A Distributed Energy-aware Task Mapping to Achieve Thermal Balancing and Improve Reliability of Many-core Systems

Marcelo Mandelli[1,2], Guilherme Castilhos[1], Gilles Sassatelli[2], Luciano Ost[3], Fernando G. Moraes[1]

[1]FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
[2] LIRMM – 161 rue Ada, Cedex 05, 34095 Montpellier, France
[3] University of Leicester - University Rd, Leicester LE1 7RH, United Kingdom
mandelli@ieee.org, guilherme.castilhos@acad.pucrs.br, sassatelli@lirmm.fr, luciano.ost@leicester.ac.uk, fernando.moraes@pucrs.br

ABSTRACT

Investigating novel techniques to improve many-core embedded systems lifetime, reliability, and thermal management is a fundamental challenge for the semiconductor industry. Imbalanced mapping of applications may considerably affect the system performance and lifetime due to thermal issues in an integrated circuit (e.g. hotspot zones). Traditional mapping techniques focus on local optimizations, e.g. minimize the number of hops between communicating tasks, which may lead to hotspot zones and underutilization of some processing resources. This paper proposes a runtime mapping heuristic whose cost function targets temporal workload and energy consumption balance in large scale systems. The proposed heuristic minimizes the occurrence of hotspots by distributing application workload onto the processing elements in a uniform way, which contributes to a balanced thermal distribution across the system. These features improve system reliability and postpone aging effects. Results with several benchmarks executing in a cycle-accurate platform model show a uniform system utilization when comparing the proposed heuristic to conventional mapping approaches.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles – advanced technologies, VLSI (very large scale integration).

General Terms

Design, Experimentation, Performance, Reliability, Verification.

Keywords

Task mapping; energy-aware mapping; load balance; energy consumption; many-core.

SBCCI '15, August 31-September 04, 2015, Salvador, Brazil

DOI: http://dx.doi.org/10.1145/2800986.2800992.

1 INTRODUCTION

In networked many-core systems, task mapping has a significant impact on both system performance and lifetime [1][2]. Task mapping leads directly to system workload distribution. Load imbalance judgments can generate hotspots zones (i.e. peaks of power dissipation) and consequent thermal implications, which may result in unreliable system operation. For instance, mapping communicating tasks far from each other may result in more data transfer, and consequently induce faults in overused links. While fault links may produce unreached zones (e.g. isolated cores), overutilization of cores makes them more susceptible to failures due to high-temperature variation and operation.

Several mapping techniques have been investigated over the last years. Such techniques rely on distinct principals (e.g. bio-inspired) and target different optimization goals (e.g. inter-tasks communication latency, network contention). Task mapping solutions aiming to improve system reliability must provide both spatial and temporal load balancing [1], considering internal node components switching-activity (e.g. processor, memory) and power-state (i.e. active, sleep mode) resulting from allocated workloads (i.e. application tasks) [3].

The *goal* of the present work is to propose a scalable and lightweight runtime distributed energy-aware mapping technique, which induces to a balanced thermal distribution, without penalizing applications' execution time in large-scale systems. Proposed mapping also uses the total energy consumption of cores to slow down their wear process. This mapping solution comprises both offline and online phases. The offline phase uses a *profiler platform* to generate performance information (inter-task communication volume, energy consumption of each task) to guide the runtime mapping process. The offline data is used at runtime to generate mapping solutions that equally distribute the applications' load among platform resources.

The main *contributions* of this work include:

- proposal of a distributed runtime mapping technique that reduces hotspot zones, with a uniform workload distribution;
- validation of the proposed mapping technique in a cycle-accurate SystemC NoC-based MPSoC model with large scenarios, considering the number of executed instructions, power, energy, temperature and application execution time;
- comparison of the proposed method to a conventional mapping heuristic, where the cost function aims to reduce communication energy.

This work is organized as follows. Section 2 presents related works. Sections 3 and 4 present both the MPSoC and energy models. Section 5 describes the proposed mapping technique. Section 6 presents and discusses the results. Section 7 concludes the paper and presents directions for the future works.

2 RELATED WORK

The literature presents different task mapping approaches to improve system reliability. Most reviewed works use a centralized approach [1][4][5][6][7][8][9][10][11][12]. Among them, some works propose temperature or reliability models [4][5][6][7][8][9], which cannot be incorporated into runtime mapping decision due to their computational complexity. For example, Coskun et al. [4][5] use temperature output data from the Hotspot thermal model [13] to take mapping decisions. Other works [6][7][8][9] use high computational models to produce mapping decisions at design time, which are stored in a database and used at runtime. This approach may reduce system performance due its incapability of dealing with unpredictable system variations. In turn, Chantem et al. [1] propose a runtime task mapping that incorporates a full system reliability analysis model, which may improve system MTTF. However, the underlying approach does not consider system scalability and performance. Task mapping approaches proposed in [1][10][11], employ physical sensors to capture thermal or wear-state condition of cores at runtime. Included sensors provide accurate information to the mapping decision at the cost of the additional system area and energy consumption. Cox et al. [12] present a design-time task mapping approach, considering throughput, communication, and storage constraints. As mentioned before, design-time approaches cannot handle dynamic workloads.

