Trading-off System Load and Communication in Mapping Heuristics for Improving NoC-Based MPSoCs Reliability

Marcelo Mandelli[1,2], Luciano Ost[3], Gilles Sassatelli[2] and Fernando 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, luciano.ost@leicester.ac.uk, sassatelli@lirmm.fr, moraes@pucrs.br

Abstract

Improving embedded systems lifetime and reliability become a major concern for the semiconductor industry. Imbalanced mapping of applications may considerably impact on system lifetime since processors and NoC links located in hotspot zones may age faster than others, compromising the overall system performance. This work proposes a dynamic mapping heuristic that makes a trade-off between processors' load and NoC communication volume, aiming to increase system reliability. Results show the proposed heuristic provides a well-balanced workload distribution while reducing communication volume. Results showed that proposed mapping reduces application execution time (average 10%) and hotspots zones when compared to conventional mapping approaches.

Keywords

MPSoC, NoC, dynamic mapping, lifetime, reliability.

1. Introduction and Related Work

Semiconductor technology evolution has enabled to reduce transistors' size and, consequently, to integrate multiple cores in the same chip. MPSoCs are expected to integrate thousands of cores by the end of the decade [1], providing a high computational performance. However, such systems became more susceptible to failures due to transistor shrinking, which can result in high power density and temperature [2][3][4]. In NoC-based MPSoCs, it is also important to consider the NoC links reliability. A higher data volume transferred through the NoC may induce links to fail, which may make processors unusable. Unusable processors induce mapping of applications onto other system processors, increasing their workload and, consequently, reducing their lifetime. Furthermore, links failures compromise system performance by increasing NoC congestion, latency and routing algorithm computation.

Such systems are likely to execute several applications in parallel. The workload imposed by such applications and its distribution impact considerably on system reliability [5][6]. Workload imbalance can generate hotspots zones (i.e. peaks of power dissipation) and consequent thermal implications, leading to unreliable system operation. Thus, task mapping has to consider the wear state of processors to better balance applications workload over the system [6].

The literature presents different task mapping heuristics to improve system lifetime reliability. Huang et al. [6] propose a task mapping allied to scheduling strategies based on simulated annealing, which also considers applications deadlines. Hartman et al. [7] present a run-time task mapping that uses sensors to capture wear patterns in the system, aiming balancing the processors' loads. Das et al. [8] propose a wear-based task mapping technique that generates mapping solutions at design-time, aiming to satisfy application deadlines and to maximize lifetime of cores and NoC links. This solution is improved in [9] by including an offline DVFS technique defining voltage and frequency levels of all cores. Bolchini et al. [10] present a run-time task mapping technique aiming to improve system reliability under energy and performance constraints. For this purpose, this work combines the technique of Das et al. [8] with a remap strategy. Chantem et al. [3] propose both dynamic task mapping and scheduling. Authors adopt the LTF-based algorithm [11] for dynamic task mapping. A scheduling technique is executed periodically, depending on system wear state conditions, trying to compensate uneven core wear states.

Most reviewed works use static mapping techniques [6][8][9][10], which are not suitable to deal with the dynamic aspects of the system. Among fully dynamic mapping heuristics [3][7], only [7] considers NoC links in the mapping decision. Embedded silicon sensors are used in [7], while our approach is software-based, which makes our system more flexible while consuming less energy and area when compared to Hartman's approach.

The main *contribution* of this work is a fully dynamic mapping heuristic that makes a trade-off between processors' workload and NoC communication volume, improving systems reliability.

2. MPSoC Architecture

This work adopts a homogeneous MPSoC architecture divided into clusters [12] as presented in Figure 1. The processing elements (PEs) are interconnected through a NoC. A PE contains a processor, a local memory (scratchpad memory, without caches), a DMA module and a router. The NoC assumes a 2D-mesh topology, credit-based flow control, round-robin arbitration, and XY routing algorithm. An external memory, called task repository, stores all applications, which are loaded into the system at runtime.

