Partitioning and Dynamic Mapping Evaluation for Energy Consumption Minimization on NoC-Based MPSoC

Eduardo Antunes, Matheus Soares, Alexandra Aguiar, Sergio Johann F., Marcos Sartori, Fabiano Hessel, César Marcon

PPGCC - Post-Graduation Program in Computer Science
PUCRS - Pontifical University Catholic of Rio Grande do Sul
Corresponding author: cesar.marcon@pucrs.br

Abstract

Software complexity has increased considerably over recent years, needing special target architectures as NoC-based MPSoCs to fulfill the heavy storage, communication and computation requirements. The design of these systems requires efficient methodologies aggregating partitioning and mapping. In these sense, this paper explores partitioning and mapping influence on energy consumption of homogeneous NoC-Based MPSoC. In addition, it compares two strategies to achieve efficient dynamic mappings: one that map tasks directly onto processors and another one that applies a previous static task-partitioning and uses this information to choose the dynamic task mapping. Experiments with various synthetic and four embedded applications show the efficiency of the second strategy that minimizes an average of 23.5% on energy consumption.

Keywords

Partitioning, Mapping, MPSoC, NoC

1. Introduction

Recent years have brought a large quantity of application, demanding huge computational power, large memory sizes, reduced energy consumption and efficient communication, which boast the research and development of special target architectures, like a NoC-based MPSoC. This one implements the complete system functionality into a single chip and support the heavy communication requirements of hundreds processors with efficient energy consumption.

From the processing point of view, homogeneous MPSoCs are those composed by processors of the same type and heterogeneous MPSoCs are those composed by at least two processors with different architectures.

Heterogeneous MPSoCs can support a wide variety of applications, since each processor has specific computation and communication features. Otherwise, homogeneous MPSoCs are easier to program, increase the mapping and partitioning possibilities, and enable global load balancing through application-task migration. Furthermore, the homogeneity may minimize the global energy consumption and area occupation for some set of applications [1].

This work employs homogeneous NoC-based MPSoC as target architecture, and presents a partial design flow containing the application-task partitioning into groups of tasks, where each group is mapped onto tiles of the target architecture. Whereas tile is a limited area of target architecture, comprising a processor, a router, a local memory and auxiliary circuits.

Several works relate to task mapping onto NoC-based architectures and some ones describe the tasks partitioning into groups ([2 - 13]), **but none evaluate the effect of using static partitioning as a previous step of the dynamic mapping**. Here, we compare two approaches: (i) the traditional one that during the run time map tasks onto processors and (ii) the one proposed here, which performs a previous analysis of tasks affinity by a partitioning step and uses this information to choose fast and efficient mappings.

Moreover, several works uses the same name "mapping" to define both mapping and partitioning, while the name "partitioning" is used only to explore hardware/software division.

This paper is organized as follows. Section 2 presents the partitioning and the mapping problem formulation together with the underlying data structures and the energy model. Section 3 shows the language used to describe parallel applications. Section 4 describes the methodology and the tools used to accomplish the experimental results. Section 5 employs an application to exemplify the methodology. Section 6 shows a synthetic application tool and experimental results, and Section 7 concludes the paper.

2. Problem Formulation

The complete homogeneous MPSoC design implies several steps with some specificity according to the application description nature and the target architecture. Here, we describe two design activities, which are the partitioning and the mapping.

Parallel applications are described as a set of communicating tasks. According to some requirements (e.g. energy consumption minimization) and some constraints (e.g. memory size limit, quantity of target processors) a given design flow enables associating tasks of the parallel application to tiles of the target architecture.

The traditional flow associates tasks directly to tiles, which is called here as *task mapping*. On the other hand, our flow considers tasks affinity to generate groups. The grouping of all application tasks, which is the *task partitioning* activity, generates a partition. The next step is to perform the selection of the best place that each group of tasks will be associated, which is the *task-group mapping* onto tiles activity.

To better understand the concepts of partitioning and mapping of the proposed flow, Figure 1 exemplifies the partitioning of a hypothetical application composed by 22 tasks into 6 groups and the corresponding mapping of these task-groups onto tiles of 2D-mesh NoC architecture. The application is composed by a set of parallel communicating tasks $T = \{t_1, t_2, \ldots, t_{22}\}$. The tasks partitioning, which is represented by continuous arrows, generates $G = \{g_1, g_2, \ldots, g_6\}$ that is a set of task-groups.

Finally, task-groups mapping onto tiles is represented by the dotted arrows. These one associates each element of G to an element of the set of NoC tiles $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_6\}$. In addition, each tile contains a single element of the set of processors $P = \{p_1, p_2, \ldots, p_6\}$.

