2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems

Pre-Mapping Algorithm for Heterogeneous MPSoCs

César Marcon, Thais Webber, Letícia B. Poehls, Igor K. Pinotti

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga 6681, Porto Alegre, Brazil – 90619-900
cesar.marcon@pucrs.br

Abstract—**A Multiprocessor System-on-Chip (MPSoC) composed of different types of processors is known as heterogeneous MPSoC. This kind of MPSoC based on Network-on-Chip (NoC) is a promising target architecture to fulfill requirements of high processing and communicating rate, enabling the execution of several tasks at the same time. Among the challenges in current heterogeneous MPSoC design is the partitioning of application tasks aiming for the energy saving and for a fair load balance. This work's contribution is twofold: (i) a partitioning algorithm is developed; and (ii) the evaluation of using partitioning as a pre-mapping task is explored. This work analyzes and compares stochastic and new heuristic partitioning algorithms for obtaining low energy consumption and for load balance when applied to tasks partitioning onto heterogeneous MPSoC. In addition, simulated results indicate that the static partitioning technique could be applied to application tasks before mapping activities in order to improve the quality of the static or dynamic mapping design and also to minimize the processing time.**

Keywords—partitioning; mapping; heterogeneous MPSoC.

I. Introduction

Networks-on-Chip (NoCs) are an efficient communication infrastructure for Multiprocessor System-on-Chip (MPSoC) architectures. A NoC is typically composed of a set of routers interconnected by communication channels. In direct NoC topologies, each router connects to a module and both are placed inside a limited region of an integrated circuit called tile. Low energy consumption, performance, scalability, modularity, and communication parallelism, make NoCs powerful communication architectures for MPSoCs [1].

Ogras et al. [2] proposed three dimensions for NoC architectural design: (i) communication architecture synthesis, (ii) communication paradigm selection, and (iii) application partitioning/mapping optimization. This paper focuses on the application-partitioning problem that consists in finding associations of tasks into groups according to criteria.

The partitioning of ***k*** application-tasks in mutually exclusive groups of tasks generates massive possible solutions, and the task groups mapping onto ***n*** processors in the NoC can generate ***n***! possible solutions. Considering a SoC containing hundreds of tiles, the exhaustive search solution is unfeasible. Thus, the development of efficient partitioning and mapping approaches is able to guarantee better SoC implementations.

In contrast of dynamic partitioning (and/or mapping) that occurs at runtime, static partitioning (and/or mapping) occurs at design time, which is suitable for static workloads. Thus, at design time the adequate placement of a given task may be found according to its computation and/or communication.

In this paper we provide a comparative analysis of Simulated Annealing (SA) [3] and Tabu Search (TS) [3] with two new heuristic algorithms based on the classical Kernighan & Lin (KL) algorithm [4] that are called KL*-width and KL*-height. While SA and TS are classical stochastic and well-known algorithms applied in several works for general-purpose problems, KL is a bisection algorithm known for minimizing the weight or the number of edges in a given graph. This work proposes two new algorithms, KL*-width and KL*-height, both, adaptations of versions of KL algorithm to the context of heterogeneous MPSoC design (i.e. composed of different types of Processing Elements (PEs)), aiming to achieve fair load balanced partitions while saving energy.

Direct task mapping into processors of the target architecture has been compared with the approach that previously applies static partitioning of tasks into a group of tasks (pre-mapping), and then maps these groups onto the processors. Thus, the application task graph is taken as input for the pre-mapping procedure, which tries to minimize the communication volume among various tasks, also aiming at a reduction of the energy consumption. Simultaneously, the procedure tries to balance the processing load on various groups related to the different types of processors of a heterogeneous NoC-based MPSoC. The pre-mapping is proposed focusing on the improvement of the dynamic mapping quality by previously sorting each application task in determined groups, thereby optimizing the mapping process.

This paper is organized as follows. Section II contains a discussion about related works; Section III presents the problem formulation of task partitioning (pre-mapping) and task/group mapping. Section IV describes the KL modified version proposed for tasks pre-mapping on heterogeneous MPSoCs. Section V presents experimental results. Finally, Section VI presents the conclusion and further discussions.

