Multi-Task Dynamic Mapping onto NoC-based MPSoCs

Marcelo Mandelli[1], Alexandre Amory[1], Luciano Ost[2], Fernando G. Moraes[1]
[1]PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
[2]LIRMM – 161 rue Ada, Cedex 05 – Montpellier – 34095 – France
marcelo.mandelli@acad.pucrs.br, alexandre.amory@pucrs.br, ost@lirmm.fr, fernando.moraes@pucrs.br

ABSTRACT
Task mapping defines the best placement of a given task in the MPSoC, according to some criteria, as energy or Manhattan distance minimization. The ITRS roadmap forecast in a near future MPSoCs with hundreds of processing elements (PEs). Therefore, dynamic mapping heuristics are required. An important gap is observed in the mapping literature: the lack of proposals targeting multi-task dynamic mapping. In this context, the present work proposes an energy-aware dynamic task mapping heuristic, allowing multiple tasks allocation per PE. Experimental results are executed in an actual MPSoC running distributed applications. Comparing a single-task to the multi-task mapping, the energy spent in the NoC is reduced in average by 51% (best case: 72%), with an average execution time overhead of 18%. Besides the communication energy reduction, the multi-task mapping enables a greater number of applications executing simultaneously, or smaller MPSoCs, which reduces the system cost.

Categories and Subject Descriptors
C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Interconnection architectures; C.2.1 [**Network Architecture and Design**]: Packet-switching networks; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms
Algorithms, Performance, Design, Experimentation.

Keywords
MPSoC, NoC, dynamic mapping, energy evaluation.

1. INTRODUCTION
NoC-based MPSoCs can execute simultaneously several and complex applications. Such applications are composed by tasks, with different workloads and deadlines [1]. The workload of such systems may vary dynamically at execution time, according to various criteria (e.g. user and/or performance requirements) [2][3][4]. In this context, the mapping of such tasks onto the NoC-based platform may drastically influence the system performance [5]. This influence is due to the inter-task communication traffic. An unoptimized mapping may place communicating tasks far from each other, increasing the communication latency and energy consumption, as well as increasing network traffic leading to congestion inside the NoC.

SBCCI'11, August 30-September 2, 2011, João Pessoa, Brazil.

Task mapping literature is rich, but a taxonomy classifying the mapping approaches is required. We propose a taxonomy for task mapping, according to four criteria.

Considering the moment when tasks are mapped, the following approaches may be used:
- at design time: called *static* or *offline*, it may use complex heuristics to better explore the MPSoC resources, resulting in optimized solutions. However, static mapping is not able to handle a dynamic workload.
- at run-time: called *dynamic* or *online*, require simple and fast heuristics since it may interfere with the applications execution time. Two dynamic mapping approaches are used:
  - *with resources reservation*: the mapping heuristics verify if there are enough resources in the MPSoC before mapping the application tasks.
  - *without resources reservation*: the mapping heuristics map one or more initial tasks of the applications (those without dependences to another tasks), mapping the remaining tasks when they are required. This approach may start applications faster, but some tasks may wait for available resources if the system usage is high.

Considering the number of task mapped per PEs, the following approaches may be used:
- *single-task*: only one task is assigned to each PE.
- *multi-task*: more than one task can be assigned to each PE This requires a *clustering* approach to define a group of tasks to be mapped onto the same PE.

The dynamic mapping requires an entity responsible for mapping the tasks at runtime. Such control may be:
- *centralized*: one PE is responsible to manage the mapping process. This approach is not scalable, and may lead to hot-spots in the NoC and reduce the overall performance.
- *distributed*: the MPSoC is divided in regions (clusters), and one PE in each region is responsible for executing the mapping heuristic inside it.

Finally, the mapping can be classified according to the system architecture model:
- *homogeneous*: when all PEs are identical.
- *heterogeneous*: when different PEs are used in the same system, including RISC, processors, DSPs, dedicated IPs and so on. Before the mapping it is necessary a *binding* process to define which PEs can execute a required task

Our main *goal* is to present an energy-aware heuristic for dynamic multi-task mapping, without resources reservation. The main features of the proposed mapping include: (*i*) execution of the mapping heuristics on an RTL NoC-based MPSoC platform, leading to accurate results; (*ii*) use of real applications as benchmarks; (*iii*) a clustering approach considering the communication dependence among tasks; (*iv*) the main cost function is the energy consumption in the communication infrastructure.