Other works use distributed approaches to improve system reliability, including task migration and DVFS [14][15][16]. Ge et al. [14] propose a task migration approach for thermal system balancing. This approach uses thermal sensors, which aggregate hardware costs. Liu et al. [16] also present a thermal management task migration approach, which does not consider performance costs. Wu et al. [15] present a dynamic frequency scaling for thermal management, which may impose additional hardware costs. These approaches rely on analytical models only, so neither real platform implementations nor real applications are considered.

Differently from reviewed works, the proposed approach proposes a scalable and lightweight runtime task mapping technique, which provides thermal balance distribution without extra hardware cost. Proposed work is validated under large scenarios (MPSoCs with up to 144 PEs), considering a cycle-accurate SystemC NoC-based MPSoC platform.

3 MPSOC MODEL

The present work adopts a NoC-based MPSoC, with distributed memory architecture. In this platform, each PE adopts a scratchpad as local storage memory due its power efficiency and management facilities when compared to cache memories [17]. Communication between tasks occurs only by message passing. The MPSoC model is a directed graph GMPSoC = (PE, L), where each vertex $pe_i \in$ PE is a processing element with a processor, a local memory, a DMA module and a router. An edge $l_{ij} \in$ L is a NoC link interconnecting pe_i to pe_j. The MPSoC is hierarchically managed, partitioned in clusters as illustrated in Figure 1(a) [18].

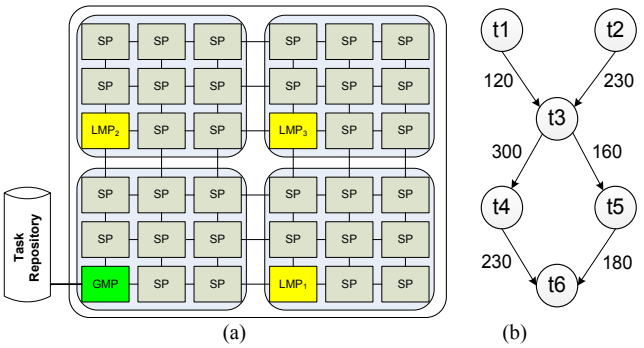


Figure 1 – (a) 6x6 MPSoC instance, with four 3x3 clusters, (b) application modeled as a task graph GApp = (T, E).

The set PE is divided into three subsets: LMP, GMP and SP, where (LMP ∪ GMP ∪ SP) = PE. Each $pe_i \in$ PE may act as:

- ***LMP*** – local manager processors, control the cluster, executing functions such as monitoring, task mapping, task migration, verification of deadlines, reclustering.
- ***GMP*** – global manager processor, execute all functions of the LMP, and functions related to the overall system management, such as defining application-to-cluster mapping, controlling external devices accesses (e.g. task repository memory).
- ***SP*** – slave processors, responsible for executing user's applications. Each SP runs a simple operating system, which enables the communication between PEs and multitask execution. The local memory is divided into MAX_SP_TASKS pages, also called *resources*. A task executing in the system is mapped to one resource. Therefore, each SP can execute MAX_SP_TASKS tasks simultaneously. If a resource does not have a task mapped on it, it is considered *free* or *available*.

The definition of the cluster size occurs at design time. At runtime, a reclustering process enables the modification of the cluster size. When an application is requested to be mapped to a given cluster, its LMP checks the availability of SPs, and if necessary, SPs are borrowed from neighbor clusters. When the application finishes its execution, the borrowed SPs are released to the original cluster.

4 ENERGY MODEL

The energy consumption in the MPSoC is mainly due to three components: memory, processors, and NoC. The number of memory accesses is identical for the same workload. Therefore, to fairly compare different mapping solutions using the same workload, we consider the processor and NoC energy consumption as main metrics.

As described in the literature [19], the energy consumption of a processor pe_i is defined by static and dynamic consumption. The processor energy consumption 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, as proposed in [20].

Each processor pe_i contains an *instruction analyzer* module, which includes the energy cost of each instruction. Such module counts and classifies the executed instructions for different classes at runtime. The set of classes is defined as $C = \{c_0, c_1, \ldots, c_8\}$, containing 9 different classes (e.g. arithmetic, logic, branch) [20]. Equation (1) presents the energy dissipation for a given task.

$$energy_{task} = \sum_{i=0}^{8} (\boldsymbol{energy}(c_i) \times \boldsymbol{total\_instructions}(c_i)) \qquad (1)$$

where: $energy(c_i)$, energy to execute a given instruction belonging to the class c_i, value obtained from the physical synthesis; $total\_instructions(c_i)$, executed instructions belonging to the class c_i.

Results show that the error of adopted instruction analyzer module varies from 0.06% to 8.05% when compared to a gate-level implementation [20]. One can argue that this profile step may be inaccurate since the task workload may vary, according to the user data. During the profiling step, each application receives workloads representing real execution scenarios. Therefore, the computed energy values are representative enough to guide the proposed mapping heuristic [21][22].