II. Related Works

Partitioning and mapping represent major research challenges that have been addressed, in several works over the last decades [17]. Table I summarizes related work, classifying them according to: (i) design tasks (static/dynamic partitioning and/or mapping); (ii) partitioning and/or mapping algorithms; (iii) design requirements (e.g. energy saving and total execution time minimization); and (iv) target architecture - MPSoC type (homogeneous/heterogeneous) and NoC type. The main contribution of the work is twofold: (i) the use of a pre-processing (partitioning) before mapping, focusing on improving the mapping quality. Our paper follows the same idea as [11] and [12]; however, we investigate the pre-processing of tasks when applied to heterogeneous architectures; and (ii) the proposal of two algorithms based on KL, which are used to fulfill the proposed strategy.

TABLE I - RELATED WORK SUMMARY.

Work, year	Design tasks	Algorithm	Design requirements	MPSoC type, NoC
[6], 2006	Static mapping	Greedy	Energy and area minimization	Homogeneous, regular mesh/torus
[7], 2008	Static partitioning, dynamic mapping	Selection / incremental	Energy and time saving	Homogeneous, regular mesh
[8], 2008	Static mapping	Stochastic and greedy	Energy and time saving	Homogeneous, regular mesh
[9], 2009	Dynamic mapping	Heuristic search space	Communication overhead minimization	Heterogeneous, regular mesh
[10], 2011	Static mapping	Multicast aware heuristic	Energy and end-to-end delay reduction	Homogeneous, regular mesh
[11], 2011	Static partitioning, dynamic mapping	Heuristic	Load balance and traffic reduction	Homogeneous, regular mesh
[12], 2011	Static partitioning, static mapping	SA	Energy saving	Homogeneous, regular mesh
[13], 2012	Static mapping	ILP	Energy saving	Homogeneous, regular/irregular mesh
[14], 2013	Static mapping	Heuristic	Total execution time minimization	Heterogeneous, arbitrary
This work	Static partitioning, static/dyn. mapping	KL based algorithms	Load balance and energy saving	Heterogeneous, regular mesh

III. PROBLEM FORMULATION

This paper assumes that an application is a group of tasks and NoC-based MPSoC with heterogeneous processors as target architecture. A NoC is a 2D mesh topology composed of tiles; and each tile τ contains a router r and a processor p. The deterministic XY routing algorithm is employed to route packets only along minimal paths. Moreover, the design of parallel applications running on heterogeneous MPSoCs involves partitioning and mapping activities, which depend on the application and target architecture characteristics.

A *partition* is a division of a set of elements into non-overlapping and non-empty blocks that cover the entire set. This work considers elements as tasks and blocks as groups of tasks. Further, *task partitioning* is defined as the activity of finding a partition that minimizes energy consumption whereas performing load balancing among groups of tasks – the partitioning cost function. Nevertheless, when concerning heterogeneous processors, each task grouping has to compute the cost function according to the available processors features. Additionally, a *map* is a function that associates source elements of the source set to target elements of the target set. This work takes into account two types of mappings: (i) *task mapping* and (ii) *task-group mapping*. Both mapping types consider processors of the target architecture as target elements, but while for *task mapping* the source elements are individual tasks of the application, for *task-group mapping* the source elements are groups of tasks provided by partitioning.

The mapping, as well as the partitioning, can be applied in static (i.e. design time) or dynamic (i.e. runtime) scenarios. The main advantage of the static approach is the independence from time to explore solutions. The dynamic approach is capable of handling unpredictable behaviors, while static algorithms can explore more adequate solutions for a set of well-known possibilities. Depending on the application's complexity, it is unfeasible to foresee all possible scenarios at design time. However, some pre-analysis of applications characteristics can be made, generating useful information for further complexity minimization at runtime.

This work focuses on static partitioning as pre-mapping in order to reduce mapping complexity. Static evaluation over tasks connections can be performed, thus enabling faster mapping decisions. In other words, if the tasks have been grouped together according to their connection's weight, the mapping algorithms have to search only inside the connection's set, instead of a wider search inside all tasks. This mapping activity can be performed statically or dynamically done, but the advantages are highlighted when the mapping is performed at runtime.

Fig. 1 illustrates the mapping process of an application composed by communicating tasks, which considers application requirements and constraints (e.g. energy consumption and load balance, connected by dotted arrows), and a given NoC-based MPSoC architecture (e.g. NoC topology, number of processors, type of processors). The process involves pre-mapping and mapping (continuous path), or applies the mapping directly (dashed path).