PEs may act as: Global Manager Processor (*GMP*), Local Manager Processor (*LMP*) and Slave Processor (*SP*). SPs are responsible for executing user's applications, running a simple operating system (OS) that allows task communication and multitask execution. An SP local memory is divided into *k* pages, also called *resources*. A task executing in the system is mapped in one resource. Therefore, an SP may execute *k*

simultaneous tasks. If a resource does not have a task mapped on it, it is considered *free* or *available*. LMPs are responsible for cluster control, such as executing task mapping and reclustering. GMP is a single processor in the system that accumulates the functions of an LMP and the overall system management functions, including application-to-cluster mapping, external devices access (e.g. task repository).

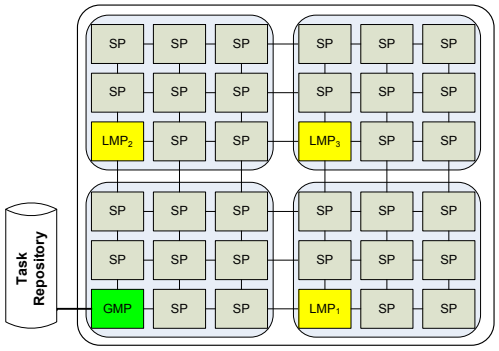


Figure 1: 6x6 MPSoC instance, with four 3x3 clusters [12].

All clusters have a fixed size, defined at design-time. At runtime, a reclustering process may modify the cluster size, according to the application mapping. When an application is mapped in 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.

3. Mapping Heuristics

This work models an application 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. The set T is divided in two subsets *iT* and *niT*, where (*iT* U *niT*) = T. The subset *iT* contains the application initial tasks, which are those without dependences to other tasks; and the subset *niT* contains non-initial tasks. A task $t_i \in T$ contains:

- a set called communicating task list, 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.
- a load value Ld_i, which represents the energy consumption [13] related to the execution of this task on an SP.

Each application is stored in the task repository with a header called *application description*. The *application description* defines the application size, the application initial tasks and information about application tasks, such as tasks' communicating task list and load.

This work adopts a mapping process divided into three steps, which may vary according to the employed mapping heuristic:

- *cluster selection:* whenever a new application is required to be mapped, the system alerts the GMP, which verifies if the system has available resources to map the entire application. In not, the application is scheduled to be mapped later. Otherwise, the GMP selects a cluster to map the required application, according to the employed heuristic. Then, the required *application description* is sent to the LMP of the selected cluster.
- *initial task mapping:* the LMP of the selected cluster receives the *application description.* Then, the LMP maps the application initial tasks inside the cluster, according to the used heuristic. The mapping of initial tasks will start the application execution.
- *non-initial task mapping*: the mapping of non-initial tasks occurs whenever a given task t_i needs to communicate with a non-mapped task t_j. In this situation, the mapping of t_j is requested by t_i to the LMP of the cluster. LMP maps the task t_j on the SP selected by the adopted heuristic.

Three mapping heuristics are evaluated in this work. Section 3.1 presents the reference mapping heuristic proposed in [14], which main cost function is to reduce communication volume transferred through the NoC. Section 3.2 proposes a heuristic aiming to balance processors' load in the system, and 3.3 proposes a heuristic that aims to make a trade-off between communication volume reduction and processor load balance. All heuristics are detailed according to the three-step mapping process previously discussed.

3.1 Premap-DN

The Premap-DN heuristic reduces the communication energy through the NoC by approximating communicating tasks that exchange a high communication volume. This heuristic acts as follows in the three mapping steps:

- *cluster selection:* selects the cluster *selected_cluster* with the large number of available resources to map an application.
- *initial task mapping:* this heuristic evaluates the SPs inside the *selected_cluster,* selecting the SP with the largest *region_free*. The function *region_free*(sp_i, *n_hops*) returns the total number of available resources of the set containing sp_i and all SPs up to *n_hops* hops from sp_i, as detailed in Figure 2.