It is necessary to compute the power per *time slice* to evaluate the mapping heuristics Using the *instruction analyzer* module employed during the profile step, now at runtime, Equation (2) computes the power consumption for each pe_i.

$$total\_power_{pe} = \frac{\sum_{i=0}^{8}(energy(c_i) \times total\_instructions(c_i))}{time\_slice} \quad (2)$$

where: *time_slice* is the power sampling period, in seconds.

The energy consumption in the NoC is proportional to the number of transmitted flits at each router port. A gate level description of the NoC is used to determine the power consumption for the main router components: buffers, internal crossbar and control logic. Equation 3 presents energy consumption corresponding to one flit being transmitted through 1 buffer of the router.

$$E_{router} = \begin{bmatrix} (n\_ports - 1) * P_{buffer}(0) + P_{buffer}(1) + \\ P_{crossbar}(1) + P_{control\_logic}(1) \end{bmatrix} * T \quad (3)$$

where: n_port is the number of ports of the router, $P_{component}(0)$ the average power without traffic, $P_{component}(1)$ the average power with an injection rate equal to 100%, T the clock period.

Equation 4 presents the power consumption of a given router for a given number of simulated cycles (*time_slice*), considering the number of flits transmitted by the router in the sampling period.

$$total\_power_{router} = \frac{E_{router} * \sum flits}{time\_slice} \quad (4)$$

5 MAPPING HEURISTIC

An application is modeled as a directed graph GApp = (T, E), where each vertex $t_i \in T$ represents an application task and each directed weighted edge $e_{ij} \in E$ represents a communication dependence between tasks t_i and t_j. The weight of an edge e_{ij} is denoted by $comm_{ij}$, representing the total data communication volume transferred between application tasks t_i and t_j.

Figure 1(b) presents an example of an application modeled as a task graph. An application has *initial tasks* (e.g. t_1 and t_2), those without dependences to other tasks, and *non-initial tasks*. Each task $t_i \in T$ contains:

- a set C_i called communication task list. This set is defined as $C_i = \{(t_j, comm_{ij}); (t_k, comm_{ik}); \ldots (t_n, comm_{in})\}$, where each element is a tuple containing a task t_j that communicates with t_i and the volume $comm_{ij}$ transferred between t_i and this task.
- an energy value E_i related to the execution of this task on the target PE.

The set C_i and E_i are obtained from profiling. For this purpose, a slightly modified version of the MPSoC model presented in Section 3, called *profiler platform*, with monitors to capture for each task the consumed energy [20], the execution time, and the communication volume between tasks. A network packet monitor is used to capture data communication among tasks. For each transmitted packet, the monitors capture the source task identifier, the destination task identifier, and the packet size. With such information, it is possible to define the set $C(t_i)$ of each task. Each application is executed in the *profiler platform*, without any disturbing traffic.

The mapping of the set T = $\{t_1, t_2, ..., t_n\}$ of GApp onto the set SP = $\{sp_1, sp_2, ..., sp_k\}$ of GMPSoC is defined by the mapping function: T → SP, where $\forall t_i \in T, \exists\ sp_j \in SP$.

Each SP computes the total consumed energy (*TE*), corresponding to the energy consumed by all already executed tasks and the tasks that are currently being executed on this processor. Whenever a task is mapped onto a SP, the *TE* value is updated. The TE is the main parameter to guide the mapping function presented in the next subsections. Further, the proposed heuristics use the following definitions:

Definition 1: *application size* (APP.size) corresponds to the number of tasks of the application to be mapped.

Definition 2: *available resources* corresponds to the number of resources that does not have a task mapped on it. This information may refers to the whole system, *available_resources*(*system*); or to a given cluster c_i, *available_resources*(c_i).

Definition 3: The function *available*(sp_i) returns *true* if sp_i is available to receive a new task, otherwise *false*. A SP is available when the number of tasks mapped onto it is smaller than MAX_SP_TASKS.

Definition 4: cluster energy, cl_energy(c_k). Obtained from (5), where p corresponds to the SP addresses' in the cluster. Note that the accumulated energy in the cluster guides the cluster selection.

$$\mathbf{cl\_energy}(c_k) = \sum_{p=address(0)}^{address(n)} TE(p) \quad (5)$$

Definition 5: *region_energy* (sp_i, n_hops), returns the average *TE* from the set with sp_i and all SPs up to n_hops from sp_i. Figure 2 shows a hypothetical example, with the numbers inside each rectangle representing the *TE* of each SP.

			123			
		66	178	280		
	114	200	80	109	77	
120	210	120	200	110	350	327
	124	156	85	413	95	
		149	123	189		
			102			

The number of SPs inside the region is 25:
$region\_energy(sp_{3,3}, 3)$=4100/25=164.

Figure 2 – Hypothetical example of *region_energy* in a 7x7 cluster, being sp_i the central SP, and n_*hops*=3.

The proposed mapping function is divided into three steps: *cluster selection* (Section 5.1), *initial tasks mapping* (Section 5.2), and *non-initial tasks mapping* (Section 5.3).