Table 1. Related work classified according to the proposed taxonomy for dynamic mapping heuristics.

Author	Resource reservation	Multi/Single Task	Architecture model	Control manager	Optimization Goal
Smit [6] 2005	Yes	Single Task	Heterogeneous	Centralized	Energy Consumption, QoS requirements for the applications
Ngouanga [7] 2006	Yes	Single Task	Homogeneous	Centralized	Communication volume, Computation load
Hölzenspies [8][1] 2007/2008	Yes	Single Task	Heterogeneous	Centralized	Energy Consumption, QoS requirements for the applications
Chou [9][10] 2007/2008	Yes	Single Task	Homogeneous	Centralized	Energy Consumption, Network contention
Al Faruque [3] 2008	No	Single Task	Heterogeneous	*Distributed*	Execution time, Mapping time
Mehran [11] 2008	Yes	Single Task	Homogeneous	Centralized	Mapping Time, Energy Consumption, Mapping Complexity
Wildermann [2] 2009	No	Single Task	Homogeneous	Centralized	Communication Latency, Energy consumption, Application deadlines
Schranzhofer [12] 2010	Yes	Single Task	Homogeneous	Centralized	Energy Consumption
Carvalho [13] 2010	No	Single Task	Heterogeneous	Centralized	Network contention, Communication volume
Singh [14][3] 2009, 2010	No	*Multi-task*	Heterogeneous	Centralized	Network contention, Communication volume, Energy Consumption
Proposed work	**No**	***Multi-task***	**Homogeneous**	**Centralized**	**inter-task dependence evaluation, energy consumption**

The rest of this paper is organized as follows. Section 2 presents related works in task mapping, classifying them according to the proposed taxonomy. Section 3 details the reference MPSoC platform. Section 4 presents the proposed dynamic mapping heuristic. Section 5 presents the experimental setup and results. Section 7 concludes this paper.

2. DYNAMIC TASK MAPPING RELATED WORK

Table 1 classifies recent works in dynamic task mapping according to the proposed taxonomy. The Table reveals two common features: centralized control and single-task mapping.

Even if centralized control is not scalable, this is the strategy adopted in most works. The only work presenting distributed control is the Al Faruque [4] proposal, using a 64x64 NoC. Define when a centralized control becomes a bottleneck requiring distributed control is an open issue in the literature. So far, the mapping heuristics are evaluated in NoCs with mesh topology, with dimensions inferior to 9x9.

Most works use a single-task mapping approach, assigning only one task for each PE. Such behavior must evolve to consider multiple applications running simultaneously, in an environment with processors executing multi-task operating system. The main challenge of multi-task mapping is how to group them, since the clustering approaches requires a global view of the application, while in dynamic mapping only a subset of the tasks are effectively mapped onto the MPSoC.

Singh et al. [3][14] extended the dynamic heuristics proposed by Carvalho et al. [13] to support multi-task mapping. A clustering approach is proposed, which tries to maximize the number of communicating tasks in the same PE. This technique verifies the previously mapped tasks in a given a PE to map a new ones on it: if the required task communicates with some previously mapped task, it is mapped; if not, then other PE is verified. The Authors mention that some PEs may receive only one task, underusing the system resources. The clustering approach, compared to a non-clustering approach, improve in average 15% the channel load and energy consumption, with some improvement in packet latency and execution time. The main drawback of Singh's approach is that only master-slave dependences between tasks are considered.

Other remark concerns the resource reservation. Some works reserve resources according to the number of the tasks, defining e.g. precomputed mapping templates for each application. Considering that not all tasks execute concurrently, reserve resources for all application tasks may underutilize the MPSoC, as well as require bigger systems. The dynamic mapping without reservation uses the system resources when they are effectively required.

Finally, the NoC-based MPSoC modeling limits the evaluation of the dynamic mapping heuristics, since designers should be able to simulate scenarios for long time in order to allow the occurrence and analyses of critical aspects (e.g. application area fragmentation). RTL modeling and cycle accurate simulation provides accurate results, with long simulation time. On the other hand, abstract models as TLM, enables faster simulations, but do not enables accurate performance evaluation. Some works, as [3][13], adopts a mixed modeling, with PEs described using SystemC and the NoC in VHDL. The remaining reviewed works employs abstract models (e.g. analytical model employed in [12]).