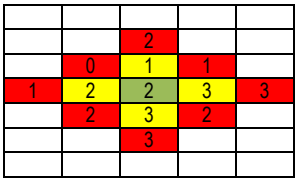


Figure 2: Hypothetical example of *region_free*($sp_{2,2}$, *2)* in a 5x5 cluster, being sp_i the central SP. The numbers inside at each SP corresponds to the number of available resources. The *region load*($sp_{2,2}$, *2*) is then 25.

- *non-initial task mapping*: when a task t_i is required to be mapped, the heuristic creates a list with all communicating tasks with t_i already mapped onto the SPs within the cluster. Next, a bounding box rectangle is defined, containing all mapped communicating tasks. When there are more than one mapped communicating tasks, the bounding box is increased by one hop 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. In the case of just one mapped communicating task t_j, this heuristic will map t_i on a SP as close as possible to the one where t_j is mapped. Otherwise, the heuristic will map t_j onto the SP with the lowest communication energy cost according to the volume based energy model proposed by Hu et al [15]. In both cases, if there are no available SPs inside the bounding box, it is increased by one hop.

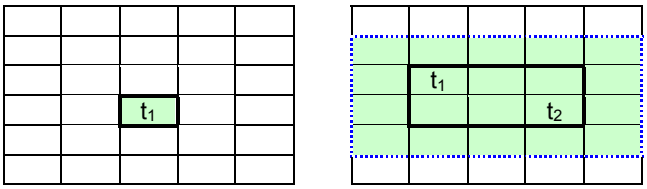


Figure 3: (a) search space when one communicating tasks is already mapped (t_1); (b) search space when more than one communicating task is already mapped (t_1 and t_2). Thicker lines correspond to the original bounding box, dashed lines to the bounding box increased by one hop.

This heuristic also uses a *premap* method with the goal to group tasks that exchanges a high volume of communication in the same SP. The *premap* defines tasks' addresses before the request to map them. The *premap* is invoked when the SP is available, and a task t_i is assigned to it. In such a case, the non-mapped tasks communicating with t_i are assigned to the same processor, according to the communication volume.

3.2 Proposed Load (L) heuristic

This heuristic uses the task load Ld to distribute tasks and balance processor load of the system. For this purpose, this heuristic assumes that each $sp_i \in SP$ has a total computational load (TL_i), which corresponds to the sum of loads (Ld) of all executed tasks and the tasks that are currently being executed on this processor. Whenever a task is mapped onto sp_i, the TL_i is updated. The three steps of the mapping process for this heuristic are:

- *cluster selection:* this heuristic first computes the total computational load *load(c_i)* of each cluster c_i by summing the TL of each SP inside the cluster. Then, the cluster with the smallest *load(c_i)* is selected.
- *initial task mapping:* this heuristic evaluates all SPs inside the cluster and maps an initial task on the SP with lowest load Ld.
- *Non-initial task mapping*: in this step, this heuristic acts as in the initial task mapping, choosing a SP inside the cluster with the lowest Ld.

3.3 Proposed Load-Communication (LC) heuristic

This heuristic makes a trade-off between processor load and reduction of communication volume. Thus, this heuristic also assumes each $sp_i \in SP$ has a total computational load (TL_i).

- *cluster selection:* this heuristic selects the cluster with the smallest *load(c_i)*, as the previous heuristic.
- *initial task mapping:* this heuristic divides the initial task process into two phases. The first phase selects the SP *selected_sp* with the smallest *region_load* to map an initial task. The *region_load*(sp_i, *n_hops*) returns the average TL from the set containing an sp_i and all SPs up to *n_hops* hops from sp_i, as detailed in Figure 4. If there is more than one initial task, it is created a set with all SPs up to *n_hops* from the *selected_sp*, selecting the SP of this set with the smallest load.

Figure 4: Hypothetical example of *region load* in a 7x7 cluster, being sp_i the central SP, and n_*hops*=3 (the number of SPs inside the region is 25): *region load*($sp_{3,3}$, *3*) = 4100/25=164. The numbers inside at each SP corresponds to the processor load.

