Power-Aware Dynamic Mapping Heuristics for NoC-Based MPSoCs Using a Unified Model-Based Approach

LUCIANO OST, LIRMM, CNRS, University of Montpellier II
MARCELO MANDELLI, PUCRS, Brazil
GABRIEL MARCHESAN ALMEIDA, ITIV, KIT
LEANDRO MOLLER, Darmstadt University of Technology
LEANDRO SOARES INDRUSIAK, University of York
GILLES SASSATELLI and PASCAL BENOIT, LIRMM, CNRS, University of Montpellier II
MANFRED GLESNER, Darmstadt University of Technology
MICHEL ROBERT, LIRMM, CNRS, University of Montpellier II
FERNANDO MORAES, PUCRS, Brazil

The mapping of tasks to processing elements of an MPSoC has critical impact on system performance and energy consumption. To cope with complex dynamic behavior of applications, it is common to perform task mapping during runtime so that the utilization of processors and interconnect can be taken into account when deciding the allocation of each task. This paper has two major contributions, one of them targeting the general problem of evaluating dynamic mapping heuristics in NoC-based MPSoCs, and another focusing on the specific problem of finding a task mapping that optimizes energy consumption in those architectures.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*Interconnection architectures*; C.3 [**Computer Systems Organization**]: Special-purpose and Application-based Systems—*Real-time and embedded systems*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms: Design, Performance

Additional Key Words and Phrases: Modeling, NoC-based MPSoCs, design space exploration, power-aware mapping

ACM Reference Format:
Ost, L., Mandelli, M., Almeida, G. M., Moller, L., Indrusiak, L. S., Sassatelli, G., Benoit, P., Glesner, M., Robert, M., and Moraes, F. 2013. Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach. ACM Trans. Embedd. Comput. Syst. 12, 3, Article 75 (March 2013), 22 pages.
DOI: http://dx.doi.org/10.1145/2442116.2442125

Authors' addresses: L. Ost, LIRMM, Montpellier, France; email: ost@lirmm.fr; M. Mandelli, PUCRS, Porto Alegre, Brazil; email: marcelo.mandelli@acad.pucrs.br; G. M. Almeida, KIT, Karlsruhe, Germany; email: gabriel.almeida@kit.edu; L. Möller, Darmstadt University of Technology, Darmstadt, Germany; email: moller@mes.tu-darmstadt.de; L. S. Indrusiak, University of York, York, U.K.; email: lsi@cs.youk.ac.uk; G. Sassatelli, LIRMM, Montpellier, France; email: sassatelli@lirmm.fr; Pascal Benoit, LIRMM, Montpellier, France; email: pascal.benoit@kurnn.fr; Manfred Glesner, Darmstadt University of Technology, Darmstadt, Germany; email: glesner@mes.tu-darmstadt.de; Michel Robert, LiRMM, Montpellier, France; email: michel.robert@lirmm.fr; F. Moraes, PUCRS, Porto Alegre, Brazil; email: fernando.moraes@pucrs.br.

DOI: http://dx.doi.org/10.1145/2442116.2442125

1. INTRODUCTION

HDTV, multiple wireless communication standards, and media players are examples of applications executing in existing embedded systems, requiring high performance allied to low power consumption. To cope with such requirements, the electronic industry has adopted MPSoCs (Multiprocessor Systems-on-Chip), due to their power efficiency and capability to increase system performance by using multiple processing elements (PEs). The interconnection between PEs plays an important role on the system performance, but also accounts for a significant part of the total power consumption [Kahng et al. 2009; Lee and Bagherzadeh 2009]. The present work assumes NoCs (Networks-on-Chip) as the interconnection architecture, due to their scalability, flexibility and power efficiency [Marculescu et al. 2009].

Most embedded applications cause time-varying workloads on the underlying MPSoC [Singh et al. 2010]. In many cases, the variations cannot be accurately predicted during design time, such as the scenarios when the system interacts with complex deployment environments or user-driven requests [Chou and Marculescu 2010]. In those cases, offline mapping techniques are sub-optimal or inadequate [Faruque et al. 2008; Wildermann et al. 2009; Molnos et al. 2010]. A typical example was described in Jalier et al. [2010], where a mobile communication system has to handle multiple data stream processing tasks over 4G mobile communication protocols following the LTE standard.

To cope with such scenarios, the deployment of new techniques to achieve runtime system adaptability is mandatory [Almeida et al. 2009; Molnos et al. 2010]. Dynamic task mapping, task migration, distributed DVFS (dynamic voltage and frequency scaling), and power gating are examples of runtime techniques that make possible to optimize various parameters such as application performance or power consumption.

Because of the complexity of typical MPSoCs, it is necessary to provide system designers with flexible frameworks that enable the exploration of different architectural options, tuning the platform to the applications needs. Such frameworks must provide accurate performance metrics such as latency, throughput, and energy consumption. Furthermore, they must be able to produce those metrics in a short amount of time, since fast evaluation covers larger design space. For instance, design exploration at the register transfer level (RTL) may be unfeasible because a single simulation scenario may easily take several hours or even days. One promising alternative to evaluate MPSoCs is to use accurate abstract models of the application running on a specific platform.