Contrary to the other works, the proposed multi-task dynamic mapping heuristic is validated using a NoC-based MPSoC platform described at the RTL level, with a clock-cycle accurate ISS describing processors. The mapping heuristic is executed in a given PE, responsible to manage the mapping requests of the remaining PEs, which execute a multi-task operating system, enabling multi-task mapping.

The present work advances the state-of-the-art for dynamic multi-task dynamic mapping heuristics, which is the most important gap observed in the mapping literature, presenting a new multi-task mapping that can be employed by actual NoC-based MPSoCs.

However, some drawbacks of the present work must be pointed out: (*i*) centralized approach – may be solved using the clustering method advanced in [4]; (*ii*) homogenous architecture – a binding process before mapping enables heterogeneous PEs; (*iii*) small NoCs – larger NoCs, as 10x10, require accurate abstract models. To have such accurate abstract models, validation at the RTL level is required beforehand develop such models.

3. MPSOC ARCHITECTURE

The proposed heuristic was implemented in a homogeneous NoC-based MPSoC platform called HeMPS [15]. Each PE, named Plasma-IP, contains a MIPS-like processor (Plasma), a local memory (RAM), a DMA controller and a Network Interface. A general view of a 2x2 instance of the MPSoC architecture is illustrated in Figure 1. Two types of Plasma-IP are used: slave (SL) and master (MP). Plasma-IP slaves are responsible to execute application tasks, while the Plasma-IP master is responsible to manage task mapping and system debug. The external memory, named *task repository*, contain all application tasks, which can be requested during the simulation According to the mapping heuristic, the Plasma-IP master maps the tasks onto the Slaves-PEs. The Plasma-IP MP can also receive debug messages from Slave-PEs, transmitting them to an external host through an Ethernet interface (not shown in Figure 1).

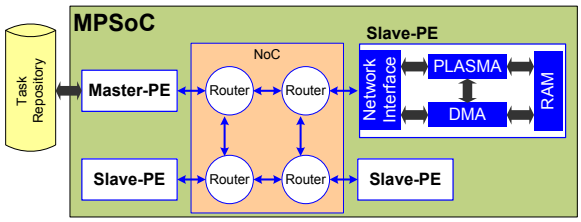

Figure 1. Block diagram of the HeMPS platform.

Each Plasma-IP SL runs a tiny operating system (named *microkernel*, whose memory footprint is around 20 KB), responsible to manage and support task execution and task communication. This microkernel is a preemptive operating system where each task uses the CPU for a pre-defined period, called *timeslice*.

All communication among tasks occurs through message passing, using a global message vector, named *pipe*, located in the microkernel, and two communication primitives: *Send*() and *Receive*(). A task executing the communication primitive *Send*() is the source task, and the one executing *Receive*() is the target task. During the execution of the *Send()* command it is verified if the target task is mapped (using a local task table). If the target task is not already mapped, the source microkernel sends a *RequestTask* message to the Plasma-IP MP, which selects the task position at run-time according to the dynamic task mapping heuristic. When the Plasma-IP MP receives the *RequestTask* message, it configures its DMA module, which accesses the task repository and transmits the target task code to the target Plasma-IP SL memory.

4. MAPPING HEURISTICS

This section describes the mapping heuristics used in this paper. Section 4.1 presents the Nearest Neighbor and the Best Neighbor mapping heuristics, used as reference for the experiments [13]. Section 4.2 details the LEC-DN (Low Energy Consumption – Dependences Neighborhood) mapping heuristic for multi-task. Section 4.3 presents the Premap clustering method, which try to group communicating tasks in the same PE. Section 4.4 presents the proposed Premap-DN, which combines Premap with LEC-DN.