- *Non-initial task mapping*: this heuristic uses a similar bounding box search method used in the Premap-DN. The only difference is that the bounding box is increased by one hop in both cases: when there is one and when there are more than one communicating tasks mapped in the cluster. Figure 5 illustrates the mapping search space in the cluster. In both cases, this heuristic selects the SP inside the bounding box with the lowest TL.

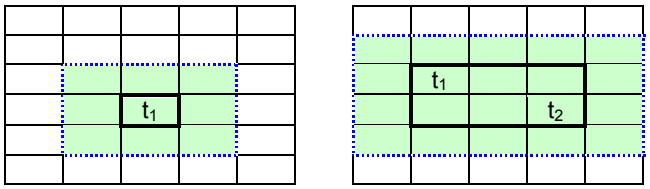


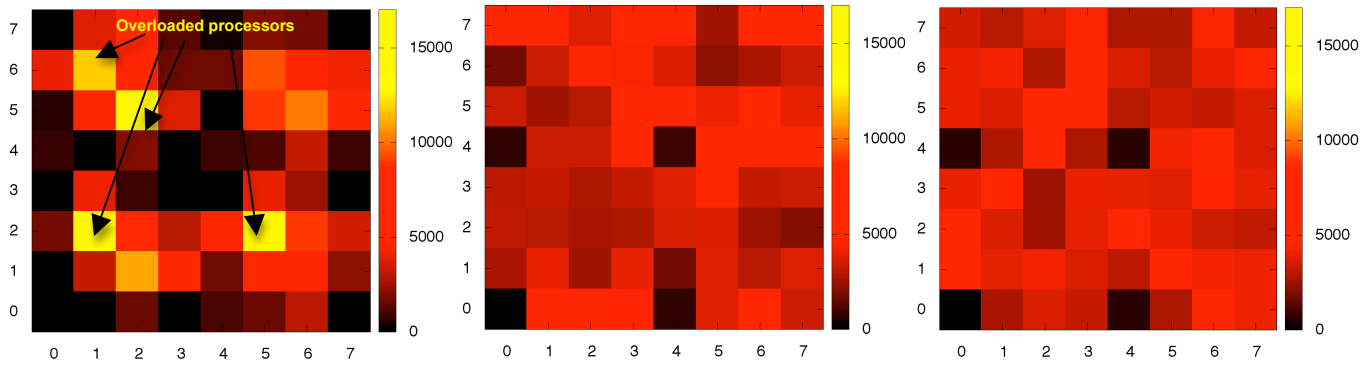
Figure 5: (a) search space when one communicating tasks is already mapped (t_1); (b) search space when more than one communicating task is already mapped (t_1 and t_2). Thicker lines correspond to the original bounding box, dashed lines to the bounding box increased by one hop.

4. Results

This Section employs an OVP [16] NoC-based MPSoC platform model to compare the mapping heuristics in terms of processor load distribution, communication volume, and total execution time. For this, purpose, five applications are used as benchmarks:

- 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.

Six different scenarios are evaluated, varying the number of applications and using an 8x8 MPSoC instance with 4x4 cluster size, as shown in Table 1.



(a) PREMAP-DN (b) LC (c) L

Figure 6: Thousands of instructions executed by each PE, for the 3 heuristics, scenario 6. Black square are manager PEs, not considered in the evaluation.

Table 1: Evaluated scenarios.