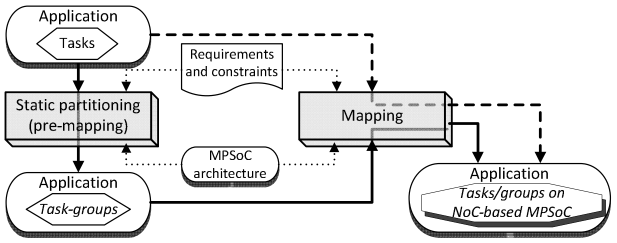
Application
Tasks
Requirements and constraints
Static partitioning (pre-mapping)
Mapping
MPSoC architecture
Application
Task-groups
Application
Tasks/groups on NoC-based MPSoC

Fig. 1 Application pre-mapping/mapping targeting to NoC-based MPSoC.

A. Application and MPSoC Definitions

Definition 1: A TCG (*Task Communication Graph*) is a directed graph $<T, S>$, where $T = \{t_1, t_2, ..., t_n\}$ represents the set of n tasks in a parallel application, i.e. the set of TCG vertices. Assuming s_{ab} is the quantity of bits sent from task t_a to task t_b, hence the set of edges S is $\{(t_a, t_b, s_{ab}) \mid t_a, t_b \in T, s_{ab} \neq 0\}$, with each edge attached to its s_{ab} value, is the total communication amount between tasks of an application.

Definition 2: A CWG (*Communication Weighted Graph*) is a directed graph $<G, W>$, similar to TCG, but $G = \{g_1, g_2, ..., g_n\}$ represents the set of n task-groups achieved from pre-mapping, which has TCG as input. Furthermore, w_{ab} is the total communication amount (in bits) transmitted from task-group g_a to task-group g_b. The set of edges W is $\{(g_a, g_b, w_{ab}) \mid g_a, g_b \in G, w_{ab} \neq 0\}$ with each edge attached to its w_{ab} value, representing all the communication between MPSoC processors, since the *task-group mapping* assigns each task-group to a unique processor. CWG reveals the relative communication volume of an application.

Definition 3: A CRG (*Communication Resource Graph*) is a directed graph $<\Gamma, L>$, where $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ is the set of n tiles (i.e. the set of CRG vertices), each tile containing a processor of $P = \{p_1, p_2, ..., p_n\}$. $L = \{(\tau_i, \tau_j), \forall \tau_i, \tau_j \in \Gamma\}$ is the set of CRG edges, i.e., the set of routing paths from tile τ_i to tile τ_j. The CRG vertices and edges represent, respectively, the routers $R = \{r_1, r_2, ..., r_n\}$ and their physical links.

Fig. 2 illustrates an example of pre-mapping and mapping of a synthetic application. Fig. 2(a) shows a TCG, which is the pre-mapping input model, where $T = \{t_1, ..., t_8\}$, and $S = \{(t_1, t_5, 6), (t_1, t_7, 8), (t_2, t_1, 4), ...\}$. Fig. 2(b) shows the pre-mapping output containing a CWG with $G = \{g_1, …, g_4\}$ and $W = \{(g_2, g_3, 30), (g_3, g_4, 5), …\}$. The CWG and CRG are inputs for mapping, whose output is shown in Fig. 2(c), where $T = \{\tau_1, ..., \tau_4\}$ and $L = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_4), …\}$. Each tile τ_i contains a processor p_i of a given type, and the mapping produces the association $\{(p_1, g_1), (p_2, g_4), (p_3, g_2), (p_4, g_3)\}$.

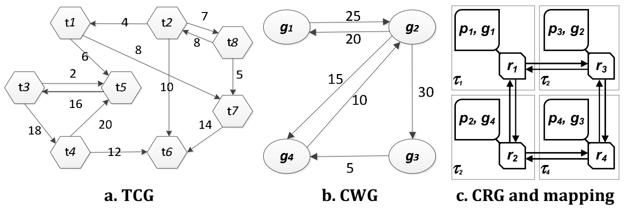

Fig. 2 Example of application and NoC descriptions

B. *MPSoC Energy Consumption Model and Algorithm*

Processors and communication infrastructure have an important influence on the energy consumption of an application mapped onto an MPSoC. The sum of the energy consumed in a specific processor during the execution of all tasks, enables to estimate the individual processor energy consumption. In a heterogeneous scenario, the energy consumed in each processor, and by a given task, could vary according to the architecture and to the optimized procedures considering the processor type. The quantity of bits exchanged between processors, enables to compute the total energy consumption related to the communication architecture. The energy consumed by tasks running on processors, plus the energy consumed on the communication architecture, determines the choice of partitions or mappings.