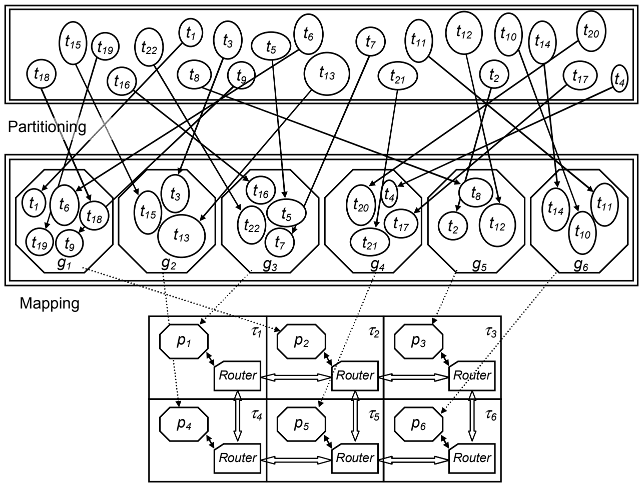


Figure 1 – Partitioning and mapping understanding.

2.1. Partitioning and Mapping Complexities

The partitioning of tasks into groups is an activity with complexity proportional to the Bell number *O(Bell(**n**))* [15], where **n** is the quantity of tasks, since there is no order relation between task-groups and even within a group. The task-groups mapping onto tiles (containing processors) of the target architecture is *O(**t**!)* complex, where ***t*** is the quantity of tiles, because it reflects all combinations of positions of groups in all tiles.

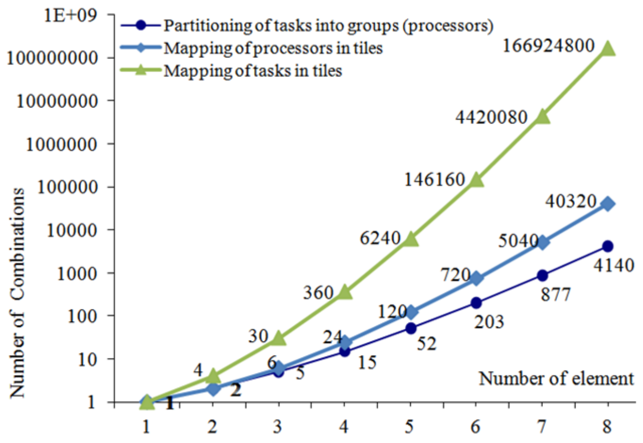


Figure 2 – Number of combinations against the quantity of elements: (i) Partitioning of tasks into groups, (ii) Mapping of task-groups onto tiles, and (iii) Mapping of tasks onto tiles.

The task mapping onto tiles of the target architecture adds the complexities of partitioning tasks across groups and the mapping of these groups onto tiles. In this case, the complexity is much higher *O(Bell(**n**) □ **t**!)*.

Figure 2 shows that the number of solutions to be explored with mapping tasks onto tiles is much higher than others are. Thus, even applying good algorithms, the results obtained with this activity tend to be worse, when compared to those obtained with the flow proposed here, mainly in the cases where it is done at run time, since the mapping has a short time to be accomplished, requiring fast but sometimes inefficient algorithms.

2.2. Structures Definitions

The partitioning and mapping have three main data structures that are set out below.

Definition 1: A *Task Communication Graph* (TCG) is a directed graph $<T, V>$. The set of vertices $T = \{t_1, t_2, \ldots, t_m\}$ represents the set of m tasks in one parallel application. Assuming v_{ab} is the bits amount of all packets sent from a task t_a to a task t_b, then the set of edges V is $\{(t_a, t_b) \mid t_a, t_b \in T \text{ and } v_{ab} \neq 0\}$, and each edge is labeled with the value v_{ab}. V represents all communications between the application tasks.

Definition 2: A *Communication Weighted Graph* (CWG) is a directed graph $<P, W>$, similar to the TCG. However, the set of vertices $P = \{p_1, p_2, \ldots, p_n\}$ represents the set of processors in one application. The quantity of processors n is equal to the total quantity of tiles, since each tile has a single processor. Furthermore, w_{ab} is the total quantity of bits sent from a processor p_a to a processor p_b. Then the set of edges W is $\{(p_a, p_b) \mid p_a, p_b \in P \text{ and } w_{ab} \neq 0\}$, and each edge is labeled with the value w_{ab}. W represents all communications between the MPSoC processors, while CWG reveals information of application's relative communication volume.

The mapping is performed regarding to a 2D mesh NoC using wormhole and deterministic XY routing algorithm. The communication resource graph stated below captures the NoC topology.

Definition 3: A *Communication Resource Graph* (CRG) is a directed graph $<\Gamma, L>$, where the vertex set is the set of tiles $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and the edge set $L = \{(\tau_i, \tau_j), \forall\ \tau_i, \tau_j \in \Gamma\}$ gives the set of paths from τ_i to τ_j. The value n is again the total quantity of tiles and is equal to the product of NoC lines and columns. CRG edges and vertices represent physical links and routers of the target architecture, respectively. The CRG definition is equivalent to the architecture characterization graph in [16] and to the NoC topology graph in [17].

2.3. Energy Model

Both, processors (with the whole memory hierarchy) and communication architecture originate energy consumption.

The sum of the energy consumed by the execution of all tasks grouped on a processor enable estimating its energy consumption. This value is used, together with the communication volume between tasks, to choose good partitions. On the other hand, the amount of bits transmitted between tasks grouped and mapped onto different processors contributes to estimate the energy consumption used to choose good mappings.