In this context, this work presents a model-based framework that simplifies the development and the validation of MPSoCs by providing flexible application modeling features ([Määttä et al. 2009]), high level and high accuracy NoC models (e.g. [Ost et al. 2009; Indrusiak and Santos 2011]), and dynamic task mapping heuristics, making it possible to rapidly assess resulting gains in terms of performance and power consumption. The fundamental principle behind the multi-layer framework is the complete separation between the different layers—application, mapping, and platform [Indrusiak et al. 2010]. Therefore, new application models, platform templates, and mapping heuristics can be integrated into the framework as long as they follow the pre-defined inter-layer APIs.

The main objective of this paper is to extend the multi-layer framework to support runtime system adaptability by means of dynamic mapping heuristics. The main benefits of employing the multi-layer framework includes (i) design flexibility, since users do not have to deal with detailed platform implementation; (ii) the platform is not restricted to small system configurations; (iii) increased debugging capability.

The contributions of this paper may be summarized as follows: (i) proposition of a power-aware dynamic multi-task mapping heuristic that uses an optimized search

algorithm to define the position allocation of tasks at run-time; (ii) integration of the proposed heuristic, as well as other reference mapping heuristics, into the unified model-based framework; (iii) evaluation of the proposed heuristic, demonstrating its effectiveness in terms of latency and energy consumption reduction within several application scenarios, (iv) validation and execution of the proposed heuristic in a real RTL-model platform.

The paper is organized as follows. Section 2 summarizes the basic aspects of the adopted multi-layer model-based approach. Section 3 describes related works in power-aware dynamic mapping. Section 4 presents the proposed power-aware dynamic mapping. A set of experiments that were used to validated the proposed mapping heuristic are presented in Section 5. Section 6 points out conclusions and directions for future work.

2. MULTI-LAYER MODEL-BASED APPROACH

The proposed approach supports the joint validation of applications mapped onto the platform model, enabling design space exploration. Three *modeling layers* divide the MPSoC design space exploration: (i) application, (ii) mapping, and (iii) platform. The first layer comprises application modeling and validation (functionality and requirements), while the second layer defines how such applications are mapped onto the MPSoC platform (third layer).

The present work provides a library with applications and platform models, as well as static and dynamic mapping heuristics, which can be used for the design space exploration of NoC-based MPSoCs. The main benefit of the multi-layer model-based approach is the separation between such modeling layers, which allows designers to model and to validate the main functionalities of their own applications (e.g., communication behavior) and platform models, individually. Once both models are validated, designers can successively evaluate the performance of an application running over a platform by using several mapping heuristics, allowing extensively exploration of the system. Another advantage of using this multi-layer model is that mapping heuristics can be easily integrated and validated.

2.1. Application Layer

The first layer provides the software engineer with the possibility to develop and validate different applications considering only their functionality and requirements. It uses executable models based on UML sequence diagrams and actor-orientation [Indrusiak et al. 2007; Määttä et al. 2008]. UML is a standard modeling language used by most part of the software development industry due to its flexibility, support to the real time requirements through profiles (e.g., Profile for Modeling and Analysis of Real-Time and Embedded Systems, MARTE [OMG 2010]) and tool support. On the other hand, actor orientation design is a component methodology, which separates the functionality concerns (modeled as actors) from the component interaction concerns (modeled as frameworks). It includes the definition of the execution semantics as a part of the model rather than of the underlying simulation engine. This means that applications can be described using distinct models of concurrent computation (e.g., process networks, event-based synchronization, dataflow) allowing specifications that use time and concurrency primitives that better reflect their nature. As a result, the concurrent behavior and the interdependencies of the application tasks are accurately captured [Lee and Neuendorffer 2004; Lee et al. 2006]. We claim here that the combination of UML and actor-orientation provides increased design flexibility to specify the application tasks, their dependencies, synchronization mechanisms, and data exchanges, when compared to application modeling approaches based on task graphs.