5.1 Cluster Selection

When a new application is requested to be mapped, the GMP selects a cluster to map this application. Algorithm 1 presents the pseudo-code of the cluster selection heuristic. The heuristic first verifies if the system has available resources to map the application (line 4). If there are no sufficient resources in the system, the application is scheduled to be mapped later. The first loop (lines 5-10) analyzes all clusters that have available resources to map the application, selecting the one with the smallest accumulated energy. If there are no clusters with available resources to map the application, a cluster with the smallest accumulated energy is selected, regardless the number of available resources (lines 12 – 18). *Note that the application is mapped in the MPSoC iff the system has available resources for the application.* Once the cluster is selected, the GMP transfers the application description to the LMP of this cluster.

```
Input: application size APP.size
Output: selected_cluster
1.  selected_cluster ← -1
2.  selected_cluster_energy ← +∞
3.  //Verify if the system has available resources to map the application
4.  IF available_resources(system) >= APP.size THEN
5.      FOR EACH cluster c_k in the system
6.          IF available_resources(c_k) >= APP.size AND
                        cl_energy (c_k)< selected_cluster_energy THEN
7.              selected_cluster ← c_k
8.              selected_cluster_energy ← cl_energy (c_k)
9.          END IF
10.     END FOR EACH
11.   // There is no cluster with enough resources to receive the application
12.   IF selected_cluster = -1 THEN
13.       FOR EACH cluster c_k in the system
14.           IF cl_energy (c_k)< selected_cluster_energy THEN
15.               selected_cluster ← c_k
16.               selected_cluster_energy ← cl_energy (c_k)
17.           END IF
18.       END FOR EACH
19.   END IF
20. END IF
21. return selected_cluster
```

Algorithm 1 - Cluster selection heuristic, executed in the GMP.

The reasoning of this first heuristic is to distribute the *energy* homogeneously when a new application arrives in the system. In the long-term, this procedure avoids hotspots, and processors stressed over the time. Consequently, this first heuristic contributes to minimizing aging effects, as wearout.

5.2 Mapping of Initial Tasks

The initial tasks have to be mapped to start the application execution. As the remaining tasks of the application will be mapped from the position of the initial tasks, the mapping of initial tasks affects the overall application mapping. In the presented approach, the LMP of the selected cluster receives the application description (GApp) from the GMP. Then, for each initial task a mapping heuristic is used to choose an SP within the selected cluster.

The initial tasks mapping heuristic selects the SP with the smallest *region_energy*, as detailed in Algorithm 2. The main loop (lines 3-8) evaluates if sp_i is available for receiving the initial task ($available(sp_i)$) and the consumed energy around it. This procedure executes fast, despite being exhaustive, because the number of SPs inside a cluster is small (between 16 to 25), and $n\_hops = \sqrt{|SP_{cluster}|}/2$. This procedure ensures that application's tasks that will be mapped later will be assigned closer to the selected SP and in SPs with a lower accumulated energy.

```
Input: n_hops
Output: selected_sp
1. selected_sp ← -1
2. selected_region_energy ← +∞
3. FOR EACH SP sp_i in the cluster
4.     IF available(sp_i) AND region_energy(sp_i, n_hops)< selected_region_energy THEN
5.         selected_sp ← sp_i
6.         selected_region_energy ← region_energy(sp_i, n_hops)
7.     END IF
8. END FOR EACH
9. return selected_sp
```

Algorithm 2 - Selection of the SP with the smallest region energy inside a cluster, executed in a LMP.

If the application has only one initial task, the SP chosen by the heuristic of Algorithm 2 is selected to execute the task. Otherwise, the heuristic presented in Algorithm 3 is executed for each initial task not mapped. At line 4 in Algorithm 3 it is created a set with all SPs up to *n_hops* from the selected SP computed by Algorithm 2 ($SP_{address}$). The loop between lines 6-11 selects an available SP from the *neighbors_list* with the smallest *TE*. If there is no available SP inside the list, the search space increases 1 hop (lines 12-15), until visiting all SPs of the cluster (line 5).

It is relevant to note that most mapping heuristics [23][24] try to minimize the number of hops between tasks to reduce the communication energy. The procedure of Algorithm 3 implements a ***tradeoff*** between the number of hops and accumulated energy in the processing elements.

```
Input: SP_address, n_hops
Output: selected_sp
1. selected_sp ← -1
2. selected_sp_energy ← +∞
3. // Get all neighbors of selected_sp within a distance n_hops
4. neighbors_list ← neighbors(SP_address, n_hops)
5. WHILE all SPs in the cluster not evaluated AND selected_sp=-1 DO
6.     FOR EACH SP sp_i IN neighbors_list
7.         IF available(sp_i) = true AND TE(sp_i)< selected_sp_energy THEN
8.             selected_sp ← sp_i
9.             selected_sp_energy ← TE (sp_i)
10.        END IF
11.    END FOR
12.    IF selected_sp = -1 THEN
13.        n_hops ← n_hops +1
14.        neighbors_list ← neighbors(SP_address, n_hops)
15.    END IF
16. END WHILE
17. return selected_sp
```

Algorithm 3 - Initial tasks mapping, executed in a LMP.