The approach used here to model the NoC's energy consumption is similar to those shown in [16] and [18]. Dynamic energy consumption is proportional to switching activity, arising from packets moving across the NoC, consuming energy on the links and inside of each router. The concept of bit energy *EBit* [18] is used to estimate the dynamic energy consumption of each bit, when this flips its polarity from a previous value. *EBit* is split into three components: (i) bit dynamic energy consumed by the router (wires, buffers and logic gates) (*ERbit*); (ii) bit dynamic energy consumed on horizontal (*ELHbit*) and vertical (*ELVbit*) links between tiles; and (iii) bit dynamic energy consumed on the links between the router and the local processor (*ECbit*). Equation (1) expresses the relationship between these quantities, which computes the dynamic energy consumption of a bit passing through a router, a vertical or horizontal link and a local link.

$$(1) \quad EBit = ERbit + (ELHbit \text{ or } ELVbit) + ECbit$$

ERbit depends on the buffer structure and technology to estimate how many bit-flips occur to write, to read and to preserve the information. *ELbit* is directly proportional to the tile dimension. For regular 2D-mesh NoCs with square tiles, it is reasonable to consider that *ELHbit* and *ELVbit* have the same value. Therefore, the next equation uses *ELbit* as a simplified representation of *ELHbit* and *ELVbit*. Equation (2) computes the dynamic energy consumed by a single bit traversing a NoC, from tile τ_i to tile τ_j, where η corresponds to the number of routers through where this bit passes.

$$(2) \quad EBit_{ij} = \eta \times ERbit + (\eta - 1) \times ELbit + 2 \times ECbit$$

Let τ_i and τ_j be the tiles to which p_a and p_b are respectively mapped. Then, the dynamic energy consumed by a $p_a \rightarrow p_b$ communication is given by $EBit_{ab} = w_{ab} \times EBit_{ij}$. Equation (3) gives the total amount of NoC's dynamic energy consumption (*ENoC*) that is computed for all bits of all communications between processors ($|W|$).

$$(3) \quad ENoC = \sum_{i=1}^{|W|} EBit_{ab}(i), \; \forall p_a, p_b \in P$$

2.4. Energy Parameter Extraction and Model Validation

To acquire *ERbit*, *ELbit* and *ECbit* values, an initial estimation was performed according to the characterization of Hermes NoC [19] (2D mesh) on a 70nm CMOS technology, which is the target communication architecture used here. Next, a 2×3 NoC described in electrical level was simulated several times, having synthetic patterns as inputs from the local links, simulating hypothetical applications. The same input patterns were applied to the high-level tool that uses Equation (3) to energy consumption estimation. Then the initial values of *ERbit*, *ELbit* and *ECbit* were refined to minimize the average difference between high-level estimation and the electrical level, which is the reference used here. This process permits to achieve high-level estimation of energy consumption with less than 7% of average deviation.

3. Input Description

A set of XML tags capture the parallel application behavior, which describes some relevant aspects to partitioning and mapping targeting homogeneous MPSoC. These tags concern some target architecture features, some aspects of parallel communication and of each task according to the processor type. The main purpose of using XML is the documentation easiness and data structure sharing across different design tools.

Figure 3 depicts the XML description structure. The MPSOC_SPECIFICATION tag encloses the following tags: PROCESSOR_TYPE, PROCESSOR_TASK_TABLE and TASK_TABLE.

```
<MPSOC_SPECIFICATION>
    <PROCESSOR_TYPE > ... </PROCESSOR_TYPE >
    <PROCESSOR_TASK_TABLE> ... </PROCESSOR_TASK_TABLE>
    <TASK_TABLE> ... </TASK_TABLE>
</MPSOC_SPECIFICATION>
```

Figure 3 – XML structure for parallel application description.

Figure 4 portrays the PROCESSOR_TYPE tag structure, which encloses FEATURES and LIST tags. The first one enables to describe physical characteristics of the processor type, such as dimensions (width and height) in mm and operation frequency in MHz. Processor dimensions permit to estimate the length of the links, and subsequently the energy consumed by these links. The LIST tag contains the name list of all processors. The task partitioning process uses the quantity of processors information to compute the quantity of task groups.

```
<PROCESSOR_TYPE type = "MIPS">
    <FEATURES frequency = "2100" width = "1.5" height = "2.0"/>
    <LIST> P1 P2 P3 </LIST>
</PROCESSOR_TYPE>
```

Figure 4 – Example of basic structure of PROCESSOR_TYPE tag, which contains three 2.1GHz MIPS processor.

Figure 5 depicts the PROCESSOR_TASK_TABLE tag, which contains the application tasks characterization when running on a given processor type. The partitioning uses the average energy consumed (energy) in uJ as a requirement to compose the partitioning cost function and the percentage of processor use (processorUse) in percentage as a constraint to limit the quantity of tasks that may be grouped and mapped onto the same processor.

```
<PROCESSOR_TASK_TABLE>
    <TASK id = "T1" energy = "10.5" processorUse = "25"/>
    <TASK id = "T2" energy = "22.0" processorUse = "10.5"/>
    <TASK id = "T3" energy = "5.0" processorUse = "5.5"/>
</PROCESSOR_TASK_TABLE>
```

Figure 5 – Example of XML structure containing task-processor characterizations.