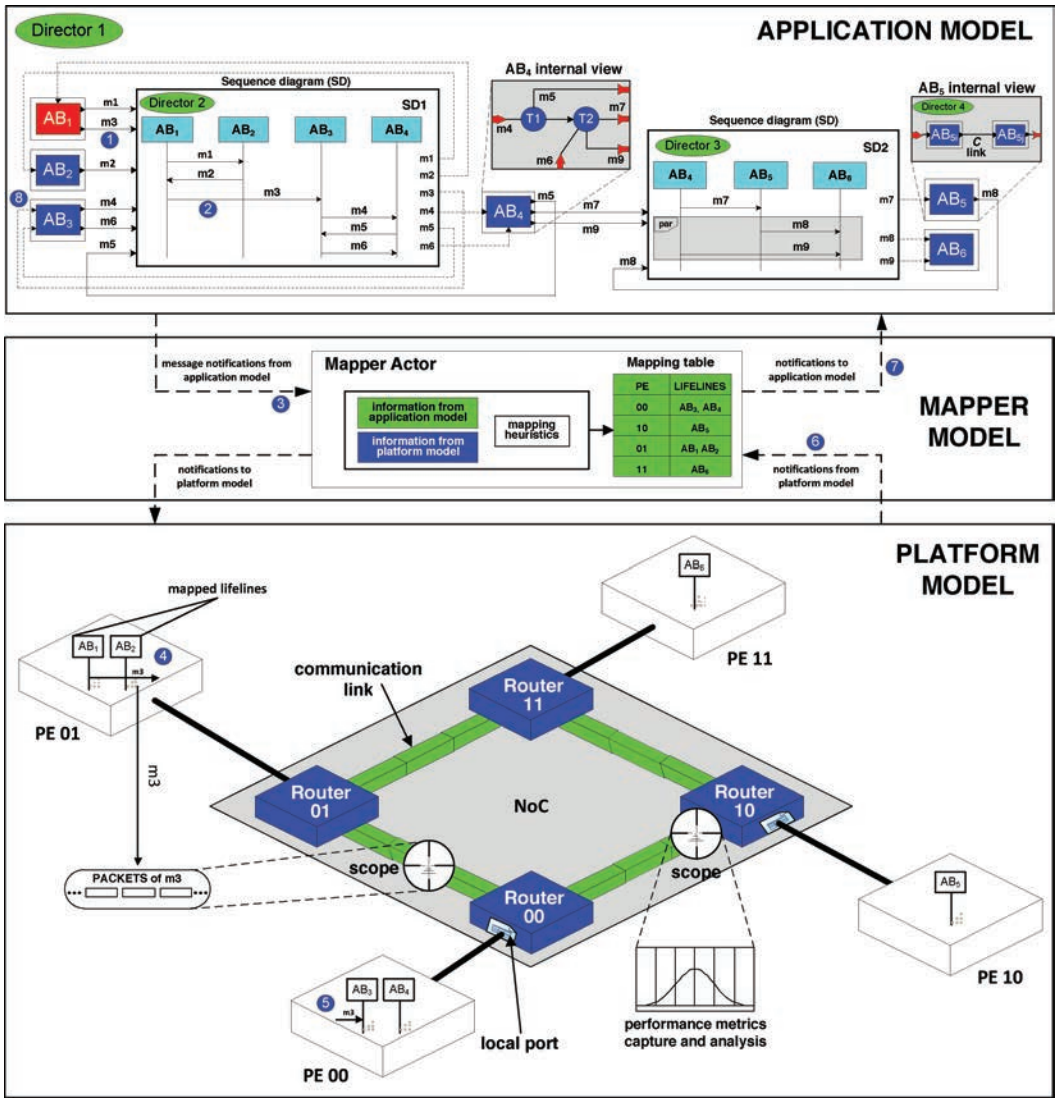

Fig. 1. An example of a unified model representation. For the sake of simplicity, the mapper model layer is represented by a static *Mapper Actor's* behavior.

At the application layer, designers specify application blocks (represented as lifelines), implemented as a set of communicating application actors (e.g., AB_1 and AB_2 in the upper part of Figure 1), according to the modeling strategy proposed in Määttä et al. [2008, 2010].

Application blocks are tasks (e.g., AB_4), which may contain more than one function that are grouped (e.g., *T1* and *T2* represented as circles in Figure 1), according to the software engineer's decision. Application actors have input and output ports for sending and receiving messages. UML sequence diagrams (SDs like *SD1* and *SD2*) are sequencing actors that constrain the order of the messages exchanged by application actors. Application actors are defined as active or passive and are represented as lifelines within one or more SDs (for instance, AB_4 is represented in both *SD1* and

SD2). Active actors (e.g., AB_1) are those that initiate a communication while passive actors (e.g., AB_2) react or initiate a communication after receiving a message from an active actor.

The application model is formally described as follows. Assuming that A is a set of actors and C is a set of communication links, an application model is a directed bipartite multigraph, G(A, C). Given that AB is a set of application actors and SA is a set of sequencing actors, A = AB U SA. For the example in Figure 1, $AB = \{AB_1, AB_2, AB_3, AB_4, AB_5, AB_6\}$, while $AS = \{SD1, SD2\}$. The set $EM = \{m1, m2, m3, m4, m5, m6, m7, m8, m9\}$ represent the messages exchanged by the application actors in the application model illustrated in Figure 1. Following, C is the vertex set of G and represents all communication links in the application model. C is in fact a set of triples with $C \subset AB \times AS \times EM$ U $AS \times AB \times EM$. Each communication link is defined by two triples in C. For the example in Figure 1, AB_1 and AB_2 exchange m_1. This is represented in C by the elements (AB_1, *SD1, m1*), (*SD1,* AB_2*, m1*). The whole C set for the example in Figure 1 is $C = \{(AB_1, SD1, m1), (SD1, AB_2, m1), (AB_2, SD1, m2), (SD1, AB_1, m2), (AB_1, SD1, m3), (SD1, AB_3, m3), (AB_3, SD1, m4), (SD1, AB_4, m4), (AB_4, SD1, m5), (SD1, AB_3, m5), (AB_3, SD1, m6), (SD1, AB_4, m6), (AB_4, SD2, m7), (SD2, AB_5, m7), (AB_5, SD2, m8), (SD2, AB_6, m8), (AB_4, SD2, m9), (SD2, AB_6, m9)\}$.