The energy consumption model applied in this work is similar to [8]. The dynamic energy consumption is related to the package exchange through the NoC, dissipating energy inside each router and on the links where the package passes by. E_{bit} is an estimation of the dynamic energy consumption for each bit, when the bit changes its value (i.e. polarity). E_{bit} is divided in three components: (i) ER_{bit} – dynamic energy consumed on the router components (e.g. wires, buffers and logic gates); (ii) ELH_{bit} and ELV_{bit} – dynamic energy consumed on horizontal and vertical links between tiles, respectively; and (iii) EC_{bit} – dynamic energy consumed on links between each router and its local processor. For regular 2D mesh NoCs with square dimension tiles, it is reasonable to estimate that ELH_{bit} and ELV_{bit} have the same value. Hence, we assume EL_{bit} as a simplified way to represent ELH_{bit} and ELV_{bit}. Eq. (1) computes the dynamic energy consumed by a bit passing through the NoC from tile τ_i to tile τ_j, with η being the number of routers that the bit passes through, i.e. the number of hops.

$$E_{bitij} = \eta \times ER_{bit} + (\eta - 1) \times EL_{bit} + 2 \times EC_{bit} \tag{1}$$

Both mapping and partitioning cost functions use the NoC energy model parameters stated by Eq. (1); however mapping provides the association of a task or task-group to a processor placed in specific tile, whereas partitioning only explores the communication needs without knowing each processor position (i.e. the number of hops between two communicating processors is unknown). Thus, partitioning cost function uses the *average of hops* concept, which allows computing the average energy consumption of all possible paths.

Assuming that both X and Y are the number of tiles in horizontal and vertical dimensions of a NoC, respectively. Therefore Eq. (5) computes the total number of hops of 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 Eq. (2), (3), (4) and (5).

$$Hops_{total} = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}\sum_{i=0}^{X-1}\sum_{j=0}^{Y-1}(|x-i| + |y-j|) \tag{2}$$

$$Num_{Processors} = X \times Y \tag{3}$$

$$Max_{Comm} = Num_{Processor} \times (Num_{Processors} - 1) \tag{4}$$

$$\bar{\eta} = (Hops\_Total) \,/\, (Max\_Comm) \tag{5}$$

The value $\bar{\eta}$ is applied to Eq. (1) replacing the value η, resulting on an average value of E_{Bitij}. Thus, the energy consumption's estimation, of each communication, used during the partitioning (i.e. pre-mapping), is the result of a multiplication of E_{Bitij} by the communication volume.

Being τ_i and τ_j the tiles that contain p_a and p_b, respectively, the dynamic energy consumed by all communications traffic $p_a \rightarrow p_b$ is given by Eq. (6). While, the total amount of NoC energy consumption (*ENoC*) related to all communication traffic between processors (|W|) is given by the Eq. (7).

$$E_{Bitab} = W_{ab} \times E_{bitij} \tag{6}$$

$$ENoC = \sum_{i=1}^{|W|} E_{Bitab}(i), \ \forall p_a, p_b \in P \tag{7}$$

The *ENoC* algorithm is implemented with four nested loops, whose pseudo-code is shown in Fig. 3. The outer loop (lines 2 to 11) searches for all processor-tasks associations finding the source processor in an inter-processor communication. The inner loop (lines 3 to 10) searches for all processor-task's associations, finding the target processor. The algorithm returns an association, instead of a map, since it is applied for performing both, partitioning and mapping.

```
1.  cost ← 0;
2.  for(Association sa: associations) {
3.    for(Association ta: associations) {
4.      if(sa.equals(ta))
5.        continue;
6.      for(Task st: sa.getTask()) {
7.        for(Task tt: at.getTask())
8.          cost ← cost + st.computeEnergy(tt);
9.      }
10.   }
11. }
```

Fig. 3 Pseudo-code of ENoC algorithm

The energy consumed by the inter-processor communication is computed by a function denominated *computeEnergy* inside two nested loops (from line 6 to 9). The function implements the Eq. (6) and the *ENoC* of Eq. (7) is the final value stored in the variable *cost*, which is the return value of the function *actualCost()* of the algorithms in Section IV.

IV. KL* Partitioning Algorithm

Kernighan & Lin (KL) [4] proposed a graph bisection algorithm, starting with a random initial partition, using pairwise swapping of vertices among partitions to reduce the cut size until no further improvement is possible. The classical KL starts partitioning the graph into two subsets of equal sizes. Pairs of vertices are exchanged across the bisection if the exchange improves the cut size. The procedure is carried out iteratively until no further improvement can be achieved.