The TASK_TABLE tag, illustrated in Figure 6, specifies all communications between application tasks. For instance, task T1 sends 160 kB to task T2. This information is used to assemble the TCG described in Section 2.2. This one is the basic data structure used on the task-partitioning algorithm.

```
<TASK_TABLE>
    <SOURCE_TASK source = "T1">
        <COMMUNICATION target = "T2" volume = "160"/>
        <COMMUNICATION target = "T3" volume = "50.5"/>
    </SOURCE_TASK>
    <SOURCE_TASK source = "T2">
        <COMMUNICATION target = "T1" volume = "50"/>
        <COMMUNICATION target = "T3" volume = "30.5"/>
    </SOURCE_TASK>
    <SOURCE_TASK source = "T3">
        <COMMUNICATION target = "T1" volume = "160"/>
    </SOURCE_TASK>
</TASK_TABLE>
```

Figure 6 – Example of XML tag structure that describes all inter-tasks communications.

4. Methodology Description

Figure 7 illustrates the flows described in Section 2, which are implemented inside CAFES [20], a framework for MPSoC design. The task mapping flow is represented by dashed arrows, while the proposed one is symbolized by continuous arrows. In addition, dotted arrows represent input information for mapping and / or partitioning.

Task partitioning into groups has as entries: (i) *application description*, which has all tasks and their communications; (ii) *Processors list*, which has the name and quantity of processors enabling to compute the quantity of task groups; (iii) *Processor use* that is a constraint to limit the quantity of task grouped into the same processor; and (iv) *NoC energy parameters* that is used to compute the energy consumption of a given partition.

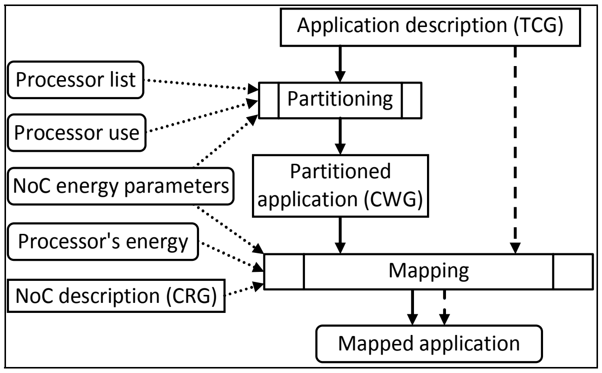


Figure 7 – Design flows showing the partitioning and mapping.

To task partitioning problem, this work applies a stochastic approach with simulated annealing algorithm (SA) [21], which implements a double nested loop. The outer loop tries to find very different partitions aiming to look for global minima. On the other hand, the inner loop explores small partition changes, aiming to find local minima. The algorithm looks for the minimum partitioning cost, which results from the best-searched partition.

The partitioning cost function takes into account the minimization of the overall communication volume. The algorithm tries to achieve a minimum cost, which implies to cluster into the same processor high communicating tasks. In addition, the algorithm tries to balance the processor use through fair distribution of tasks over the available processors, respecting processor use constraint. In other words, tasks that communicate most are grouped as far as they do not compromise more than the maximum processor use for each processor - a parameter set according to the application requirements. The processor use constraint is neglected only in cases where there are no other available processors, i.e. the task association to every processor always implies more than the maximum processor use. The partitioning tool generates a CWG description (Section 2.2) that contains all processor-tasks associations.

The SA algorithm achieves good results for static partitioning problem, since the designer has much time to perform it. On the other hand, dynamic mapping requires fast decisions to not postpone the application execution implying a simpler but efficient algorithm.

This work implements two mapping algorithms: (i) one that has as input the TCG description, which maps the most communicating tasks onto the same processor, while the maximum processor use is not reached. When it happens a new neighbor tile is searched using a mapping cost function; and (ii) the second one that has as input the CWG description. This algorithm searches in the set of task groups for the tile where at least one other task of the same group had been previously mapped. If no previous task was mapped, the algorithm uses the same mapping cost function of the previous algorithm to search the new target tile.

The mapping cost function takes into account the communication volume between processors and the NoC energy parameters to compute the energy consumed on a given mapping. Considering a given pair of communicating processors, together with CRG and NoC parameters, the energy consumption is computed through the energy model described on Section 2.3. The energy consumption achieved by task running on processor is used only to compute the total energy consumption, but does not affect the mapping choice.

For both flows, the mapping generates a file containing all processor-tile associations that implies a minimum energy consumption of all evaluated maps.

Partitioning and mapping cost functions use the same NoC energy parameters stated by Equation (2). However, while mapping specifies the exact processor place into the NoC, the partitioning only explores the communication needs, but the number of hops there is between two communicating processors is unknown. In these sense, partitioning cost function uses the concept of *average of hops* that enables to compute the average energy consumption of all possible paths. Let **X** and **Y** be the number of tiles in horizontal and vertical dimension of a NoC, respectively, than Equation (4) computes the total number of hops of all paths that all processors have regarding to XY routing algorithm.