2.2. Platform Layer

The platform layer provides multi-accuracy and executable platforms models, allowing designers to trade-off accuracy and simulation time. The target architecture is a 2D mesh NoC, with one PE connected to the local port of each router. The adopted NoC-based platform can be seen as a directed graph H = G(PE,C), where PE is a set of processing elements and C are the communication channels through which PEs exchange data packets.

Designers need tools to observe and debug the execution of the set of applications running on top of an NoC model. In this context, the platform layer includes *Scope* actors that can be used to check the running status of the system, as well as to collect performance results that can be used for application/platform model optimization. Examples of *Scopes* used in this work: *PowerScope*, *LatencyScope* and *MapperScope*. The *PowerScope* applies the volume-based power model, generating a power report. The *LatencyScope* provides end-to-end communication latency figures for each task communication (e.g., number of clock cycles to delivery *m1* from AB_1 to AB_2 in the upper part of Figure 1). The end-to-end latency is defined as the delay between the instant a PE starts its message transmission and the time the target PE receives the message. The *MapperScope* is used to monitor the mapping layer (e.g., capturing each task requesting time) and to generate mapping results, as the number of hops among communicating tasks.

2.3. Mapping Layer

The mapping layer is the link between the application and the platform layers, and it is responsible for mapping tasks (lifelines) onto PEs. A lifeline is the smallest grain of the proposed mapping approach and the *Mapper Actor* can map one or more lifelines, according to the adopted mapping heuristic, onto one of the available PEs of the platform. For the example in Figure 1, PE 01 (AB_1 and AB_2) and PE 00 (AB_3 and AB_4) received two tasks, whereas PE 10 and PE 11, received one task each. Thus, keeping the example in Figure 1, let $AB = \{AB_1, AB_2, AB_3, AB_4, AB_5,$ and $AB_6\}$ be the set of n tasks, and PE = {PE00, PE01, PE10, and PE11} be the set of m processing elements of an MPSoC = <PE, C>. A mapping is an injective function f: AB $\rightarrow$ PE, which associates tasks to PEs.

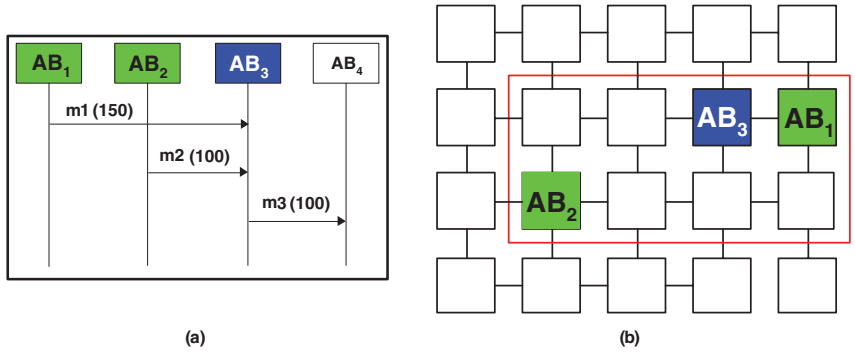

Fig. 2. (a) Application described as UML diagram, where *m1*, *m2*, and *m3* represent the communication volume between tasks; (b) search space to map the AB_3 task, and one possible mapping for AB_3.

The *Mapper Actor* implements the mapping heuristics. The present work supports static [Ost et al. 2010] and dynamic mapping heuristics (described in Section 4 of this work). In the related works the mapping process is manual or pure random selection (e.g., [Kangas et al. 2006; Ha 2008; Pimentel et al. 2008; Mischkalla et al. 2010]). In a concrete MPSoC implementation, a given processor executes the function of the Mapper Actor, where the MPSoC might have a centralized control [Carara et al. 2009], or it can be controlled by the operating system running onto each PE in a decentralized fashion.

2.4. Unified Model Execution

This section illustrates the interactions among the modeling layers. The numbers inside blue circles in Figure 1 exemplify the interactions in the unified model. Considering that messages *m1* and *m2* were already sent by and received at their respective applications blocks, the application block AB_1 then sends the message *m3* to the sequencing actor SD1, which receives it (event 1 in application layer of Figure 1). When *m3* is received, a message within its sequence diagram is triggered (event 2 in application layer, the message *m3* sent from lifeline AB_1 to lifeline AB_3). When this happens, the corresponding director *Director 2* interrupts the message delivery and notifies the *Mapper Actor* about the message (event 3 in mapper layer). Following the guidelines stated in Lee and Neuendorffer [2004], such directors (link between application and mapper layers) control the messages' delivery order (e.g., total or partial) between application blocks, considering the communication behavior described in its sequence diagram.

