lateral	central	central	central	central	lateral
diagonal	lateral	lateral	lateral	lateral	diagonal

Fig. 3. Relative position of the PEs for the LUT-based model.

Fig. 4 presents an abstraction of the data structure employed for one *thermal matrix*. Each *thermal matrix* has 3 dimensions:

- *x* coordinate: power index, with the power discretized in 10 ranges according to the maximum power dissipated in the sampling window;
- *y* coordinate: thermal inertia (20 values);
- *z* coordinate: effect of the power on the temperature of the PE under evaluation and its neighbors (3 values). For *z*=0 it is the PE under evaluation, *z*=1 the lateral PEs and *z*=2 the diagonal PEs.

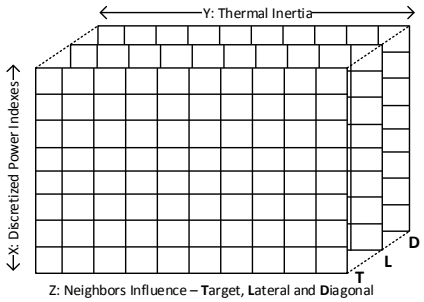

Fig. 4. Example of the data structure adopted for the *thermal matrices*.

Each thermal matrix requires 600 integers (10 * 20 *3), which represents a low memory usage (less than 2 KB) to store the set of tables to estimate the temperature.

IV. Temperature Estimation Heuristic

To compute at runtime the current temperature of all PEs, the heuristic must be as simple as possible, without incurring in inaccuracy. The basis of the temperature computation are the 3 *thermal matrices*, obtained in the offline phase.

The proposed heuristic adopts 3 data structures to compute the temperature of each processing element:

- temperature[|PE|]: vector with the temperature of each PE, all initialized at ambient temperature (45ºC).
- power[|PE|]: vector with the average power of each PE, obtained at the end of each monitoring window (5 ms in our current implementation). If a given PE does not send the monitoring packet, the average power in the period is assumed zero (for example, a processor may be in hold state).
- thermal_influence[INERTIA_SIZE][|PE|]: matrix used to store the effect of the temperature over time. It works as a circular FIFO.

To understand as the *thermal_influence* matrix is used, let's assume: INERTIA_SIZE=4, |PE|=1. Also, for one 1W the effect of the temperature is {10º, -5º, -3º, -2º} and for 0.5 W {6º, -3º, -2º, -1º}. Table I presents a hypothetical example, with at the top of the Table the current temperature (T_{cur}), and the average power obtained from monitoring. Below, it is presented the current state of the *thermal_influence* matrix and the effect of the applied power on the temperature (in fact the value of the current effect corresponds to the addition of both values). The gray cells correspond to the current access position of the *thermal_influence* matrix. After accounting the effect of the power in the temperature, its influence is zeroed for the next measurement (red zeros). The last row is the new temperature considering the effect of the power on the temperature, as well as the effect of the previously sampled power values.

TABLE I. EXAMPLE OF HOW THE MEASURED AVERAGE POWER AFFECTS THE TEMPERATURE USING THE THERMAL_INFLUENCE MATRIX.

T_{cur}	45 º			55 º			60 º			58 º			56 º	
Power	1W			1W			0.5W			0.5W			1W	
thermal_influence_matrix	cur	effect		cur	effect		cur.	effect		cur	effect		cur	effect
	0	**10**		**0**	-2		-2	-2		-4	-3		**-7**	**10**
	0	-5		**-5**	**10**		**0**	-1		-1	-2		-3	-5
	0	-3		-3	-5		**-8**	**6**		**0**	-1		-1	-3
	0	-2		-2	-3		-5	-3		**-8**	**6**		**0**	-2
New T	55 º			60 º			58 º			56 º			59 º	

The first power sample adds 10 º in the PE temperature. The second power sample adds 5º in the PE temperature (-5º due to the previous thermal influence plus 10º due to the current power). The third power sample, 0.5W, would add 6º to the PE, but due the thermal influence of the previous power samples, the temperature reduces 2º. The Figure also shows the circular behavior of the *thermal_influence* matrix.