The *average of hops* is computed dividing the summation of all hops of all paths of all processors by the total number of communications, which is stated by Equations (5)(6) and (7).

(4) $$totalHops = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}\sum_{i=0}^{X-1}\sum_{j=0}^{Y-1}(|x-i|+|y-j|)$$

(5) $$\#Processors = X \times Y$$

(6) $$maxComm = \#Processors \times (\#Processors - 1)$$

(7) $$hopsAverage = \frac{totalHops}{maxComm}$$

The $hopsAverage$ value is used on Equation (2) in place of η, which results an average value of $EBit_{ij}$. This one multiplied by the communication volume is the energy consumption estimation of each communication, which is used during the partitioning.

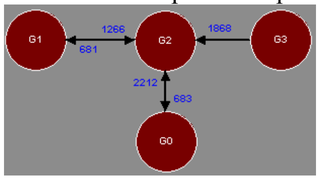
5. Methodology Exemplification

This Section exemplifies the methodology used here. It starts with a partial XML structure (Figure 8) containing: (i) four **PowerPC** processors (**P_0**, **P_1**, **P_2** and **P_3**), each one placed inside a single tile; (ii) the description of a synthetic parallel application composed by 6 tasks (**T_0**, **T_1**, **T_2**, **T_3**, **T_4** and **T_5**); (iii) the tasks characterization on **PowerPC** processor; and (iv) the communications between tasks.

```
<PROCESSOR_TYPE type = "PowerPC">
    <FEATURES frequency = "2100" width = "1.5" height = "2.0"/>
    <LIST> P0 P1 P2 P3 </LIST>
</PROCESSOR_TYPE>
<PROCESSOR_TASK_TABLE>
    <TASK id = "T0" energy = "20.68" processorUse = "63.82"/>
    <TASK id = "T1" energy = "21.53" processorUse = "33.45"/>
    <TASK id = "T2" energy = "36.18" processorUse = "27.13"/>
    <TASK id = "T3" energy = "8.75" processorUse = "69.18"/>
    <TASK id = "T4" energy = "22.59" processorUse = "25.47"/>
    <TASK id = "T5" energy = "16.67" processorUse = "55.96"/>
</PROCESSOR_TASK_TABLE>
<TASK_TABLE>
    <SOURCE_TASK source = "T0">
        <COMMUNICATION target = "T2" volume = "1868"/></SOURCE_TASK>
    <SOURCE_TASK source = "T1">
        <COMMUNICATION target = "T5" volume = "681"/>
        <COMMUNICATION target = "T2" volume = "2183"/></SOURCE_TASK>
    <SOURCE_TASK source = "T2">
        <COMMUNICATION target = "T1" volume = "1516"/></SOURCE_TASK>
    <SOURCE_TASK source = "T3">
        <COMMUNICATION target = "T1" volume = "2212"/></SOURCE_TASK>
    <SOURCE_TASK source = "T4">
        <COMMUNICATION target = "T3" volume = "683"/>
        <COMMUNICATION target = "T2" volume = "1774"/></SOURCE_TASK>
    <SOURCE_TASK source = "T5">
        <COMMUNICATION target = "T4" volume = "1266"/></SOURCE_TASK>
</TASK_TABLE>
```

Figure 8 – Example of a synthetic application description.

Having as input the description of Figure 8, the partitioning tool generated the following task group association: {(**G_0**, **T_3**), (**G_1**, **T_5**), (**G_2**, (**T_1**, **T_2**, **T_4**)), (**G_3**, **T_0**)} and Figure 9 (a) shows a graphical description of the CWG, which is the output description of the partitioning tool.



(a)

Groups	Tasks
G0	T3
G1	T5
G2	T1, T2, T4
G3	T0

(b)

Figure 9 – (a) Graphical CWG description of a synthetic application partitioned into 4 groups; (b) Tasks-group association.

CWG vertices and edges are $P = \{$**G_0**, **G_1**, **G_2**, **G_3**$\}$ and $W = \{$(**G_0**, **G_2**), (**G_2**, **G_0**), (**G_1**, **G_2**), (**G_2**, **G_1**), (**G_3**, **G_2**)$\}$, respectively. The edge labels $w_{G0\_G2} = 2212$, $w_{G2\_G0} = 683$, $w_{G1\_G2} = 1266$, $w_{G2\_G1} = 681$ and $w_{G3\_G2} = 1868$ can be easily extracted from Figure 8 with the task-group partitioning described above.

The mapping tool has as input the above CWG, the topology (CRG) and the energy parameters of the target architecture. Figure 10 depicts the associations ((τ_1, **G_3**), (τ_2, **G_0**), (τ_3, **G_1**), (τ_4, **G_2**)) generated by mapping.