5.3 Mapping of non-initial Tasks

Following the example, after mapping t_1 and t_2 (initial tasks, Figure 1(b)), t_3 must be mapped at the first attempt of t_1 or t_2 sending a message to it. When the *kernel* running in an SP does not find a task address, it sends a message to the LMP requesting the mapping of a new task. Algorithm 4 presents the heuristic used to map non-initial tasks. The heuristic creates a list with all tasks communicating with t_i already mapped onto the processing elements within the cluster (line 3). In the sequel, it is defined a bounding box rectangle (line 4), with all mapped communicating tasks. This bounding box is increased by one hop (line 5), offering a larger search space to map t_i. Figure 3 illustrates the mapping search space when one (a) or more communicating tasks (b) are mapped in the cluster. The adoption of a bounding box aims to reduce the distance among communicating tasks. A list with candidate SPs is created (line 7). The available SP in the list with the smallest *TE* is selected (lines 8-13). If no SP can be selected, the bounding box is increased by one hop (lines 14-16). This process continues up to find a SP or visiting all SPs of the cluster.

```
Input: t_i , set C(t_i)
Output: selected_sp
1.  selected_sp ← -1
2.  selected_sp_energy ← +∞
3.  MC(t_i)← mapped_tasks(C(t_i))   // all tasks communicating with t_i already mapped
4.  bounding_box ← area(MC(t_i))
5.  increase(bounding_box, 1)
6.  WHILE all SPs in the cluster were not evaluated AND selected_sp=-1 DO
7.      neighbors_list ← search_SPs(bounding_box)
8.      FOR EACH SP sp_i IN neighbors_list
9.          IF available(sp_i) = true AND TE(sp_i)< selected_sp_energy THEN
10.             selected_sp ← sp_i
11.             selected_sp_energy ← TE(sp_i)
12.         END IF
13.     END FOR
14.     IF selected_sp = -1 THEN
15.         increase(bounding_box, 1)
16.     END IF
17. END WHILE
18. return selected_sp
```

Algorithm 4 - Mapping of non-initial tasks, executed in a LMP.

The mapping of the initial and non-initial tasks may fail if there are no available resources in the cluster (*select_sp=-1*). In this case, the reclustering process is executed. The reclustering process asks a resource to the neighbor LMPs, selecting the one closest to the mapped tasks.

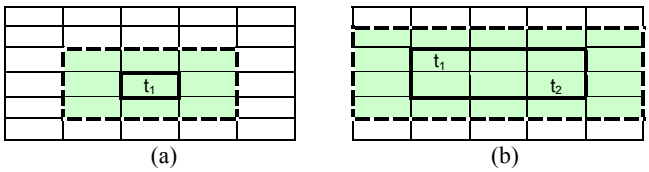


Figure 3 – (a) search space when one communicating tasks is already mapped (t_1); (b) search space when more than one communicating task is already mapped (t_1 and t_2). Thicker lines correspond to the original bounding box, dashed lines to the bounding box increased by one hop.

6 EXPERIMENTAL SETUP AND RESULTS

Experimental results use a SystemC cycle-accurate NoC-based MPSoC platform [25]. Each PE executes a small multi-task μkernel, and five applications described in C language:

- DTW - Digital Time Warping (DTW), with 10 tasks;
- MPEG decoder, with 5 tasks;
- DJK - Dijkstra, with 6 tasks;
- SYN1, synthetic application, with 12 tasks, which emulates the communication behavior of an MPEG4 full decoder;
- SYN2, synthetic application, with 12 tasks, that emulates the communication behavior of VOP (Video Object Plane) decoder application.

Table 1 presents the five scenarios adopted to evaluate the proposed mapping heuristic, varying the MPSoC size, the cluster size and the number of applications.

Table 1 - Evaluated scenarios configuration.

Scenario	MPSoC size	Cluster size	Applications	Number of tasks
1	6x6	3x3	8 x DTW, 8 x MPEG, 8 x DJK, 8 x SYN1, 8 x SYN2	360
2			8 x DJK, 11 x SYN1, 11 x SYN2	312
3			16 x MPEG, 8 x DJK, 8 x SYN2	224
4			3 x DTW, 6 x MPEG, 9 x DJK, 5 x SYN1, 8 x SYN2	270
5	12x12	4x4	10 x DTW, 25 x MPEG, 18 x DJK, 14 x SYN1, 12 x SYN2	645

Each scenario is executed using the proposed energy-aware mapping heuristic and a communication-aware mapping heuristic proposed in [26], called *LEC-DN*. Such heuristic reduces the communication energy in the NoC by reducing the distance in hops between communicating tasks. When a given task t_i is required to be mapped, this heuristic first analyses the set of communicating tasks with t_i already mapped. Then, the heuristic approximates t_i to the tasks it has a higher communication volume. The LEC-DN heuristic does not consider the tasks' load in the mapping function and may induce the mapping of tasks with a high load in the same SP to reduce the communication energy.