Fig. 6 presents the proposed heuristic to update the current temperature of all PEs. Besides the *temperature* and *power* vectors, the procedure receives the *idx* values, which corresponds to the access position of the *thermal_influence* matrix (gray cells of the example presented in Table I). This heuristic uses 3 functions:

- **get_matrix_type**(PE) (at lines 3 and 7) – according to the position in the system, the PE under evaluation can be diagonal (PEs 1, 3, 7, 9 in Fig. 5), lateral (PEs 2, 4, 6, 8) or central (PE 5). This function returns a pointer to one of the *thermal matrices*.

PE 7	PE 8	PE 9
PE 4	PE 5	PE 6
PE 1	PE 2	PE 3

Fig. 5. PE index to illustrates the concept of diagonal, lateral and central PEs.

- **neighbors** (PE) (line 5) – this function returns a list with the direct neighbors of a given PE. For example (Fig. 5), if the PE under evaluation is 1 the returned set is {2,4,5}, if PE=5 the returned set is {1,2,3,4,6,7,8,9}.
- **get_ng_influence_over_pe**(PE) (line 8) – returns how a given PE (ng) influence the PE under evaluation. In the previous example, if PE=1 and ng=2, the function returns *lateral.* If ng=5, the function returns *diagonal*.

```
void update_temperature(idx, temperature[], power[])
    // idx: current position in the thermal influence matrix
    // temperature[|PE|]: current temperature of all PEs, updated by this procedure
    // power[|PE|]: monitored average power of all PEs in the monitoring window
1.  FOR EACH PE pe in the system
2.    FOR inertia =1 to INERTIA_SIZE
3.        *m = get_matrix_type(pe)          // matrix type (lat/diag/cent)
4.        thermal_influence(idx, pe) += m [power(pe)] [inertia] [0]
5.        neighbors_list ← neighbors(pe)
6.        FOR EACH PE ng in the neighbors_list
7.          *m = get_matrix_type(ng)
8.          pos = get_ng_influence_over_pe(pe, ng)      // lat or diag
9.          thermal_influence (idx, pe) += m [power(ng)] [inertia] [pos]
10.       END FOR
11.     idx ← (idx +1) mod INERTIA_SIZE
12.   END FOR
13.   temperature(pe) += thermal_influence(idx, pe)
14.   thermal_influence(idx, pe) ← 0
15. END FOR
16. idx ← (idx +1) mod INERTIA_SIZE
```

Fig. 6. Heuristic to update the current temperature of all PEs.

The external loop (lines 1 to 15 in Fig. 6) updates the temperature of all PEs. The loop between lines 2 to 12 updates the *thermal_influence* (TI) matrix, as in the example of Table I, now considering the influence of the direct neighbors. Lines 3-4 update the TI matrix considering only the PE under evaluation. Next, lines 5-10 the TI matrix is updated considering the direct neighbors.

Line 11 increments the current index of the TI matrix. Note that the loop 2-12 has, in fact, two indexes: *inertia*, which accounts the effect of the temperature over time*; idx*, used to fill the TI matrix. When the loop 2-12 ends, *idx* returns to its original value. At line 13 the temperature of the PE under evaluation is updated. At line 14 the thermal influence in the current TI matrix is zeroed, as shown in the example of Table I. When all PEs have their temperature updated, the *idx* parameter is incremented (line 16).

This heuristic has a computational complexity O(*n*), where *n* is the number of PEs. In the experiments for an MPSoC running at 100 MHz and 3x3 size, the worst case execution time of the heuristic is 325 µs.

V. RESULTS

The experiments were executed in a public domain NoC-based MPSoC [18], using a clock cycle accurate model described in SystemC. Each PE executes a multi-task operating system (µkernel) and user tasks.

Five benchmarks, described in C language, are used: (*i*) DTW - Digital Time Warping (DTW), with 10 tasks; (*ii*) MPEG decoder, with 5 tasks; (*iii*) DJK - Dijkstra, with 6 tasks; (*iv*) SYN1, synthetic application, with 12 tasks, which emulates the communication behavior of an MPEG4 full decoder; (*v*) SYN2, synthetic application, with 12 tasks, that emulates the communication behavior of VOP (Video Object Plane) decoder application.

Experiments are conducted using the scenarios presented in Table II with three different MPSoC Size: 3x3; 4x4; 6x6. In all MPSoC sizes, one PE is reserved for management purposes (e.g. to execute the proposed heuristic). The experiments limit the size of the system to 36 PEs, since larger systems adopt a cluster-based organization, where the MPSoCs is partitioned in clusters, with one manager per cluster, ensuring scalability [19].