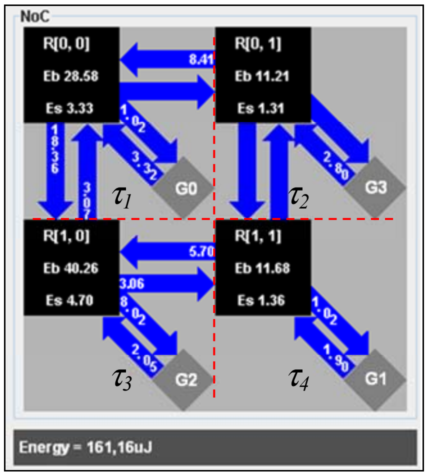


Figure 10 – A processor mapping onto a 2D-mesh NoC architecture with energy consumption annotated inside links and routers. E.g.: inside the router R[0,0], the dynamic energy consumed by the buffers (*Eb*) and switches (*Es*) are 28.58uJ and 3.33uJ respectively; The dynamic energy consumed in link between router and processor P0 are 1.02uJ and 3.32uJ; The dynamic energy consumed by vertical and horizontal links are 8.41uJ, 3.87uJ and 18:36uJ.

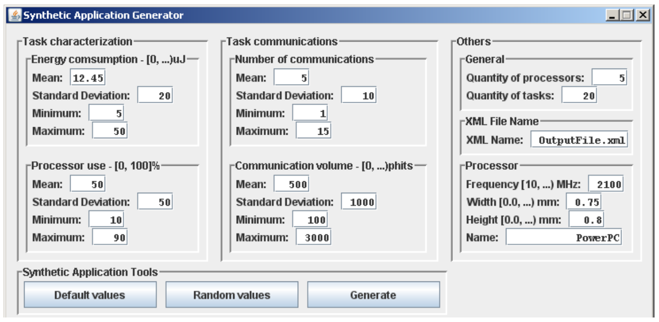
Each link of the communication architecture is associated to an estimated value of the dynamic energy consumption, which depends on the energy parameters and on the quantity of flits, which passes through the communication links. The energy parameters of local links and links between routers are *ECbit* and *ELbit*, respectively. Furthermore, each router contains the energy consumed in buffers (*Eb*) and switches (*Es*). The sum of *Eb* and *Es* is the *ERbit* parameter of Section 2.3. Besides, **Energy** provides an overall estimation of the NoC's energy consumption (161.16 µJ). The energy consumption values are achieved by the characterization of Hermes NoC [19] on a 70nm CMOS technology.

6. Experimental Results

6.1 Synthetic Application Generator

To achieve fair and meaningful results, it is necessary to study a wide range of applications, which is a very time consuming task. Besides, it is hard to find a set of applications, which covers several parallel aspects needed to evaluate MPSoC designs. With this purpose, it was developed a generator of synthetic parallel applications that allows characterizing several application classes according to the input parameters.

Figure 11 shows the interface of synthetic application generator. There are six parameterizable fields inside three set of parameters: (i) *Task characterization*, (ii) *Task communications* and (iii) *Others*.



Synthetic Application Generator

Task characterization

Energy comsumption - [0, ...)uJ
Mean: 12.45
Standard Deviation: 20
Minimum: 5
Maximum: 50

Processor use - [0, 100]%
Mean: 50
Standard Deviation: 50
Minimum: 10
Maximum: 90

Task communications

Number of communications
Mean: 5
Standard Deviation: 10
Minimum: 1
Maximum: 15

Communication volume - [0, ...)phits
Mean: 500
Standard Deviation: 1000
Minimum: 100
Maximum: 3000

Others

General
Quantity of processors: 5
Quantity of tasks: 20

XML File Name
XML Name: OutputFile.xml

Processor
Frequency [10, ...) MHz: 2100
Width [0.0, ...) mm: 0.75
Height [0.0, ...) mm: 0.8
Name: PowerPC

Figure 11 – Interface of synthetic application generator.

The *Task characterization* set contains fields that allow characterizing the tasks of the application when running on the chosen processor: (i) *Energy consumption* is the average of energy consumed by each task; (ii) *Processor use* allows modeling the average consumption of processing time for each task (in percentage).

The *Task communications* set encloses two fields describing application communication aspects: (i) *Number of communication* is the normal distribution of all communications that a task performs during the application execution; (ii) *Communication volume* filed contains the number of phits each communication has. All these first four fields allow generating values according to normal distributions. The designer specifies the main, the standard deviation, the minimum and the maximum values, and the tool generates a random XML description that contains the application built according to the Gauss parameters portrayed in each field.

The *Others* set encloses three fields: (i) *General* that contains the fields *Quantity of processors* and *Quantity of tasks*, which express the exact quantities of processors and of tasks that the application contains, respectively; (ii) *XML File Name*, which is the output file name, where the synthetic application will be stored; and (iii) *Processor*, containing operating frequency, width, height, and name of the processor type.

The *Synthetic Application Tools* encloses three buttons: (i) *Default values* that completes all fields with default values; (ii) *Random values* that completes all fields, except the XML File Name, with values randomly generated; and (iii) *Generate* that generates a XML file containing a synthetic application according to the specified parameters.