The use of an MPSoC is dynamic, i.e., applications start at different moments, characterizing a dynamic workload behavior. A simulated dynamic workload is used in the experimental scenarios to stress the mapping heuristic, with situations with a large number of available resources and few available resources. For this purpose, two thresholds control the workload behavior: minimum (MIN_USAGE) and maximum (MAX_USAGE) system usage. The simulation starts with a usage equal to 0%. Applications are then inserted into the system until MAX_USAGE is reached. When the load attains MAX_USAGE, the insertion of new applications is blocked. The load of the system reduces when applications finish their execution. When MIN_USAGE is reached, new applications are inserted into the system again. The experiments assume MIN_USAGE=20% and MAX_USAGE=70%.

Proposed and LEC-DN heuristics are compared in terms of execution time, energy, power and temperature distribution. In addition, the mapping heuristics cost is evaluated, demonstrating the impact of the mapping process in terms of executed instructions. The goal of Scenario 5 is to show the *scalability* of proposed mapping in large systems.

6.1 Instructions and Energy Distribution

Table 2 presents the number of executed instructions and the consumed energy values (total, average and standard deviation), for the LEC-DN and proposed mapping heuristics.

Table 2 – Instructions (thousands of instructions) and energy (mJ) for the evaluated scenarios.

	LEC-DN – Instructions			Energy-aware mapping - Instructions		
	Total	Average	Std. Dev.	Total	Average	Std. Dev.
Sc1	96,668	3,021	**2,342**	98,028	3,063	**813**
Sc2	72,983	2,278	**2,876**	75,926	2,373	**851**
Sc3	49,322	1,541	**1,641**	49,581	1,549	**789**
Sc4	75,901	2,372	**2,242**	75,936	2,373	**829**
Sc5	175,692	1,220	**1,705**	183,347	1,273	**613**
	LEC-DN – Energy (mJ)			**Energy-aware mapping – Energy (mJ)**		
Sc1	2,858	89	**70**	2,908	91	**24**
Sc2	2,175	68	**86**	2,274	71	**25**
Sc3	1,447	45	**48**	1,456	46	**24**
Sc4	2,242	70	**66**	2,251	70	**25**
Sc5	5,101	35	**48**	5,434	38	**18**

The proposed energy-aware mapping heuristic executes more instructions than the LEC-DN (+ 2.6%), resulting in higher energy consumption (+3.62%). This difference is not significant, and it results from the reduced CPU sharing in the proposed mapping heuristic (energy-aware), which leads to more processing elements executing the µkernel. On the other hand, the proposed heuristic reduces the application execution time, as discussed next.

The gray columns highlight the *most relevant results* of the Table 2. The standard deviation using the proposed heuristic varies from 0.3 to 0.5 compared to the LEC-DN heuristic. This result points out to a uniform load and energy distribution along the time.

To understand the standard deviation reduction values of Table 2, Figure 4 presents the energy consumed per SP for scenario 2. Note the LEC-DN heuristic produces an unbalanced load distribution, with several underused processors (15 consuming less than 10 mJ - green rectangles), and 8 SPs consuming more than 100 mJ (red rectangles). Otherwise, the proposed heuristic provides a better load distribution, with all SPs executing applications' tasks, and only 5 SPs consuming more than 100 mJ. The highest energy value is 132.75 mJ, while the LEC-DN heuristic has 8 SP consuming more than 180 mJ. The LEC-DN heuristic concentrates the load in few SPs, increasing the probability to failures due to wear process. The proposed heuristic uses the total energy consumed in a SP (TE) value to avoid overused cores and reduce the probability of failures due to the wear process.

(a)

198.39	68.35	0.02	96.05	54.44	0.02
6.18	286.04	128.70	19.05	231.48	65.67
LMP	0.02	0.02	**LMP**	0.02	0.02
25.84	44.93	5.66	256.11	118.91	0.02
5.01	210.83	86.40	3.00	180.97	69.05
GMP	3.25	0.02	**LMP**	8.07	2.18

(b)

60.64	66.51	47.25	42.32	53.70	62.23
104.40	64.06	33.99	132.75	103.15	114.57
LMP	44.05	63.85	**LMP**	92.50	46.99
57.29	64.31	99.45	57.90	95.77	95.06
62.55	74.79	114.21	44.87	49.79	54.27
GMP	62.87	62.88	**LMP**	96.45	48.39

Figure 4 – Average energy spent per SPs (mJ) - (a) LEC-DN, (b) proposed mapping. Each rectangle represents a PE.