Scenario	Applications	Number of Applications	Number of tasks
1	8 x DTW, 8 x MPEG, 8 x DJK, 8 x SYN1, 8 x SYN2	40	360
2	8 x DJK, 11 x SYN1, 11 x SYN2	30	312
3	21 x MPEG, 12 x DJK, 5 x SYN1, 12 x SYN2	50	381
4	4 x DTW, 6 x MPEG, 14 x DJK, 7 x SYN1, 13 x SYN2	44	394
5	8 x DTW, 11 x MPEG, 10 x DJK, 11 x SYN1, 9 x SYN2	49	435
6	12 x DTW, 5 x MPEG, 8 x DJK, 13 x SYN1, 12 x SYN2	50	493

4.1 Load Distribution

Figure 6 presents the number of instructions executed by each SP, for the three heuristics. Note Figure 6(a) illustrates the presence of hotspots (yellow squares), i.e., some PEs execute a larger number of instructions, while others execute a small number of instructions. Figure 6(b) and Figure 6(c) present a uniform load distribution for the LC and L heuristics. Such different behavior highlights the benefits of using heuristics that consider the load in the mapping process to avoid hotspots, therefore increasing the system lifetime.

Table 2 presents the average number of instructions executed per SP and its standard deviation for all evaluated scenarios. Results show that the average number of executed instructions per SP, for the same scenario, has small variations for the different heuristics. This is an expected result, since the workload is the same.

Most relevant result in Table 2 is the standard deviation value. A small standard deviation value represents a better load distribution among the SPs. The L and LC heuristics reduces the standard deviation in average 68% and 72% respectively when compared to the PREMAP-DN heuristic. The reason explaining this result is the fact that the PREMAP-DN heuristic does not take into account the tasks' load in the mapping decision. In addition, the PREMAP-DN heuristic tends to group tasks together in the same SP, eliminating communication through the NoC. Furthermore, the L heuristic has an average reduction of 13% compared to LC, since it has a larger search space searching inside the whole cluster.

Table 2: Average number of instructions per SP and standard deviation on the number of executed instructions per SP (thousands of instructions).

	PREMAP-DN		LC		L	
Scenario	Avg./SP	**Std. Dev.**	Avg./SP	**Std. Dev.**	Avg./SP	Std. Dev.
1	2,812	3,165	2,790	1,153	2,783	873
2	2,402	3,402	2,413	777	2,422	805
3	2,326	2,662	2,348	923	2,363	868
4	3,055	3,497	3,026	1,266	3,064	1,115
5	3,392	3,888	3,412	1,260	3,433	1,081
6	3,914	4,189	3,814	1,174	3,829	836

4.2 Communication Volume

Table 3 shows the total communication volume in flits transferred through the NoC for each scenario. The PREMAP-DN heuristic presents the large reduction in the communication volume, being used as the reference mapping. The LC and L heuristics increase the communication volume in 81 and 154% (average values), respectively, when compared to the PREMAP heuristic. This happens since these heuristics have as main function to distribute tasks along the SPs, inducing the use of the NoC. Otherwise, as explained before, the PREMAP-DN heuristic tends to map tasks together in the same SP. In this case, there is an intra-SP communication between tasks. Moreover, the LC heuristic reduces the total communication volume in average 29% compared to the L heuristic. The LC heuristic considers the distance of the communicating tasks in the mapping decision, explaining the obtained results.

Table 3: Total communication volume (thousands of flits) and the difference compared to the reference mapping**.**

Scenario	PREMAP-DN	LC		L	
	Reference		**Diff. %**		**Diff. %**
1	1,771	3,048	+72.15	4,128	+133.12
2	1,417	3,070	+73.34	4,134	+133.42
3	1,982	2,515	+42.05	3,444	+94.49
4	1,956	2,761	+55.91	4,043	+128.32
5	2,143	3,803	+114.78	5,637	+218.34
6	2,528	4,057	+129.08	5,688	+221.22

4.3 Execution Time

Table 4 compares the execution time of each scenario. Results show the LC heuristic reduces the total execution time by 13% and 8% (average values) when compared to the PREMAP-DN and L heuristics. This result comes from the load distribution and communication energy trade-off used by the LC heuristic.

The PREMAP-DN increases the execution time because it increases the processor sharing (communicating tasks are mapped in the same SP). Distributing tasks and using as much as possible SPs of the system at the same time reduces the execution time, minimizing the processor sharing. The L heuristic increases the total execution time for two reasons. First, due to the larger search space, the heuristic takes longer to execute. Second, by not considering the NoC traffic in the mapping cost function, a large communication volume transferred through the NoC induces congestion and increases latency.