4.1 Reference dynamic mappings heuristics

The Nearest Neighbor (NN) heuristic considers only the proximity of an available resource to execute a given task. NN starts searching for a free PE able to execute the target task near the source task. The search tests all *n*-hop neighbors, *n* varying between 1 and the NoC limits in a spiral way, stopping when the first PE free is found. The Path Load (PL) heuristic computes the load in each channel used in the communication path. PL computes the cost of the communication path between the source task and each one of the available resources. The selected mapping is the one with minimum cost. The Best Neighbor (BN) heuristic combines NN search strategy with the PL computation approach. The search method of BN is similar to NN, i.e., spiral searches from the source node. This avoids computing all feasible mapping solutions, as in the PL heuristic, reducing the execution time for the mapping. BN selects the best neighbor, according to PL equations, instead of the first free neighbor as in NN.

Note that the BN and PL heuristics require a monitoring infrastructure to evaluate at runtime the load at each NoC link. Such monitoring processes increases the traffic load inside the NoC, as well as the CPU utilization that executes the mapping heuristic (Plasma-IP MP), since it must receive the monitoring data and fill the data structures responsible to keep the monitoring information.

Singh et al. [14] extended BN and NN heuristics to multi-task mapping. Basically, the algorithm starts verifying if the source task (0 hops distance) is able to map the target task instead of looking for the neighbor PEs at 1 hop distance. Our work adopts a similar approach (0 hops), with a new search method (bounding box) and cost function (energy consumption).

4.2 LEC-DN

Differently from NN and BN heuristics, which map the target task as close as possible to its source task, the LEC-DN [16] considers the proximity of the target task to all communicating tasks that are already mapped. LEC-DN employs two search methods to select the PE that receives the target task. When the target task has only one communicating task already mapped, LEC-DN uses the NN search method (spiral search). If there is more than one communicating task already mapped, the LEC-DN searches for a PE inside the bounding box defined by the position of such communicating tasks. The bounding box search method uses the volume-based energy model proposed by Hu et al. [17] to select the position of the task to be mapped.

Consider the application of Figure 2(a), containing 4 tasks, where A and B are initial tasks. The mapping of task C is fired by the first communication with it. The search space to map task C corresponds to the bounding box defined by the position of A and B tasks (Figure 2(b)). Task C will be mapped nearest to task A, since according to the application task graph the communication volume A→C is higher than B→C. Note that task D is not mapped, since it depends from task C.

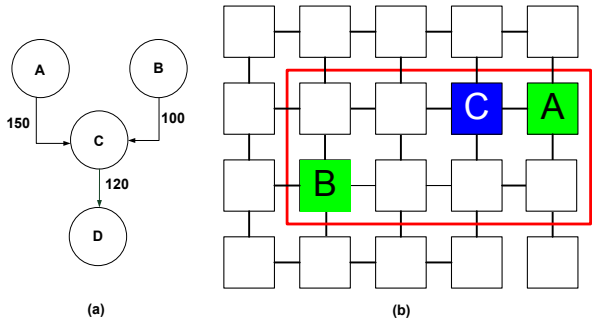

Figure 2. (a) application graph G=<T,V> describing an application, where T is the task set and V the communication volume between tasks; (b) search space to map the C task, and one possible mapping for C.

The present work extended the original LEC-DN heuristic to execute multi-task mapping, extending the search space to start in the processor that requested the new task.

4.3 Premap

The goal of this method, herein proposed, is to group a set of communicating tasks onto the same PE. The idea is not to reserve resources for the whole application. The *premap* heuristic is executed when a new task is mapped in a new PE. When a given task is *premapped,* only its placement is reserved. The effective mapping of the *premapped* tasks is executed when the task is requested.

Consider as an example the application of Figure 3(a), with 8 tasks, being t_A the initial task. A 2x2 MPSoC instances is used, resulting in three available Plasma-IP SL, each one able to execute up to 3 tasks. Figure 3(c) presents the *communication task list* (CTL), data structure contained in the Plasma-IP MP microkernel used by *premap*. Each entry of the CTL is a task t_i, containing the set $C = \{t_1, t_2, \ldots, t_n\}$, corresponding to all tasks connected to t_i, sorted according to their communication volume with t_i.

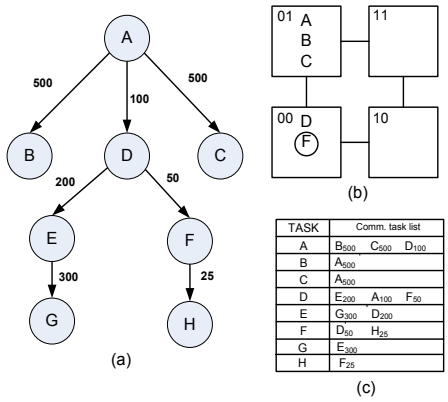

Figure 3. Premap heuristic example.