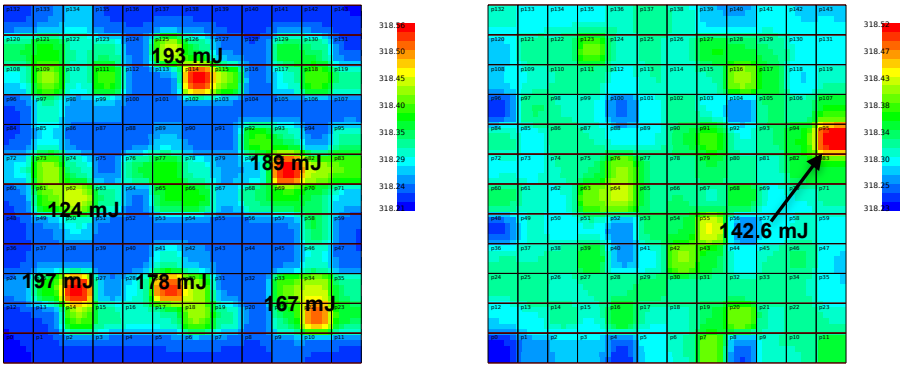
Table 3 presents the execution time for the 5 simulated scenarios. Note that the execution time reduced up to 14.7% (SC3, SC4) when the proposed heuristic is employed. This reduction comes from the better load distribution and the smaller CPU sharing among tasks. The LEC-DN heuristic tries to group the applications in clusters to reduce the communication energy, increasing the number of tasks per SP. On the other side, the proposed heuristic spreads the tasks at different SPs to obtain a uniform load distribution. A smaller execution time contributes to reducing the power due to the leakage current.

Table 3 – Execution time of the simulated scenarios (in ms).

	LEC-DN	Energy-aware mapping	Gain
SC1	**170**	**157**	**7.6 %**
SC2	**167**	**155**	**7.2 %**
SC3	**97**	**83**	**14.4 %**
SC4	**163**	**139**	**14.7 %**
SC5	**109**	**97**	**11.0 %**

A power report is generated using Equation 2. The HotSpot simulator [13] computes the temperature of the MPSoC at the end of each simulation using the power report (65 nm, 1 GHz, 1.2V, ambient temperature 318.15 K). Figure 5 presents the temperature of the MPSoC for scenario 5. The small temperature gradient (2 K) comes from the simple processor architecture (Plasma) used in our platform, with a peak consumption of 25 mW. Nonetheless, the proposed heuristic generates a uniform temperature distribution, because most PEs have a warm temperature (represented as green rectangles), showing the effectiveness of the proposal.

Scenario 5 (645 tasks) shows that proposed mapping performs well when used in a large MPSoC dimension (12x12, cluster size equal to 4x4), showing its scalability behavior. Observe in Table 2 the reduction in the standard deviation values (instructions – 64% and energy – 62.5%). While LEC-DN heuristic has 18 SPs consuming more than 100 mJ, only 1 SP achieves this energy consumption with the proposed mapping.



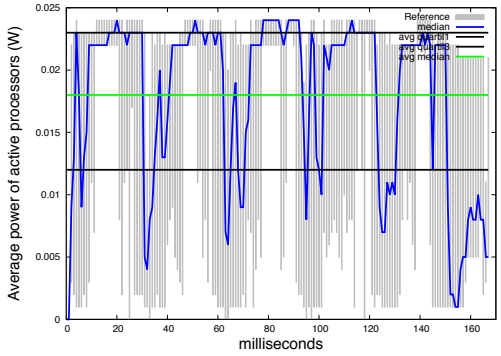
Scenario 5: (a) LEC-DN **Scenario 5: (b) Energy-aware mapping**

Figure 5 - Temperature of each PE for scenario 5, in an MPSoC with 144 PEs, being 16 PEs reserved for management (1 GMP and 15 LMPs).

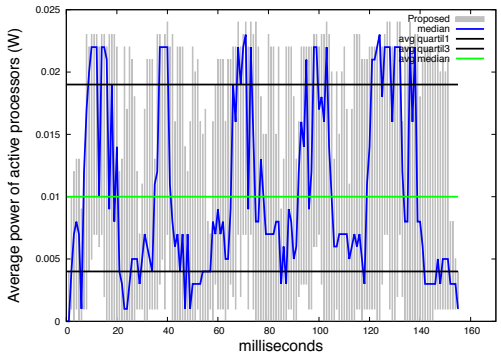
6.2 Power Distribution

Figure 6 details the power traces along the time. Note that although different, the power traces of Figure 6 correspond to the execution of the same workload but varying the task mapping. These graphs show:

- *Median*: the LEC-DN heuristic has a higher median (blue lines) than the proposal mapping, corresponding to a higher power dissipation of SPs at most sampled periods.
- *Quartiles* (black lines): the average first and third quartiles are much higher in the LEC-DN heuristic, also corresponding to a higher power dissipation of SPs.



(a) LEC-DN



(b) Energy-aware mapping

Figure 6 - Instantaneous PEs power dissipation for scenario 2. X-axis: time in milliseconds (only PEs executing tasks are considered). Y-axis: average power of active processors (W). Gray bars: 50% of the population, first to third quartiles. Black lines: average first and third quartiles. Green line: average median. Blue line: instantaneous median.

- *Instantaneous power dissipation* (gray bars): most of the time the gray bars of the LEC-DN heuristic are higher than the proposed mapping, corresponding to a worst power distribution. In the LEC-DN, there are 451 occurrences of SPs consuming more than 23 mW (average third quartile of the LEC-DN), while in the proposed mapping the number of occurrences is 241.
- The execution time reduces from 167 to 155 ms (this execution time may be also observed in Table 3).