The method is particularly well suited for bisection, since it divides the graph into two parts, but can be abstracted to perform partitioning into unequal parts, becoming the basis of a hierarchical partitioning scheme. However, KL does not totally comply with the application's partitioning problem as stated above, because it has to associate a task or a group of tasks to a given processor type (i.e. when dealing with heterogeneous MPSoCs). In addition, groups of tasks are limited to the quantity of processors, and the cost of each task-group associated to a processor depends on the processor type. Thus, we implemented a modified KL algorithm (KL*), based on the classical KL idea but having as input a list of individual tasks or groups of tasks to execute partitioning as well as a list of processors with their characteristics (quantity and type). The output is a list containing task groups associated to types of processors. Fig. 4 shows the pseudo-code of KL implemented according to depth and width bisection.

```
1.   interaction ← Interaction input parameter
2.   while(interaction > 0) {
3.     interaction--
4.     while(!reachConstraints() || !reachBipartitionLimit()) {
5.       if(isDepthBipartition)
6.         depthBipartition()
7.       else
8.         widthBipartition()
9.     }
10.    if(minimumPartitionCost > actualPartitionCost) {
11.      minimumPartitionCost = actualPartitionCost
12.      saveActualPartitionAsMinimum()
13.    }
14.  }
```

Fig. 4 Pseudo-code of KL* algorithm.

The outer loop of KL* (lines 2 to 14) is responsible for enlarging the search space, since each loop starts new initial random partition, which is controlled by the parameter *interaction*. The inner loop (lines 4 to 9) accomplishes the depth partition by algorithm *depthBipartition* (KL*depth), or the width partition through *widthBipartition* (KL*width), according to the input Boolean parameter *isDepthBipartition*. Fig. 5 exemplifies the process of both algorithms.

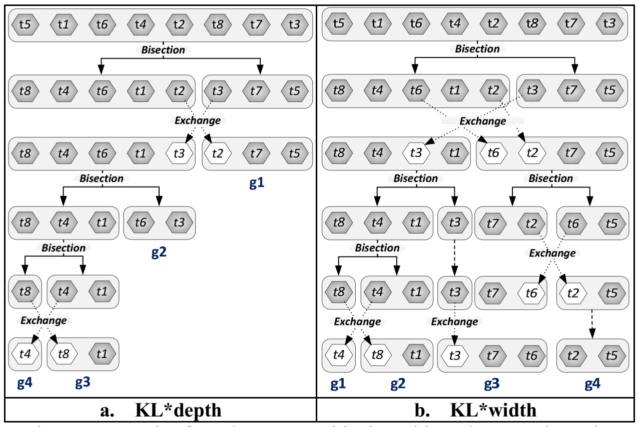

Fig. 5 Example of a task-group partitioning with KL* approaches. The application contains 8 tasks, targeting an MPSoC with at least 4 processors.

The KL*depth algorithm starts performing a bisection, trying to find a target task-group that fulfills the constraints. Thus, the system is divided into 2 groups, the target group that the algorithm is trying to optimize, and the one that contains the remaining tasks. Hence, the algorithm performs several task exchanges between task-groups, aiming to minimize the cost function of the target task-group; and the quantity of exchanges is defined by an input parameter. The tasks exchange may be performed in two different ways: (i) a pair of tasks is exchanged between a pair of groups; or (ii) a task migrates from a group to another. When exchange is performed, the target task-group remains unchanged until the algorithm does not reach the end of the inner loop. If the inner loop condition (line 4) is satisfied, the algorithm restarts a new sequence of bisection plus several tasks exchanges, but now taking as input the remaining task-group provided by the last bipartition. As the remaining task group is always smaller than the former, each iteration has to handle a less complex problem, which is performed in less time.

In comparison, the KL*width starts performing a bisection trying to find task-groups that equally fulfill the constraints. Thus, all groups have the same optimization priority. The algorithm performs exchanges of tasks between task-groups aiming to minimize the cost function of all task-groups and the quantity of exchanges is previously defined by an input parameter, which is typically lesser than the one used on KL*depth. The task exchanges may be performed exactly the same way they would be in the KL*depth. Having performed the exchange, the inner loop condition (line 4) is verified; if satisfied, KL*width restarts applying bisections to all groups, until reaching all processors of the target architecture. Also, the exchanges occur among all task-groups. As a consequence, all iterations are similarly complex and time consuming.