TABLE II. SCENARIOS USED TO EVALUATE THE HEURISTIC TO ESTIMATE THE TEMPERATURE AT RUNTIME.

Scenario	Applications	Number of tasks
1	20 x MPEG, 20 x DJK, 20 x SYN1, 20 x SYN2, 20 x DTW	780
2	10 x MPEG, 10 x DJK, 10 x SYN1, 10 x SYN2, 10 x DTW	390
3	50 x MPEG	250
4	100 x DTW	1000
5	100 x MPEG	500

A. Accuracy of the Proposed Model

This section compares the proposed model w.r.t. HotSpot. Table III summarizes the absolute errors considering the HotSpot as the reference. Two mapping heuristics are used in the evaluation: "*Standard*", whose cost function is the minimization of communication energy in the NoC; "*Energy-Aware*" [20], whose cost function is the workload distribution over time.

TABLE III. ERROR OF THE PROPOSED HEURISTIC W.R.T HOTSPOT.

Scenario / MPSoC size		"Standard" Mapping			Energy-Aware Mapping		
		AVG	STDEV	MAX	AVG	STDEV	MAX
3x3	1	2.33%	2.19%	8.88%	1.34%	1.33%	5.71%
	2	1.98%	2.12%	8.88%	1.14%	1.23%	5.71%
	3	1.46%	1.61%	8.00%	1.08%	1.07%	5.44%
	4	1.21%	0.81%	5.14%	1.13%	1.06%	5.50%
	5	2.42%	2.44%	9.74%	1.44%	1.57%	6.69%
4x4	1	**3.52%**	**3.31%**	11.99%	1.87%	1.84%	**6.94%**
	2	2.53%	2.53%	10.86%	1.38%	1.41%	6.18%
	3	1.92%	1.66%	7.70%	0.87%	0.80%	3.66%
	4	1.79%	1.88%	9.02%	1.14%	0.98%	4.05%
	5	3.13%	3.03%	12.14%	1.43%	1.41%	5.75%
6x6	1	3.18%	3.16%	14.20%	1.91%	1.64%	5.89%
	2	2.17%	2.22%	12.05%	1.05%	0.99%	4.43%
	3	0.95%	0.88%	5.00%	0.52%	0.45%	2.49%
	4	0.57%	0.50%	2.06%	0.68%	0.53%	2.28%
	5	1.66%	1.78%	10.07%	0.94%	0.88%	3.87%

The average absolute error (columns AVG) is below 3.6%, with and standard deviation below 3.2%. The "*Standard*" mapping creates hotspot zones, inducing a larger temperature in some PEs, revealing the effects induced by the assumptions made in the model. A mapping heuristic that distributes the workload evenly prevents the creation of hotspots, resulting in a smaller average absolute error and standard deviation compared to the "*Standard*" mapping. In most cases, the maximum absolute error (columns MAX) is below 10%. The higher errors are observed with the "*Standard*" mapping due to the hotspot zones. The maximum error, using the "*Energy-Aware*" mapping, is 6.94%.

The proposed software-based model (executed in 0.35 ms@100MHz) enables to estimate with accuracy the temperature of each PE at runtime. A temperature estimation with an error smaller than 10% w.r.t HotSpot enables to manage safely the system temperature at runtime. The temperature is available for each PE (fine-grain data), differently from approaches with thermal sensors, which provide temperature data for the entire system or larger areas.

B. Thermal Maps

Fig. 7 presents the thermal distribution for scenario 4 in a 6x6 MPSoC (similar results are observed for the others scenarios). Each square represents a processor in the MPSoC (x and y-axis), and the time axis represents different moments of the simulation (5%, 25%, 50%, 75% and 100% of execution time). As illustrated in Fig. 7(a), the proposed model produces a similar thermal distribution compared to Hotspot Model (Fig. 7(b)), which validates the proposed model.