6.3 Proposed Mapping Cost

The proposed mapping heuristic has a small computation complexity (worst case $O(n^2)$). In fact, the execution time to execute the heuristic is small due to its *hierarchical* implementation. Table 4 details the average number of instructions to execute each step of the mapping. The cluster selection is fast since it only verifies the availability of resources. The most complex step is the mapping of non-initial tasks because it is necessary to evaluate all possible regions inside a cluster. The mapping of the non-initial task requires few instructions because a fixed region restricts the search space. The hierarchical implementation of the mapping heuristic ensures scalability because the cluster size defines the search space, not the MPSoC size. With larger MPSoCs, the cluster selection step increases its execution time (Sc5). With larger clusters, the initial tasks mapping may also increase its execution time as shown in Sc5. In both cases, the execution time to map each non-initial task is negligible compared to the time to execute them (thousands of clock cycles).

Table 4 – Average number of instructions to execute each step of the proposed heuristic.

Scenario	Cluster selection	Initial tasks mapping	Non-initial tasks mapping
Sc1	280	2497	688
Sc2	288	2337	674
Sc3	299	3057	645
Sc4	282	2588	664
Sc5	562	8584	749

7 CONCLUSIONS AND FUTURE WORKS

This paper presented a mapping heuristic that improves the thermal balance in the system. The cost function of the mapping is the energy to execute each task. The simulation of a SystemC clock cycle-accurate model of the platform generated the results for systems up to 144 processing elements. The proposed mapping distributes the load over the time, with a homogeneous utilization of PEs and a reduced number of hotspots. *It is also worth noting that the proposed mapping heuristic changes the traditional cost function used in dynamic mapping heuristics*. Instead of aiming at the reduction of the communication energy (number of hops), the proposal makes a trade-off between this parameter and the energy in the systems along the time.

Future works include: (1) integration of a lifetime model to evaluate MTTF; (2) runtime monitoring to evaluate the system load, to enhance the data obtained from profiling; (3) add the power due to

the static consumption in the power reports to obtain more accurate thermal maps.

8 ACKNOWLEDGMENTS

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

9 REFERENCES

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

[2] Wang, Z.; Chen, C.; Sharma, P.; Chattopadhyay, A. "System-level reliability exploration framework for heterogeneous MPSoC". In: GLSVLSI, 2014, pp 9-14.

[3] Chandra, V. "Quantifying workload dependent reliability in embedded processors". In: ASP-DAC, 2014, pp. 474-477.

[4] Coskun, A.; Rosing, T.; Whisnant, K. "Temperature Aware Task Scheduling in MPSoCs". In: DATE, 2007, 6p.

[5] Coskun, A.; Ayala, J.; Atienza, D.; Rosing, T.; Leblebici, Y. "Dynamic thermal management in 3D multicore architectures". In: DATE, 2009, pp.1410-1415

[6] Huang, L.; Yuan, F.; Xu, Q. "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms". In: DATE, 2009, pp. 51-56.

[7] Das, A.; Kumar, A.; Veeravalli, B. "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems". In: DATE, 2013, pp. 689–694.

[8] Das, A.; Kumar, A.; Veeravalli, B. "Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs". In: DATE, 2014, 6p.

[9] Bolchini, C.; Carminati, M.; Miele, A.; Das, A.; Kumar, A.; Veeravalli, B. "Run-time mapping for reliable many-cores based on energy/performance trade-offs". In: DFT, 2013, pp. 58–64.

[10] Hartman, A. S.; Thomas, D. E. "Lifetime improvement through runtime wear-based task mapping". In: CODES+ISSS, 2012, pp. 13-22.

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

[12] Cox, M.; Singh, A.; Kumar, A.; Corporaal, H. "Thermal-aware mapping of streaming applications on 3D Multi-Processor Systems". In: ESTIMedia, 2013, pp. 11-20.

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

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

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

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

[17] Villavieja, C; Etsion, Y.; Ramirez, A.; Navarro, N. "FELI: HW/SW Support for On-Chip Distributed Shared Memory in Multicores". In: Euro-Par, 2011, pp. 282–294.

[18] 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.

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

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

[21] Tiwari, V.; Malik, S.; Wolfe, A. "Power analysis of embedded software: a first step towards software power minimization". IEEE Transactions Very Large Scale Integration Systems, v.2(4), 1994, pp. 437–445.

[22] Murali, S.; Mutapcic, A.; Atienza, D.; Gupta, R.; Boyd, S.; De Micheli, G. "Temperature-aware processor frequency assignment for MPSoCs using convex optimization". In: CODES+ISSS, 2007, pp. 111-116.

[23] Das, A.; Kumar, A.; Veeravalli, B. "Communication and migration energy aware task mapping for reliable multiprocessor systems". In: Future Generation Computer Systems, v.30, 2014, pp. 216-228.

[24] Singh, A. K.; Kumar, A.; Srikanthan, T. "Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs". In: ACM Transactions on Design Automation of Electronic Systems, v.18(1), 2012, pp. 1-29.

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

[26] Mandelli, M.; Ost, L.; Amory, A.; Moraes, F. "Multi-Task Dynamic Mapping onto NoC-based MPSoCs". In: SBCCI, 2011, pp. 191-196.