Independent of the bipartition approach, the inner loop stops when all constraints (e.g. limit of processor workload or maximum energy consumption) are satisfied by all associations of task-groups with the corresponding processor type, or when the bipartition reaches the limit (i.e. cannot perform more bisections, since there are no more processors to associate a new task-group with and at least one round of exchanges was already performed). These verifications are accomplished by the functions *reachConstraints()* and *reachBipartitionLimit()*, respectively. Finally, every time the inner loop finishes, the cost of the achieved partition is stored to be compared to all costs, making possible the identification of the best of all reached partitions.

V. Experimental Results

This section explores two types of experiments: (a) the influence of pre-mapping plus mapping (PM) when compared to a direct mapping approach (DM), both targeting a heterogeneous MPSoC; and (b) the evaluation of a set of static partitioning algorithms used to perform the pre-mapping.

A. Pre-Mapping and Mapping versus Direct Mapping

DM has a TCG as input, which maps tasks that communicate the most with the same processor, whereas satisfying the processor constraints. When some processor's constraint is reached, the search for a neighboring processor is initiated. The heterogeneous approach expands the algorithm implemented in [16] by also taking into account the processor type and its influence on the mapping cost function. The PM has a CWG as input after the partitioning, which searches the set of task-groups for the processor that other tasks of the same group have been already mapped onto. If no task was previously mapped, PM searches for a near processor of the same type, elected by the static partitioning algorithm.

Experiments evaluate how PM saves energy and improves load balancing if compared to DM. They are composed of synthetic applications, where tasks have on average 15% of inter-tasks communication (e.g. for an application with 20 tasks, each task communicates with 3 others), each phit (physical link) is 16-bits length, and each inter-task communication has 100 phits. All MPSoCs are composed of 3 processors type, with different performance and energy consumptions, and whose quantities are proportional to NoC size; their positions are randomly distributed into the tiles of the target architecture. Besides, applications consider that any task could be executed on any type of processor. Thus, combining 6 *quantities of tasks* (25, 50, 75, 100, 125 and 150) with 4 *NoC sizes* (3×3, 4×4, 5×5 and 7×7) totalize 24 synthetic applications. These experiments are used as input for both flows of Fig. 1, regarding energy consumption and load balancing. For each cost function, the results are compared proportionally, and the percentages of improvement when using PM instead of using DM are illustrated in Fig. 6.

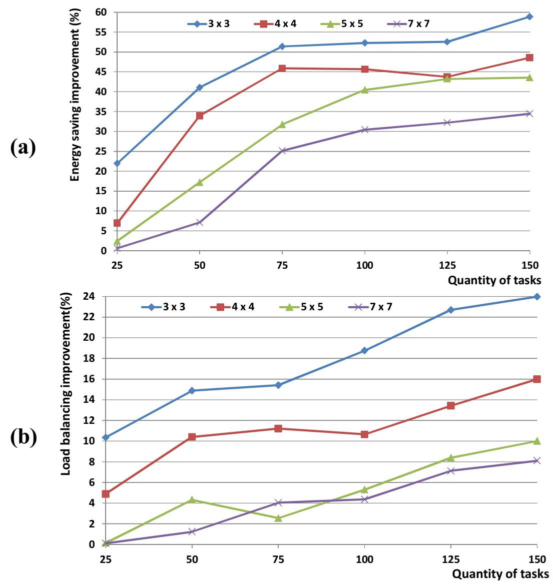

Fig. 6 Improvements when using PM instead of DM approach; (a) energy saving, (b) load balancing

The results show that PM always enables to achieve better results than DM, because PM may capture some information during the design time, where constraints still allow exploring implementation options. Also, this static information enables to reduce the search space for solutions during runtime (i.e. the quantity of task-groups used in the mapping is always smaller than the quantity of tasks used in DM), which is an advantage for NP-complete problems. The increased quantity of grouped tasks improves the quality of the results obtained with PM in comparison with DM results, because it increases the ratio of the number of tasks by the quantity of task-groups. Further, it is worth noting that the improvement when using PM is directly proportional to the quantity of processors type, since even for small task-groups, the decision of the target processor type may be statically made by PM, whereas it is dynamically performed in DM. Finally, although not explicitly shown in Fig. 6, PM improves the load balancing and energy saving in 9.5% and almost 34%, respectively, when compared to DM.

B. *Analysis of Pre-Mapping Activity*

We evaluate SA, TS and KL* static partitioning algorithms in relation to energy consumption minimization and load balancing performance. As illustrated in Fig. 7, the flow used to evaluate algorithms for pre-mapping presents as input the application composed of communicating tasks, the MPSoC characteristics (e.g. type and number of processors), the requirements and constraints of each processor type.