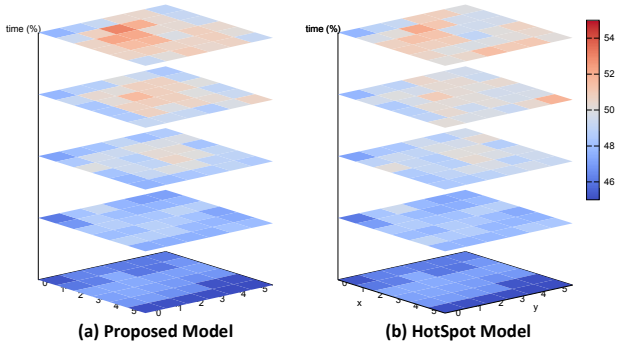

Fig. 7. 6x6 MPSoC platform temperature distribution considering (a) proposed and (b) HotSpot models.

Fig. 8 compares thermal maps, considering the two mapping heuristics.

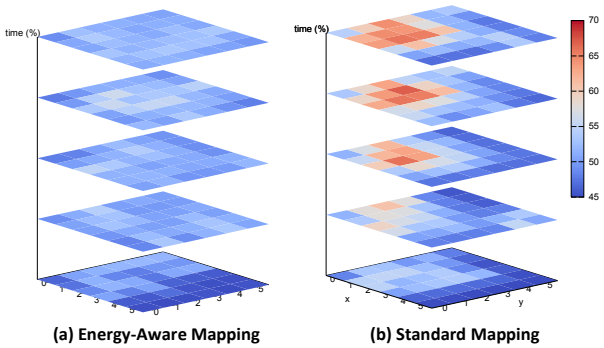

Fig. 8. (a) Energy-Aware Mapping, (b) Standard Mapping.

The goal is to present the occurrence of hotspots (Fig. 8(b)). With the presence of hotspots, the influence of hot processors increases, rising the error of the model, as presented in Table III. Even with hotspots the average error is small (<3.6%). Only the processors in the hotspot zones have the estimation error increased (worst case: 14.2%). This thermal comparison aids in the development of heuristics, like mapping, or other power management heuristics (e.g. DVFS).

C. Computational Cost Estimation

This Section evaluates two overheads induced by the inclusion of monitoring and temperature estimation in the system. The graph presented in Fig. 9 evaluates the execution time and energy overheads. All scenarios were executed with and without the proposed method. The total execution time overhead of the proposed model is less than 4.9% in the worst case. As the execution time increases, the energy consumed by MPSoC also increases. Results show that the energy overhead of our model varies from 0.87% to 3.64% depending on the scenario. These results demonstrate the low cost of proposed model.

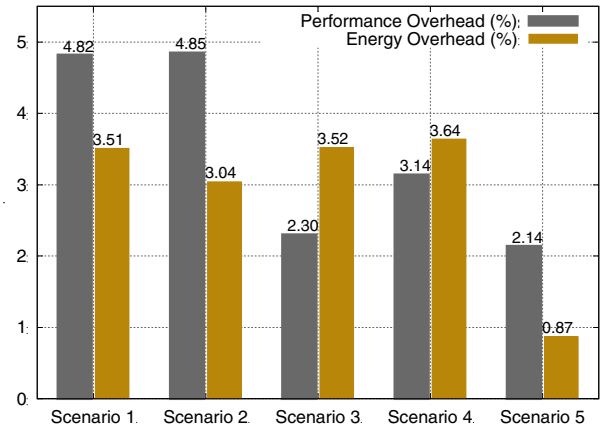

Fig. 9. Overhead of the proposed model (6x6 MPSoC).

VI. Conclusions and Future Works

This paper proposed a lightweight temperature model for monitoring the system temperature at runtime. The features included in proposed temperature model include scalability, small computation cost, and runtime execution. HotSpot tool is used to generate the system thermal behavior (*design-time phase*), which is used to calculate the system temperature at runtime (*runtime phase*).

The proposed model achieved a low error, with minimal impact on energy consumption (<4%) and runtime performance overhead (<5%). Results show that the average absolute error of the temperature estimation, compared to HotSpot is smaller than 4% in systems with up to 36 processing elements.

Future works include to: (1) integrate of a lifetime model to evaluate MTTF (Mean Time to Failure); (2) include the proposed temperature model to guide mapping heuristics.

VII. Acknowledgments

The Author Fernando Moraes is supported by CNPq - projects 472126/2013-0 and 302625/2012-7, and FAPERGS - project 2242- 2551/14-8.