Table 4: Execution time (thousands of clock cycles).

Scenario	PREMAP-DN	LC	L
1	18,520	16,514	18,854
2	18,939	16,893	18,230
3	17,083	14,866	17,409
4	23,923	18,933	18,977
5	21,139	18,927	20,322
6	25,998	22,382	24,008

4.4 Reliability

The reliability of a given system is proportional to its use [5], i.e. processors with the highest usage have more probability to suffer failures. The same happens with NoC links: an increased use of a given link will reduce its lifetime. The PREMAP-DN heuristic compromises processors reliability due to its behavior that tries to group tasks at the same SP to reduce communication energy. At the same way, the L heuristic compromises links reliability by increasing communication volume in 154% and 29% compared to the PREMAP-DN and LC heuristics, respectively.

5. Conclusion and Future Work

This work proposed a task mapping heuristic that makes a trade-off between processors load and communication volume – *LC heuristic*. This heuristic is compared with two heuristics: one reduces the communication volume and energy, and another evenly distributes the system workload. Results show the proposed heuristic reduces communication volume without compromising processors workload balance. Further, the proposed heuristic reduces the total execution time in average 10% compared to the other heuristics.

Future works aim to incorporate a reliability model in the MPSoC architecture to estimate the system mean-time-to-failure (MTTF).

6. Acknowledgements

The Authors would like to thank Imperas Software Ltd. (www.imperas.com) and Open Virtual Platforms (www.OVPworld.org) for their support and access to their models and simulator. Fernando Moraes is supported by CNPq projects 472126/2013-0 and 302625/2012-7, and CAPES project CAPES-COFECUB 708/11.

7. References

[1] International Technology Roadmap for Semiconductors. Available at: http://www.itrs.net/reports.html. February 2013.

[2] Meyer, B; et al. “Cost-effective lifetime and yield optimization for NoC-based MPSoCs”. In: ACM Transactions on Design Automation Electronic Systems, vol. 19(2), 2014.

[3] Chantem, T.; et al. “Enhancing multicore reliability through wear compensation in online assignment and scheduling”. In: DATE, 2013, pp. 1373 -1378.

[4] Wang, Z; et al. “System-level reliability exploration framework for heterogeneous MPSoC”. In: GLSVLSI, 2014, pp 9-14.

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

[6] Huang, L.; et al. “Lifetime reliability-aware task allocation and scheduling for MPSoC platforms”. In: DATE, 2009, pp. 51-56.

[7] Hartman, A., et al. “Lifetime improvement through runtime wear-based task mapping”. In: CODES+ISSS, 2012, pp. 13-22.

[8] Das, A.; et al. “Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems”. In: DATE, 2013, pp. 689–694.

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

[10] Bolchini, C.; et al. “Run-time mapping for reliable many-cores based on energy/performance trade-offs”. In: DFT, 2013, pp. 58–64.

[11] Chen, J-J. et al. "Approximation Algorithms for Multiprocessor Energy-Efficient Scheduling of Periodic Real-Time Tasks with Uncertain Task Execution Time". In: RTAS, 2008, pp. 13-23.

[12] Mandelli, M.; Castilhos, G.; Moraes, F. “Enhancing Performance of MPSoCs through Distributed Resource Management”. In: ICECS, 2012, pp. 544-547.

[13] Rosa, F.; Ost, L.; Rosa, T.; Moraes, F.; Reis, R. “Fast Energy Evaluation of Embedded Applications for Many-core Systems”. In: PATMOS, 2014, pp 1-6.

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

[15] Hu, W.; et al. “An Efficient Power-Aware Optimization for Task Scheduling on NoC-based Many-core System”. In: CIT, 2010, pp. 171–178.

[16] OVP 2014, www.ovpworld.org/technology_ovpsim.php.