6.2 Results

All experiments were designed in order to evaluate the effect of using static partitioning as a previous step of the dynamic mapping on energy savings, which is the approach proposed here. All tabulated values show, in percentage, how much the use of a static partitioning before dynamic mapping allows minimizing the energy consumption.

Table 1 summarizes the first set of experiments, which is composed by synthetic applications with 50 tasks, *processor use* varying randomly from 10% to 80%, 16 processors, and a set of *number of communications* (10%, 20%, 40%, 60%, 80% and 100%) combined with a set of *communication volume* (1, 10, 100, 1000 and 10000). The *number of communications* is expressed in percentages - 100% means that all tasks communicate with all other tasks, 0% is the absence of communications and intermediate values are linearly computed.

Table 1 - Results of energy consumption minimization for 30 synthetic applications with variable number of communications between tasks and quantity of bits of each communication.

Energy savings (%)		Number of communications (%)						
		10	20	40	60	80	100	ACV
Communication volume	1	19.7	18.4	16.4	16.0	15.7	8.8	**15.8**
	10	19.1	17.3	14.5	15.4	14.8	14.2	**15.9**
	100	18.2	15.9	16.4	14.3	11.7	15.7	**15.4**
	1000	20.1	15.7	15.2	13.2	13.5	13.0	**15.1**
	10000	17.1	19.6	15.9	13.5	13.1	12.5	**15.3**
ANC		**18.8**	**17.4**	**15.7**	**14.5**	**13.8**	**12.8**	15.5

Legend: ACV - Average of communication volume
ANC - Average of number of communications

The column ACV shows how much the number of bits transmitted between tasks affects on energy consumption minimization. It is a fact that the increase of communication volume raises the energy consumed by the target architecture. However, Table 1 shows that the average of energy savings is similar for all *communication volumes*. It means that this feature does not affect the efficiency of the approaches evaluated here, and the total average of all minimizations of energy consumption is 15.5%.

The line ANC shows the effect of *number of communications* between tasks variation on energy savings. The results achieved with *number of communications* variation are similar to the ones achieved with the *communication volume* variation. Nevertheless, our approach is more efficient on energy savings for a small number of communications, because there are more scenarios that allow approximating the heavily communicating tasks.

Table 2 - Results of energy consumption minimization for 36 synthetic applications achieved by the combination of 6 quantities of tasks and 6 sizes of NoCs.

Energy savings (%)		Quantity of tasks						
		10	20	40	60	80	100	ANS
NoC sizes	2 x 3	2.4	6.5	10.9	16.6	23.8	48.3	**18.1**
	3 x 3	3.5	7.7	11.2	19.4	34.6	68.8	**24.2**
	3 x 4	7.7	12.3	18.3	23.4	35.9	73.6	**28.5**
	4 x 4	9.3	15.3	19.3	20.1	39.9	79.7	**30.6**
	4 x 5	12.5	18.6	23.1	28.9	45.6	89.0	**36.3**
	5 x 5	15.0	25.5	26.8	31.2	49.6	93.1	**40.2**
AQT		**8.4**	**14.3**	**18.3**	**23.3**	**38.2**	**75.4**	29.7

Legend: ANS - Average of NoC sizes
AQT - Average of quantity of tasks

Table 2 shows the results of the second set of experiments. This set is composed by synthetic applications, where, for all applications, any task communicates with 40% of the remaining tasks of the application, each communication between tasks has 100 phits (phit is 16 bit length) and the *processor use* varies randomly between 10% and 80%. Combining six *quantities of tasks* (10, 20, 30, 40, 50 and 100) with six *NoC sizes* (2×3, 3×3, 3×4, 4×4, 4×5 and 5×5) totalizes 36 synthetic applications. Some combinations have empty tiles - not populated by tasks (e.g. a 3×4 NoC with 10 tasks implies two empty tiles), and other combinations will have more than a task by tile (e.g. a 3×3 NoC with 10 tasks implies that at least one tile have more than one task). Since each tile contains a processor, there are six quantities of processors: 6, 9, 12, 16, 20 and 25, for each size of NoC, respectively.

The energy consumption minimization achieved with our approach is not meaningful when applied to applications with small quantity of tasks running on small NoCs. On the other hand, for large applications or large NoCs the efficiency of our approach is evident, since it deals better with these complexities. Column ANS illustrates the effect of *NoC size* variation on energy savings, where it is clear the increase of energy savings provided by our approach, when the NoC's complexity increases.

Line AQT illustrates the effect of *quantity of tasks* variation on energy consumption minimization. It implies the most important part of the energy savings in relation to the experiments conducted here, which is justified by the increase of task grouping possibilities when the quantity of tasks increases and these possibilities are well captured by the static partitioning step of our approach.

Four embedded applications comprise the last set of experiments: (i) a digital PBX (privative branch exchange); (ii) an image recognition system (IRS); (iii) a distributed algorithm for Romberg's integral calculus; and (iv) a multimedia system (MMS). Table 3 depicts some relevant features of these applications and Table 4 shows the values of energy consumption minimization for these applications.