Since the *Mapper Actor* is responsible for assigning each lifeline to a PE, it knows that for instance lifeline AB_1 is mapped to PE 01, whereas AB_3 is mapped to PE 00. Once the *Mapper Actor* receives the information about the triggered message, it will command the processing element (PE 01) associated to the sender of the message (*m3*) to generate the corresponding traffic into the NoC platform. Thus, it creates a packet and writes it on the local input port of the corresponding Router 01 (event 4 in the platform layer). The platform model simulates the message delivery by transmitting packets from the source to the target tasks (e.g., packets of *m3* in Figure 2). Then the *Mapper Actor* waits until the processing element (PE 01) associated to the receiver of the message (*m3*, event 5 in the platform layer) notifies the complete reception of the packet (event 6 in the mapper layer). Upon notification, the *Mapper Actor* calls back the *Director 2*, which has notified the triggering of the message, and informs it that the message can now be delivered (event 7 in the application layer). After that, the *Director 2* can forward the message to the output port of the SD1, and the message reaches its

destination (AB_3) with the exact latency that it would take if the application had been executed on top of the implementation platform (event 8 in the application layer).

3. BASIC CONCEPTS AND DYNAMIC MAPPING REVIEW

Task mapping literature is wide, requiring a taxonomy to classify different mapping approaches. In this context, we classify the mapping process according to four criteria: (i) the moment in which it is executed; (ii) the number of tasks per PE; (iii) the entity responsible for controlling the mapping; (iv) the target architecture.

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 (such as simulated annealing or genetic algorithms) 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, it requires simple and fast heuristics since it may interfere with the applications execution time. Two dynamic mapping approaches are found in the literature.
  —*With resources reservation*. The mapping heuristics verify if there are enough resources in the MPSoC before mapping the application tasks. The clear advantage of this method is to guarantee the complete application mapping. On the other side, the application may demand more time to start its execution if resources are not available when the application is requested.
  —*Without resources reservation*. The mapping heuristics map one or more initial tasks of the application (those without dependences on other tasks), placing the remaining tasks whenever necessary. This approach may start applications faster, but some tasks may wait for available resources if the system usage is high.
—*Dynamic re-mapping.* Also called task migration, it may employ dynamic mapping heuristics together with some criteria (e.g., PE workload, communication overhead) in order to increase the overall performance of the system (e.g., load balancing).

Considering the number of tasks mapped per PEs, the following approaches may be used:

—*Mono-task*. In this approach, only one task is assigned to each PE.
—*Multi-task*. In this approach, clustering methods are used within the mapping heuristics to group tasks according to some criteria, such as communication, execution time and task deadlines. As PEs are often processors executing multi-task operating systems, multi-task mapping can better utilize existing resources.

Dynamic mapping requires an entity, called in this paper a *Mapper Actor*, responsible for mapping the tasks at runtime. The mapping control may be the following.

—*Centralized*. One PE is responsible for receiving the mapping requests, reading the task code from an external memory (task repository), executing the mapping heuristic, and sending the task to the chosen PE. This approach is not scalable, and may lead to hotspots[1] in the NoC, while reducing the performance of the whole system.
—*Distributed*. The MPSoC is divided in regions (clusters), and one PE in each region is responsible for executing the mapping heuristic on it. Despite the increased scalability, one bottleneck subsists: the access to the task repository, which is global to the system.

[1]This term refers to instants where power dissipation reaches a peak value, which may increase the temperature at specific regions of the chip that can, consequently, generate hotspots.

Finally, the mapping can be classified according to the architecture model.

—*Homogeneous*. All PEs are identical. This makes task mapping and task migration easier, because it is not necessary to consider the PE type in the heuristics.
—*Heterogeneous*. Different PEs are used in the same architecture, such as DSPs, dedicated IPs, accelerators. Before mapping, a binding process is executed, assigning to each PE the task that it may execute.

3.1. Power-Aware Dynamic Mapping Review

According to the literature, static mapping will be insufficient to handle the dynamic and unpredictable behavior of future embedded applications running in parallel [Chou and Marculescu 2008; Hölzenspies et al. 2008, Schranzhofer et al. 2010]. Thus, research approaches on power-aware dynamic mapping have been proposed.

In Smit et al. [2005], an iterative hierarchical dynamic mapping approach is proposed. This approach aims to reduce energy consumption of the SoC, while providing the required quality-of-service (QoS). In such a strategy, tasks are first grouped by being assigned to a system resource type (e.g., FPGA, DSP, ARM), according to performance constraints. Then, each task inside a group is mapped, minimizing the distance among them and reducing communication cost. Finally, the resulting mapping is checked, and if it does not meet the application requirements, a new iteration is required.

In Ngouanga et al. [2006] a force-directed mapping heuristic is presented for homogeneous NoC-based MPSoCs. The heuristic selects the new position for a task according to a resulting force proportional to the communication volume and distance between tasks.

Chou and Marculescu [2007] introduce an incremental dynamic mapping process approach, where PEs connected to the NoC have multiple voltage levels, while the network has its own voltage–frequency domain. A global manager (OS-controlled mechanism) is responsible for finding a contiguous area to map an application, and for defining the position of the tasks within this area, as well. According to the authors, the strategy avoids the fragmentation of the system and aims to minimize communication energy consumption, which is calculated according to Ye et al. [2002]. This work was extended in Chou and Marculescu [2008, 2009], incorporating the user behavior information in the mapping process. The user behavior corresponds to the application profile data, including the application periodicity in the system and data volume transferred among tasks. For real applications considering the user behavior information, the approach achieved around 60% energy savings compared to a random allocation scenario.