When the system execution starts, the initial tasks are mapped, according to their position defined at design time. According to the example, t_A is mapped to PE01 (Figure 3(b)). The *premap* evaluates each task t_j in the set $C(t_A)$ to be *premapped* in PE01, starting with the one with higher communication volume. In the example $C(t_A)=\{t_B,t_C,t_D\}$, and t_B will be *premapped* iff there is no task in $C(t_B)$ with a higher volume with t_A. As t_B and t_C only communicates with t_A, both are *premapped* in PE01. At this moment the method stops, since the PE has already 3 tasks assigned to it. During system execution, t_B and t_C are required to be mapped. As they were already *premapped*, it is not necessary to use the mapping heuristic, being only necessary to transmit the object codes to PE01. Next t_D is required to be mapped. The LEC-DN chooses PE00, which is the nearest PE to t_A (it could also be PE11). As t_D "opened" a new PE, the *premap* is executed. In this case $C(t_D)=\{t_E,t_A,t_F\}$. As the communication volume t_E-t_G is higher than the communication volume t_E-t_D, t_E is not *premapped*. Next, as t_A is already mapped, it is not evaluated. Finally, t_F is evaluated, and it is *premapped* with t_D because the communication volume t_F-t_D is higher than t_H-t_F.

Figure 4 shows the implementation of the *premap* heuristic. The heuristic begins assigning to the set $N(t_i)$ the non-mapped tasks of $C(t_i)$ (line 2). The next step evaluates each task d_i from $N(t_i)$, to choose which tasks will be premapped onto p_i. This evaluation (line 5-16) happens while p_i has less than TASKS_PER_PE (the maximum number of tasks supported per PE) mapped/premapped tasks onto it, or if all possible tasks in $N(t_i)$ were already evaluated. For each task d_i, the first task h_i in its CTLC(d_i) is obtained (line 7). So, the task h_i is compared to the task t_i (line 8) to verify if d_i has the highest communication volume with t_i. In an affirmative case, t_i is premapped onto p_i, also increasing the p_i number of mapped/premapped tasks (line 9-12). Otherwise, if available, other task from $N(t_i)$ is evaluated.

```
Input: The PE p_i, the task t_i mapped onto p_i
Output: A set of tasks premapped onto p_i
1.      // N(t_i) contains all non-mapped tasks which t_i communicates with
2.      N(t_i) ← non-mapped_tasks(C(t_i))
3.      // Get the first task in the N(t_i)
4.      d_i ← first(N(t_i))
5.      WHILE tasks(p_i)<TASKS_PER_PE or !end(N(t_i)) DO
6.          // Get the first task h_i (with highest communication volume) in C(d_i)
7.          h_i ← first(C(d_i))
8.          IF h_i=t_i THEN
9.              // premap d_i onto p_i
10.             premap(d_i,p_i)
11.             // increase the number of mapped/premmaped tasks onto p_i
12.             tasks(p_i)++
13.         END IF
14.         // Get the next task in the N(t_i)
15.         d_i ← next(N(t_i))
16.     END WHILE
```

Figure 4. *Premap* mapping heuristic pseudocode.

4.4 Proposed Premap-DN

The *premap-DN* optimizes the multi-task mapping by integrating the LEC-DN and the *premap* clustering method. Figure 5 shows the integration of LEC-DN and the *premap* clustering method in the Plasma-IP MP *microkernel*. When a task t_i is requested to be mapped by a PLASMA-IP SL, it sends a REQUEST_TASK message to the PLASMA-IP MP. The PLASMA-IP MP receives this message and starts executing the mapping flow. First, it checks if there is some available resource in the system. If there is no available resource, the task is scheduled to be mapped later. The schedule mechanism is out of the scope of this work. In the other case, the flow proceeds to the next step.

The next step verifies if the target task is already *premapped*. In an affirmative case, the task is allocated to the assigned PE; otherwise, the LEC-DN mapping heuristic is executed.