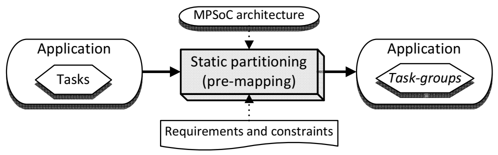
MPSoC architecture
Application
Tasks
Static partitioning (pre-mapping)
Application
Task-groups
Requirements and constraints

Fig. 7 Evaluation flow for algorithms targeted for pre-mapping activity

The synthetic applications are similar to those employed in Section V.A, but exploring the tasks connectivity. The MPSoC has the following features: (i) a 3x3 mesh NoC; (ii) 16-bit length for each phit; (iii) three types of processors; (iv) the MPSoC contains exactly three processors of each type, totalizing 9 processors; (v) the processors' positions in the NoC tiles are randomly chosen.

The applications have the following features: (i) 6 *quantities of tasks* (25,50,75,100,125 and 150) for each set of experiments; (ii) TCG generated considering 5 task connectivity (10%, 15%, 20%, 25% and 30%) and that any task could be executed on any type of processor; (iii) each task sends 1000 phits during each communication. The task workload and power dissipation that depends on the processor type, are randomly generated, whose ranges are from 5% to 30% and from 5uW to 15uW, respectively. Moreover, the partitioning applies 100% as the maximum processor workload and 150uW as maximum energy consumption per processor.

Fig. 8 collects the energy saving and the load balancing improvements of all sets of experiments, where each dot in each curve represents an average of the values reached with the five task connectivity sets (10%, 15%, 20%, 25% and 30%) presented above. For all experiments the reference values were produced inverting the objective of the cost function, i.e. the reference algorithm tries to maximize the energy consumption and to unbalance the workload. Thus, the values of Fig. 8 are percentages of how algorithms improve energy consumption and load balancing, when compared to the values acquired with reference algorithm, respectively.

Fig. 8(a) shows that SA and TS are always able to reduce more energy consumption when compared with the KL* approach. However, the improvements are not sufficiently meaningful if compared to KL*width for experiments with 100 or less tasks, and if compared to KL*depth for experiments with 50 to 150 tasks.

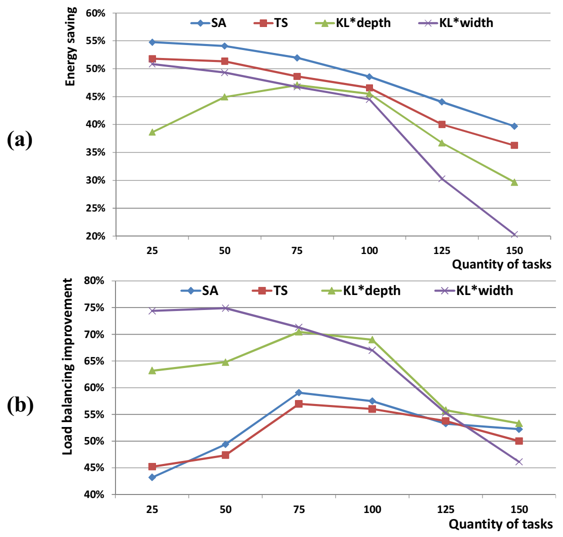

Fig. 8 Improvements on energy consumption and load balancing achieved with partitioning algorithms (TS, SA, KL*depth and KL*width).

In fact, Fig 8(a) shows that KL*width performs better than KL*depth for low complex applications, whereas KL*depth performs better for high complex applications. This behavior probably occurs because KL*depth tries to produce an optimum task-group, before considering optimizations in the remaining tasks. However, when the focused task-group is optimized, it is withdrawn from the partitioning, thus simplifying next steps. In contrast, KL*width leaves the possibility to revisit previously visited task-groups and to apply changes if they minimize the overall cost function. Consequently, for low complex applications, KL*width performs better because it considers the entire set of tasks during partitioning, whereas for high complex problems, KL*depth performs better due to the minimization of the application complexity at each step.

Fig. 8(b) illustrates that the bisection nature of the KL* algorithm enables to produce high quality load balanced partitions, but the application complexity may minimize its gains when KL* is compared to stochastic methods. For some experiments with 125 tasks or more, stochastic methods have demonstrated better load balancing results. The experiments presented on Fig. 8 were acquired concerning around of one million iterations for both methods, which implied an average of 2 minutes for each experiment's execution; whereas KL*width takes merely a second, and KL*depth is still four times faster, on average. This shows the algorithm's efficiency when concerning load balancing.