Hölzenspies et al. [2008] investigate a run-time spatial mapping technique with real-time requirements, considering streaming applications mapped onto heterogeneous MPSoCs. In the proposed work, the application remapping is determined according to a set of information (i.e., latency/throughput) that is collected at design time, aiming to satisfy the QoS requirements, as well as to optimize the resources usage and to minimize the energy consumption. A similar approach is proposed in Schranzhofer et al. [2010], merging pre-computed template mappings (defined at design time) and online decisions that define newly arriving tasks to the PEs at run-time. Compared to the static-mapping approaches, obtained results reveal that it is possible to achieve an average reduction on power dissipation of 40–45%, while keeping the introduced overhead to store the template mappings as low as 1kB. Another power-aware approach is presented in Wilderman et al. [2009]. This approach employs a heuristic that includes a Neighborhood metric inspired by rules from Cellular Automata, which allows decreasing the communication overhead and, consequently, the power dissipation imposed by dynamic applications. In Hosseinabady and Nunez [2010] a stochastic dynamic task mapping and a routing algorithm are used to minimize reconfiguration overhead. Lu

et al. [2010] propose a dynamic mapping algorithm, called Rotating Mapping Algorithm (RMA), which aims to reduce the overall traffic congestion (take in account the buffer space) and communication energy consumption of applications (reduction of transmission hops between tasks).

In turn, Mandelli et al. [2011] propose a power-aware task mapping heuristic, which is validated using a NoC-based MPSoC described at the RTL level, with a clock-cycle accurate ISS describing processors. The mapping heuristic is performed in a given PE of the system that executes a preemptive operating system [Carara et al. 2009]. Due to the use of a low level description, accurate performance evaluation of several heuristics (execution time, latency, energy consumption) is supported. However, the scope of the work is limited to small systems configurations due to the simulation time. In the previous works, only one task is assigned to each PE. A multi-task dynamic mapping approach was proposed in Singh et al. [2010]. This work extends the work described in Carvalho et al. [2009], which uses an RTL NoC and abstract PEs implemented in System-C. In this work, different mapping heuristics were used to evaluate the power dissipation as the product of number of bits to be transferred and distance between source-destination pair. Differently from other works, Faruque et al. [2008] and Weichslgartner et al. [2011] proposed the use of distributed dynamic mapping approaches. Faruque et al. [2008] present a distributed agent-based mapping scheme. The proposed scheme divides the system into virtual clusters. A cluster agent (CA) is responsible for all mapping operations within a cluster. Global agents (GAs) store information about all the clusters of the NoC and use a negotiating policy with CAs in order to define to which cluster an application will be mapped. In turn, Weichslgartner et al. [2011] explore different implementations of a decentralized self-embedding algorithm, aiming to minimize network contention and latency while providing fault-tolerance support for NoC-based systems.

Table I summarizes the reviewed works according to the proposed taxonomy for dynamic mapping. Most reviewed works reserve resources according to the number of the tasks, defining, for instance, pre-computed mapping templates for each application [Hölzenspies et al. 2008]. On the other side, Carvalho et al. [2009] and Singh et al. [2010] do not employ resource reservation but rather allocate PEs whenever actually required. Considering that not all tasks execute concurrently, allocating resources for all application tasks may underutilize the MPSoC, as well as possibly requiring larger systems. The work present in Singh et al. [2010] is the only one to support dynamic multi-task mapping. Excluding the work proposed by Faruque et al. [2008] and Weichslgartner et al. [2011], the mapping control management is centralized; which can become the bottleneck for large multiprocessors systems.

In the context of dynamic mapping heuristics, the present work proposes a mapping heuristic according to the last line of Table I. The homogenous architecture model may be easily extended to heterogeneous architectures by adding a binding heuristic before the mapping process. The centralized approach is a drawback for large MPSoCs with hundreds of PEs. One way to move to distributed mapping is to divide the MPSoC in clusters, including a mapper at each cluster. In the scope of this work, our goal is to propose a mapping heuristic (presented in Section 4.3), integrated into the unified multi-layer model, described in Section 2.

4. DYNAMIC MAPPING HEURISTICS

This section presents the dynamic mapping heuristics integrated in the Mapper Actor. Section 4.1 presents the mapping heuristics used as reference for the experiments. Section 4.2 details the DN (Dependences Neighborhood) and LEC-DN (Low Energy Consumption-Dependences Neighborhood). Following, the proposed Premap-DN is presented in Section 4.3.