The LEC-DN executes and returns the PE p_i where the task t_i is mapped on. After this, it checks if p_i has just one task, which means that it contains just the task t_i. If it is true, the *premap* method is called to find the tasks communicating with t_i to be *premapped* onto p_i and the flow is finished.

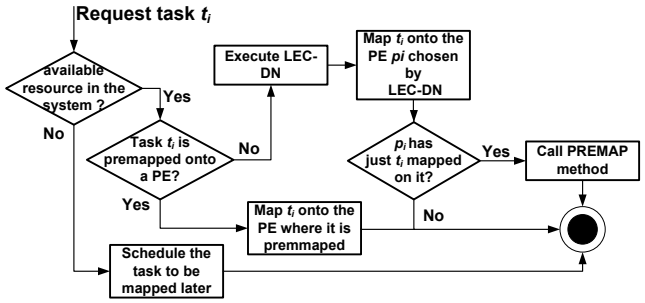

Figure 5. Integration of the heuristics in the microkernel.

5. RESULTS

The MPSoC used to evaluate the mapping heuristics is configured as follows: 2D-mesh topology, XY routing algorithm, 16-bit flit size, packets with 128 flits and credit-based control flow. The MPSoC is sized as follows: 7x6 (1 Plasma-IP MP, 41 Plasma-IP SL) for single-task mapping and 3x5 (1 Plasma-IP MP, 14 Plasma-

IP SL) for multi-task mapping. Such configuration was chosen to have, if possible, the same number of simultaneous tasks executing in the systems. This criterion enables a fair comparison among the heuristics, since the MPSoC occupation is the same for both single and multi-task mapping. Considering up to 3 tasks mapped per processor, the multi-task mapping may map 42 tasks (14 Plasma-IP SL * 3 tasks per PE), while the single-task may map up to 41 tasks.

The energy model was calibrated using the ST/IBM CMOS 65 nm technology at 1.0 V, adopting clock gating, and a 100 MHz clock frequency. Additionally, in order to evaluate the BN heuristic, a monitoring infrastructure is included in MPSoC to obtain the load in each NoC link.

Real and synthetic applications are modeled in C code. The C code contains the communication primitives enabling the communication among tasks, and the computation time of each task. Five application scenarios were evaluated:

A. MPEG (12 tasks), VOPD (12 tasks), Vehicle (10 tasks) and Circuit (4 tasks);
B. MPEG, VOPD, Segmentation Image (6 tasks) and Synthetic (6 tasks);
C. MPEG and VOPD;
D. MPEG, Vehicle and Circuit.
E. MPEG, MWD (12 tasks) and VOPD

Scenarios *A*, *B* and *E* contains 38, 36 and 36 tasks, respectively. Such scenarios correspond to an MPSoC occupation equal to 93% (scenario A) and 86% (scenario B and E). The dynamic mapping heuristics are stressed in these two scenarios, since the search space is drastically reduced when almost all tasks are already mapped. Scenarios *C* and *D* contain 24 and 26 tasks respectively, enabling to evaluate the mapping heuristics when the search space is not a constraint.

5.1 Execution time evaluation

Table 2 presents the total execution time to execute 10 application iterations, for each evaluated scenario, considering up to three tasks mapped to the same PE. The average execution time overhead compared to the single-task LEC-DN, is 14%, 13%, 16% and 19% for the NN, BN, LEC-DN multi-task and Premap-DN heuristics respectively. This overhead in the multi-task approach can be explained due to the time-sharing between tasks in the same processor. The scheduling algorithm shares the processor execution in *timeslices* among tasks. Thus, during a *timeslice*, a task can stay long periods in an idle state, waiting, for example, to receive a particular message. This results in a waste of time, because when this task becomes idle, a new task could be executed.

Table 2. Execution time, in clock cycles (thousands).