REFERENCES

[1] Henkel, J.; Bauer, L.; Dutt, N.; Gupta, P.; Nassif, S.; Shafique, M.; Tahoori, M.; Wehn, N. "*Reliable On-chip Systems in the Nano-era: Lessons Learnt and Future Trends*". In: DAC, 2013, 10p.

[2] Semenov, O.; Vassighi, A.; Sachdev, M. "*Impact of self-heating effect on long-term reliability and performance degradation in CMOS circuits*". IEEE Trans. on Device and Materials Reliability, v.6(1), 2006, pp. 17–27.

[3] Hruska, J. "*NVIDIA denies rumors of faulty chips, mass GPU failures*". http://arstechnica.com/gadgets/2008/07/nvidia-denies-rumors-of-mass-gpu-failures/.

[4] Chantem, T.; Xiang, Y.; Hu, X. S.; Dick, R. P. "*Enhancing multicore reliability through wear compensation in online assignment and scheduling*". In: DATE, 2013, pp. 1373–1378.

[5] Rudi, A.; Bartolini, A.; Lodi, A.; Benini, L. "*Optimum: Thermal-aware task allocation for heterogeneous many-core devices*". In: HPCS, 2014, pp. 82–87.

[6] Wanner, L.; Apte, C.; Balani, R.; Gupta, P.; Srivastava, M. "*Hardware Variability-Aware Duty Cycling for Embedded Sensors*". IEEE Trans. on Very Large Scale Integration Systems, v.21(6), 2013, pp. 1000–1012.

[7] Kong, J.; Chung, S. W.; Skadron, K. "*Recent Thermal Management Techniques for Microprocessors*". ACM Computing Surveys, v.44(3), 2012, 42 p.

[8] Ware, M.; Rajamani, K.; Floyd, M.; Brock, B.; Rubio, J. C.; Rawson, F.; Carter, J. B. "*Architecting for power management: The IBM POWER7 approach*". In: HPCA, 2010, 11 p.

[9] Huang, W.; Ghosh, S.; Velusamy, S.; Sankaranarayanan, K.; Skadron, K.; Stan, M. R. "*HotSpot: a compact thermal modeling methodology for early-stage VLSI design*". IEEE Trans. on Very Large Scale Integration Systems, v.14(5) , 2006, pp. 501–513.

[10] Ge, Y.; Malani, P.; Qiu, Q. "*Distributed task migration for thermal management in many-core systems*". In: DAC, 2010, pp. 579–584.

[11] Liu, Z.; Tan, S. X. D.; Huang, X.; Wang, H. "*Task Migrations for Distributed Thermal Management Considering Transient Effects*". IEEE Trans. on Very Large Scale Integration Systems, v.23(2), 2015, pp. 397–401.

[12] Kursun, E.; Cher, C.-Y. "*Variation-aware thermal characterization and management of multi-core architectures*". In: ICCD, 2008, pp. 280–285.

[13] Wu, Y. K.; Sharifi, S.; Rosing, T. S. "*Distributed thermal management for embedded heterogeneous MPSoCs with dedicated hardware accelerators*". In: ICCD, 2011, pp. 183–189.

[14] Lee, K. J.; Skadron, K. "*Using performance counters for runtime temperature sensing in high-performance processors*". In: IPDPS, 2005, pp. 8 -13.

[15] Jejurikar, R.; Pereira, C.; Gupta, R. "*Leakage aware dynamic voltage scaling for real-time embedded systems*". In: DAC, 2004, pp. 275-280.

[16] Rosa, F.; Ost, L.; Raupp, T.; Moraes, F; Reis, R. "*Fast energy evaluation of embedded applications for many-core systems*". In: PATMOS, 2014, 6 p.

[17] Martins, A.; Silva, D.; Castilhos, G.; Monteiro, T.; Moraes, F. "*A method for NoC-based MPSoC energy consumption estimation*". In: ICECS, 2014, pp. 427-430.

[18] Carara, E.; Oliveira, R.; Calazans, N.; Moraes, F. "*HeMPS - a Framework for NoC-based MPSoC Generation*". In: ISCAS, 2009, pp. 1345-1348.

[19] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. "*Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes*". In: ISVLSI, 2013, pp. 153-158.

[20] Castilhos, G.; Mandelli, M.; Ost, L.; Moraes, F. "Hierarchical Energy Monitoring for Task Mapping in Many-core Systems". Journal of Systems Architecture, v.63, 2016, pp. 80–92.