Table 3 - Characteristics of the four embedded applications.

Application features	PBX	IRS	Romberg	MMS
NoC size (lines x columns)	2 x 3	2 x 3	3 x 4	4 x 4
Quantity of processors	5	6	10	16
Quantity of tasks	24	12	30	34
Number of communications	142	53	60	182
Average communication volume (bytes)	2,334	30,827	35	22,135

Table 4 - Results of energy consumption minimization for 4 embedded applications.

Energy savings	Embedded applications				Average
	PBX	IRS	Romberg	MMS	
Difference (%)	29.1	9.8	51.0	22.8	28.2

Similarly to the results acquired with synthetic applications, the efficiency of our approach is achieved in applications with more quantity of tasks mapped on more complex NoC, which are the cases of Romberg and MMS. In addition, the Romberg and PBX applications are mapped into NoCs with more tiles than the quantity of processors requires, providing more places to map groups of tasks. Our approach explores better this feature allowing searching for better results. Moreover, as stated above, the communication volume does not influences on the energy savings, as much that the results of Romberg application shows much more energy consumption minimization than those achieved in IRS and MMS applications.

7. Conclusions

The task mapping onto tiles of the target architecture is an NP-complete design activity, and when performed at runtime may not get good results, due to the exiguous time and to the large number of solutions to be explored. This work proposes to apply the partitioning of tasks into groups before the mapping. Once tasks were grouped, the search space of the mapping is minimized, which allows building efficient dynamic mapping algorithm that may performs an ideal task mapping in a short period of time. As a consequence, more than one application requirement may be better fulfilled. This work used several synthetic and four embedded applications to show that when this approach is well conducted may lead a significant energy savings.

8. Acknowledgment

The Authors acknowledge the support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq-Brazil) through research grants 308924/2008-8 and 306178/2009-5.

9. References

[1] Jalier, C. et al. **Heterogeneous vs. homogeneous MPSoC approaches for a Mobile LTE modem**. *DATE*, pp.184-189, 2010.

[2] Bononi, L. et al. **NoC Topologies Exploration based on Mapping and Simulation Models.** *DSD*, pp.543-546, 2007.

[3] Chen-Ling, C.; Marculescu, R.; **Contention-aware application mapping for Network-on-Chip communication architectures**. *ICCD*, pp.164-169, 2008.

[4] Le Beux, S. et al. **Combining mapping and partitioning exploration for NoC-based embedded systems**. *JSA*, v.56(7), pp.223-232, 2010.

[5] Sahu, P. et al. **A new application mapping algorithm for mesh based Network-on-Chip design**. *INDICON*, pp.1-4, 2010.

[6] Bo Yang et al. **Multi-application multi-step mapping method for many-core Network-on-Chips**. *NORCHIP*, pp.1-6, 2010.

[7] Carvalho, E.; Calazans, N.; Moraes, F. **Dynamic Task Mapping for MPSoCs**. *IEEE Design & Test*, v.27(5), pp. 26-35, 2010.

[8] Leupers, R.; Castrillon, J.; **MPSoC programming using the MAPS compiler**. *ASP-DAC*, pp.897-902, 2010.

[9] Guang, S. et al. **Energy-aware run-time mapping for homogeneous NoC**. *SoC*, pp.8-11, 2010.

[10] Nedjah, N.; Silva, M.; Mourelle, L. **Customized computer-aided application mapping on NoC infrastructure using multi-objective optimization**. *JSA*, v.57(1), pp.79-94, 2011.

[11] Tsai, K. et. al. **Design of low latency on-chip communication based on hybrid NoC architecture**. *NEWCAS*, pp.257-260, 2010.

[12] Youness, H. et al. **A high performance algorithm for scheduling and hardware-software partitioning on MPSoCs**, *DTIS*, pp.71-76, 2009.

[13] Go□hringer, D. et al. **A Design Methodology for Application Partitioning and Architecture Development of Reconfigurable Multiprocessor Systems-on-Chip**. *FCCM*, pp.259-262, 2010.

[14] Sherwani, N. A. **Algorithms for VLSI Physical Design Automation**. *Kluwer Academic Publisher*, 1999.

[15] Zwillinger, D. **Standard Mathematical Tables and Formulae**, *CRC Press*, 1996.

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

[17] Murali, S.; De Micheli, G. **Bandwidth-constrained mapping of cores onto NoC architectures**. *DATE*, pp.896-901, 2004.

[18] Ye, T.; Benini, L.; De Micheli, G. **Analysis of power consumption on switch fabrics in network routers**. *DAC*, pp.524-529, 2002.

[19] Moraes, F et al. **HERMES: an infrastructure for low area overhead packet-switching networks on chip**. *Integration, the VLSI Journal*, v.38(1), pp.69-93, 2004.

[20] Marcon, C. et al. **CAFES: A framework for intrachip application modeling and communication architecture design.** *JPDC*, v.71(5), pp.714-728, 2011.

[21] Kirkpatrick, S.; Gelatt, C.; Vecchi, M. **Optimization by simulated annealing**, *Science*, pp.671-680, 1983.