Scenario	Single-task	Multi-task – up to 3 tasks per processor			
	LEC-DN	NN	BN	LEC-DN	Premap-DN
A	4,623	5,419	5,329	5,787	5,755
B	2,350	2,436	2,555	2,603	2,483
C	1,700	1,932	1,950	1,912	2,042
D	4,591	5,465	5,430	5,251	5,454
E	1,866	2,168	2,027	2,143	2,312

Multi-task mapping heuristics can influence the total execution time if the communicating tasks are not grouped correctly, since the network congestion may increase due the lower number of available network channels. The execution time overhead of the proposed Premap-DN is 4%, 4.9% e 2.7% compared to NN, BN e LEC-DN multi-task heuristics. This is a small overhead since the energy consumption reduction is considerable better than the other heuristics (next subsection). According to this evaluation, the mapping of several tasks in the same processor increases the processors utilization with a small penalty in the execution time.

5.2 Energy evaluation

Table 3 presents the total energy consumption in the NoC (routers and links). The multi-task mapping reduces in average 56% the communication energy, when compared to single-task heuristics. As shown in Figure 6, the Premap-DN heuristic achieves the highest reduction in most cases (up to 72% reduction), except for the scenario B, where the NN heurist has the highest reduction. The reduction in the communication energy is due to the adopted *premap* clustering method. Grouping tasks in the same processor reduces the network traffic, and consequently the NoC energy consumption.

Table 3. Total communication energy consumption (in nJ).

Scenario	Single-task	Multi-task – up to 3 tasks per processor			
	LEC-DN	NN	BN	LEC-DN	Premap-DN
A	2,829	2,098	1,332	1,476	1,260
B	1,929	618	901	678	755
C	1,189	449	371	462	370
D	2,022	1,007	888	1,249	559
E	1,228	599	596	617	521

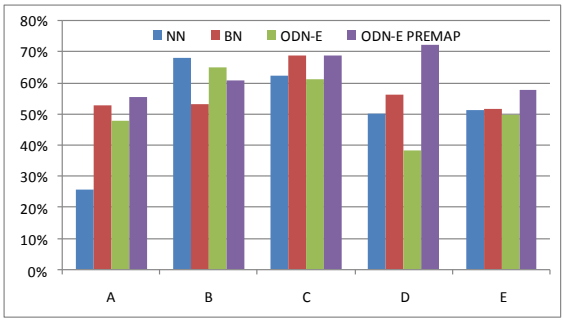

Figure 6. Reduction on the communication energy normalized to the LEC-DN single-task mapping.

Comparing Premap-DN to other heuristics, the communication energy reduction is 18.6%, 14.3% and 18.8% for NN, BN and LEC-DN-MT, respectively. The BN heuristic obtains the best results in total execution time, while increasing the average communication energy consumption approximately 19.7%, compared to other heuristics. The monitoring infrastructure used to obtain NoC data at runtime explains the increased energy consumption. Furthermore, this infrastructure also implies the use of monitoring packets, which are not taken into account in the results and cause higher consumption in communication energy.

The LEC-DN is the reference heuristic for single-task mapping, since in all scenarios it obtained the higher energy reduction compared to the other single-task mapping heuristics. However, observing the Table 3, the use of this heuristic for multi-task mapping does not achieve the expected results. NN and BN heuristics have higher energy reduction in some scenarios due to their behavior: communicating tasks are always mapped in the same PE, as long as possible. On the other side, LEC-DN considers all tasks dependences, and may spread the tasks in the MPSoC. This behavior induced the development of the *Premap-DN* heuristic that pre-assigns the communicating tasks to the same processor.

5.3 Spatial Task Distribution

Figure 7 presents the spatial task distribution for a multi-task mapping, where most PEs receives 3 tasks. The MPEG (blue) and Circuit (yellow) applications are mapped in continuous regions, while a small fragmentation is observed in the application VOPD (red) and Vehicle (green). Note also that 3 PEs received tasks from 2 distinct applications. This Figure illustrate that the proposed heuristic effectively reduce the distance among communicating tasks, using the energy consumption as cost function, even in situation where almost all MPSoC's resources are allocated to other tasks.

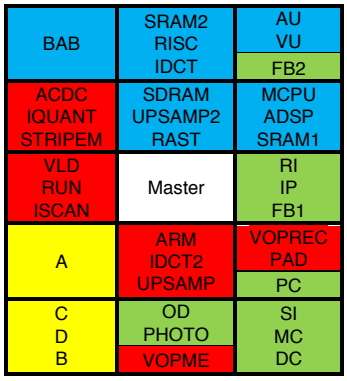

Figure 7. Spatial task distribution for scenario A, Premap-DN multi-task mapping (15 PEs). In blue: MPEG, red: VOPD, green: Vehicle, yellow: Circuit, white: Plasma-IP MP.