Table I. Related Work Classified According to the Proposed Taxonomy for Dynamic Mapping Heuristics

Author Year	Resource reservation	Multi/Mono task	Architecture model	Control manager	Optimization Goal
Smit et al. [2005]	Yes	Mono-task	Heterogeneous	Centralized	Energy Consumption and QoS application requirements
Ngouanga et al. [2006]	Yes	Mono-task	Homogeneous	Centralized	Communication volume, Computation load
Chou and Marculescu [2007, 2008, 2010]	Yes	Mono-task	Homogeneous	Centralized	Energy Consumption, Internal and external Network contention
Hölzenspies et al. [2007, 2008]	Yes	Mono-task	Heterogeneous	Centralized	Energy Consumption and QoS application requirements
Al Faruque et al. [2008]	No	Mono-task	Heterogeneous	Distributed	Execution time, Mapping time and Monitoring traffic
Wildermann et al. [2009]	No	Mono-task	Homogeneous	Centralized	Communication Latency, Energy consumption
Hosseinabady and Nunez [2009]	Yes	Mono-task	Homogeneous	Distributed	Reconfiguration overhead
Schranzhofer et al. [2010]	Yes	Mono-task	Heterogeneous	Centralized	Energy Consumption
Lu et al. [2010]	not available	Mono-task	Homogeneous	Centralized	Communication Latency and Energy Consumption
Carvalho et al. [2010]	No	Mono-task	Heterogeneous	Centralized	Network contention, Communication volume
Singh et al. [2009, 2010]	No	Multi-task	Heterogeneous	Centralized	Network contention, Communication volume and energy consumption
Weichslgartner et al. [2011]	Yes	Mono-task	Homogeneous	Distributed	Communication Latency and Network contention
Proposed work	**No**	**Multi-task**	**Homogeneous**	**Centralized**	**Energy Consumption**

All application tasks are dynamically mapped, except the *initial task(s)* (those that do not have dependences to other tasks, e.g., AB_1 in Figure 1, modeled as active actors). In the present work, their positions in the MPSoC are manually defined. The criterion to select the position of a given initial task in the MPSoC is to virtually divide the MPSoC in k regions, k being the number of applications to be executed simultaneously, and assign the initial task of each application in the center of each region. The reasoning for selecting this criterion is to minimize the resource sharing among tasks. As future work, the initial task mapping can be easily integrated into the *mapping layer*.

4.1. Reference Mappings Heuristics

Due to its simplicity, the nearest Neighbor (NN) is used as reference mapping heuristics for dynamic single-task and multi-task purpose. The NN mapping considers only the proximity of an available resource to execute the required task. NN starts searching for a free PE able to execute the task near the requesting task. The search tests all n-hop neighbors, n varying from 1 to the NoC size in a spiral way, stopping when the first PE free is found.

The NN heuristic was extended and integrated in the proposed model-based approach to support multi-task mapping, according to the definitions presented in Singh et al.

[2010]. In this extension, the *Mapper Actor* verifies if the requesting task PE (0 hop distance) is able to receive the required task instead of looking for the neighbor PEs at 1 hop distance. This extended version is used as a reference dynamic multi-task heuristic.

4.2. DN and LEC-DN Mappings Heuristics

Differing from the NN heuristic, which tries to map the requested task as closely as possible to the requesting task, the DN heuristic considers all dependencies between tasks, mapping the requested task as closely as possible to the already mapped tasks with which it communicates, by employing a proximity (in number of hops) cost function.

The LEC-DN heuristic extends the DN, employing two cost functions: (i) proximity in number of hops and (ii) communication energy involved in the data transfer/reception between tasks. This second criterion employs the volume-based energy model proposed by Hu and Marculescu [2005] to select the position of the task to be mapped, and it is used when a given task communicates with at least two mapped tasks. In this situation, the new task is mapped closer to the task with higher communication volume.

When the requested 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 tasks.

Consider the application illustrated in Figure 2(a), containing 4 tasks, where AB_1 and AB_2 are initial tasks. The mapping of task AB_3 is fired by the first communication with it. The search space to map task AB_3 corresponds to the bounding box defined by the position of AB_1 and AB_2 tasks (Figure 2(b)). Thus, AB_3 will be mapped nearest to task AB_1, since according to the application sequence diagram the communication volume $AB_1 \rightarrow AB_3$ is higher than $AB_2 \rightarrow AB_3$. Note that task AB_4 is not mapped, since it depends on the task AB_3.

The implementation of the LEC-DN heuristic is described in Algorithm 1. The heuristic starts by inserting the mapped tasks that communicate with the requested task t_i in the *communicating_tasks_list* (line 3). If the *communicating_tasks_list* has just one element (c_0), the NN algorithm is executed (lines 6–19). If the *communicating_tasks_list* has more than one communicating task mapped (lines 21–44), the algorithm first set the energy to a maximum value (line 22), and defines the coordinates of the bounding box according to the elements inside the *communicating_tasks_list* (line 23). Line 26 gets all PEs inside the bounding box. The loop between lines 27 to 39 computes the communication energy to map each PE inside the bounding box (line 32), selecting the PE that results in the mapping with the smallest energy cost. The bounding box size is increased one hop in all directions (line 41), if there is not a free PE inside the bounding box. The loop between lines 24–43 is repeated until finding an available *pe*, or untill all PEs were visited. If no *pe* is found, the *Mapper Actor* schedules the task t_i to be mapped when a PE becomes available.