Finally, it is important to remark that all algorithms are penalized with application complexity, including the one used as reference. This drawback reduces the quality of partitions but also minimizes the difference between the achieved values and the corresponding reference, which is probably one of the main reasons that significant improvements are noticed only for applications with hundred tasks or more.

VI. Conclusion

We show that static partitioning may be used as a dynamic pre-mapping activity in order to conduct to efficient task mappings. In fact, the application of partitioning techniques before mapping is a promising study to reduce design space problem complexity, mainly dealing with applications composed of hundreds of communicating tasks that are dynamically mapped into several processors of heterogeneous MPSoCs. Based on the Kernighan-Lin (KL) approach, this work proposes two new but similar algorithms to perform static partitioning, namely KL*depth and KL*width, whose difference is the way tasks are grouped (i.e. the application graph describing parallel communicating tasks is searched either in depth or in width). The presented experimental results show that both KL* algorithms are static migration methods that, with low energy consumption penalty, enable to achieve good load balanced partitions. A further advantage is the low computational effort needed, when compared with SA and TS.

Acknowledgement

This work is partially funded by FAPERGS under grants PqG 12/1777-4 and Docfix SPI n.2843-25.51/12-3.

References

[1] L. Benini, G. De Micheli. **Networks on chips: a new soc paradigm**. *Computer*, v.35, n.1, pp.70–78, 2002.

[2] U. Ogras, J. Hu, R. Marculescu. **Key research problems in NoC design: a holistic perspective**. *CODES+ISSS*, pp. 69–74, 2005.

[3] H. Faragardi et al. **Reliability-Aware Task Allocation in Distributed Computing Systems using Hybrid Simulated Annealing and Tabu Search**. *HPCC-ICESS*, pp 1088–1095, 2012.

[4] B. Kernighan, S. Lin. **An efficient heuristic procedure for partitioning graphs**. *Bell Sys. Tech. J.*, v.49, n.2, pp.291–307, 1970.

[5] P. Sahu, S. Chattopadhyay. **A survey on application mapping strategies for Network-on-Chip design**. *Journal of Systems Architecture (JSA)*, v.59, n.1, pp.60–76, 2013.

[6] S. Murali et al. **Mapping and Configuration Methods for Multi-Use-Case Networks on Chips**. *ASP-DAC*, pp 146–151, 2006.

[7] C-L. Chou, U. Ogras, R. Marculescu. **Energy- and Performance-Aware Incremental Mapping for Networks on Chip with Multiple Voltage Levels**. *IEEE Trans Comput-aided Des Integr Circuits Syst.*, v.27, n.10, pp.1866–1879, 2008.

[8] C. Marcon et al. **Comparison of network-on-chip mapping algorithms targeting low energy consumption**. *IET Computers & Digital Techniques*, v.2, n.6, pp.471–482, 2008

[9] A. Singh et al. **Mapping Algorithms for NoC-based Heterogeneous MPSoC Platforms**. *DSD*, pp 133–140, 2009.

[10] A. Habibi, M. Arjomand, H. Sarbazi-Azad. **Multicast-Aware Mapping Algorithm for On-chip Networks**. *PDP*, pp 455–462, 2011.

[11] S. Kaushik, A. Singh, T. Srikanthan. **Preprocessing-based Run-time Mapping of Applications on NoC-based MPSoCs**. *IEEE Computer Society Annual Symposium on VLSI*, pp 337–338, 2011.

[12] E. Antunes et al. **Partitioning and Mapping on NoC-Based MPSoC: An Energy Consumption Saving Approach**. *NoCArc*, pp.51-56, 2011.

[13] O. He et al. **UNISM: Unified Scheduling and Mapping for General Networks on Chip**. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, v.20, n.8, pp.1496–1509, 2012.

[14] J. Castrillon, R. Leupers, G. Ascheid. **MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs**. *IEEE Transactions on Industrial Informatics,* v.9, n.1, pp.527–545, 2013

[15] T. Wiangtong, P. Cheung, W. Luk, **Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign**. *DAES*, v.6, n.4, pp.425–449, 2002.

[16] E. Antunes et al. **Partitioning and dynamic mapping evaluation for energy consumption minimization on NoC-based MPSoC**. *ISQED*, pp 451–457, 2012.

[17] A. Singh et al. **Mapping on multi/many-core systems: survey of current and emerging trends**. DAC, pp 1-10, 2013.