6. ACKNOWLEDGMENTS

The Authors acknowledge the support of CNPq and FAPERGS, projects 301599/2009-2 and 10/0814-9, respectively.

7. CONCLUSION AND FUTURE WORK

This work improved the quality of the dynamic mapping for NoC-based MPSoCs platforms, proposing a new heuristic with the following features: (*i*) multi-task mapping; (*ii*) inclusion of the cost of the already mapped tasks connected to the task being mapped, previous approaches considers only master-slave connections; (*iii*) use the communication energy as cost function, not only the hop number.

This paper also compares multi-task mapping heuristics with single-task ones in NoC-based MPSoC. Results demonstrate that the multi-task approach reduces the energy consumption, with a small execution time overhead. The communication energy consumption is reduced up to 72% since the multi-task approach reduces the distance among the tasks. The overall energy consumption can also be reduced since a smaller NoC-based platform can be used to map the same number of tasks. The time-sharing among tasks in the same processor inherent to the multi-task approach causes an execution time overhead of up to 19%. However, this overhead can be considered small since the NoC size has been reduced from 41 Plasma-IP SL, for single-task mapping, to 14 Plasma-IP SL, for multi-task mapping.

Future works includes the proposition of decentralize mapping, evaluation the trade-off of controlling the decentralized method per regions or applications.

8. REFERENCES

[1] Hölzenspies, P.K.F.; et al. "Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)". In: DATE, 2008, pp.212-217.

[2] Wildermann, S.; et al. "Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures". In: FPT', pp. 514 - 517.

[3] Singh, A. K.; et al. "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms". Journal of Systems Architecture, vol. 56(7), 2010, pp. 242 - 255.

[4] Faruque, M.A.; et al. "ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication". In: DAC, 2008, pp. 760-765.

[5] Carvalho, E.; et al. "Evaluation of Static and Dynamic Task Mapping Algorithms in NoC-Based MPSoCs". In: SOC, 2009, pp. 87-90.

[6] Smit, L.T.; Hurink, J.L.; Smit, G.J.M. "Run-time mapping of applications to a heterogeneous SoC". In: SoCC, 2005, pp.78-81.

[7] Ngouanga, A.; et al. "A contextual re-sources use: a proof of concept through the APACHES platform". In: DDECS, 2006, pp.42-47.

[8] Holzenspies, P.K.F.; Smit, G.J.M.; Kuper, J. "Mapping streaming applications on a reconfigurable MPSoC platform at run-time". In: SoCC, 2007. pp. 1-4.

[9] Chou, C-L.; Marculescu, R. "Incremental Run-time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels". In: CODES+ISSS, 2007. pp. 161-166.

[10] Chou, C-L.; Marculescu, R. "User-Aware Dynamic Task Allocation in Networks-on-Chip". In: DATE, 2008, pp. 1232-1237.

[11] Mehran, A.; Khademzadeh, A.; Saeidi, S. "DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip". IEICE Electronics Express, vol. 5-13, 2008, pp. 464-471.

[12] Schranzhofer, A.; et al. "Dynamic and adaptive allocation of applications on MPSoC platforms". In: ASP-DAC, 2010, pp. 885-890.

[13] Carvalho, E.; Calazans, N.; Moraes, F.. "Dynamic Task Mapping for MPSoCs". In: IEEE Design and Test of Computers, vol. 27(5), 2010, pp. 26-35.

[14] Singh, A.K.; et al. "Efficient heuristics for minimizing communication overhead in NoC-based heterogeneous MPSoC platforms". In: RSP, 2009, pp. 55 - 60.

[15] Carara, E.; et al. "HeMPS - a Framework for NoC-based MPSoC Generation". In: ISCAS'09, 2009, pp. 1345 - 1348.

[16] Mandelli, M.; et al. "Energy-Aware Dynamic Task Mapping for NoC-based MPSoCs". In: ISCAS, 2011, pp. 1676-1679.

[17] Hu, J.; Marculescu, R. "Energy-aware mapping for tile-based NoC architectures under performance constraints". In: ASP-DAC, 2003, pp. 233-239.