The theoretical complexity of LEC-DN is $O(|PE|^2)$, where $|\mathrm{PE}|$ corresponds to the number of processing elements. The algorithm part comprising the NN execution (lines 6–19) has two nested loops (line 8 and line 11). The loop starting at line 8 varies the parameter *dist* from 1 to the NoC size (maximum of the x or y NoC side). The inner loop, line 11, visits all PEs at a *dist* distance. In this way, in the worst case when executing the code between lines 6–19, all PEs are visited once. Therefore, the complexity of the first part of the algorithm is $O(|\mathrm{PE}|)$.

The second part of the algorithm (lines 22–43) has three nested loops (lines 24, 27, 31). The loops started at lines 24 and 27 can be seen as one, since the goal is to visit all PEs of the MPSoC once. The inner loop (lines 31–33) may also visit all PEs, if the task to be mapped is connected to all other tasks. Therefore, the complexity of the second

ALGORITHM 1: LEC-DN Mapping Heuristic Pseudocode

Input: A task t_i to be mapped, the set T_i containing the tasks t_i communicates with.
Output: The PE to map the task t_i.

```
1. pe ← −1
2. // Get all t_i communicating tasks already mapped
3. communicating_tasks_list ← mapped_tasks(T_i)
4. // If there is only one communicating task
5. if size(communicating_tasks_list) = 1 then
6.    dist ← 1 // Initializes the search distance to 1 hop
7.    // Search until the distance dist = NoC_size
8.    while dist!= NoC_size do
9.       c_0 ← first element of the communicating_tasks_list
10.      pe_list ← neighbors(dist, c_0) // Get all neighbors of c_0 within a distance dist
11.      for all elements p_i in pe_list
12.         // Checks PE occupation
13.         if state(p_i) = free then
14.            pe ← p_i
15.            return pe // PE is found, stop searching
16.         end if
17.      end for
18.      dist ← dist + 1 // Go to next range of neighbors
19.   end while
20. end if
21. if size(communicating_tasks_list) > 1 then
22.   min_energy ← ∞ // Initializes energy with highest value
23.   bounding_box←area(communicating_tasks_list) //set the bounding box's coordinates
24.   while while bounding box <= NoC_size and pe= −1 do
25.      // Get the PEs inside the bounding box
26.      pe_list ← search_PEs(bounding_box)
27.      for all elements p_i in pe_list
28.         // Checks PE occupation
29.         if state(p_i) = free then
30.            energy ← 0
31.         for all c_i in communicating_tasks_list
32.            energy← energy + communication_volume(c_i,p_i)*distance(c_i,p_i)
33.         end for
34.         // Set the minimum energy and the target PE
35.         if energy_pe < min_energy then
36.            min_energy ← energy_pe
37.            pe ← p_i
38.         end if
39.      end for
40.      if pe = −1 then
41.         increase(bounding_box)
42.      end if
43.   end while
44. end if
45. return pe
```

part of the algorithm is $O(|PE|^2)$. In practice, each task is connected to few other tasks, minimizing the search space in the inner loop (lines 31–33). The experiments demonstrate that the execution time of the heuristic is small, since in most applications, tasks send/receive messages from few other tasks.

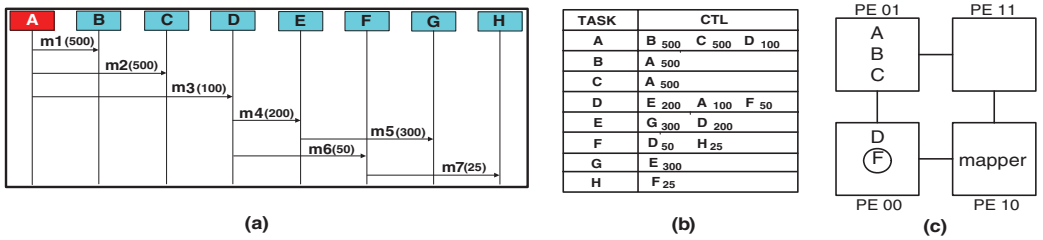

Fig. 3. Premap example, considering up to three tasks per PE.

Before integrating LEC-DN in the *Mapper Actor*, LEC-DN was validated in a homogeneous NoC-based MPSoC, described in synthesizable VHDL [Mandelli et al. 2011]. LEC-DN reduced the execution time (4.3% on average) and the energy consumption (10.8% on average), when compared to other dynamic mapping heuristics (NN and Best Neighbor -BN).

4.3. Premap-DN

This section proposes a multi-task dynamic mapping, called *Premap-DN*. The *Premap-DN* integrates the LEC-DN heuristic and the *premap* clustering method. The goal of the *premap* clustering method is to group a set of communicating tasks into the same PE, without reserving resources for the whole application. Thus, when a given task is pre-mapped, only its placement is reserved. Therefore, the effective mapping of the pre-mapped tasks is executed when it is requested. Figure 3(a) shows a sequence diagram that describes the communication pattern of an application with 8 tasks, t_A being the initial task. The *premap* uses a data structure named CTL (communication task list), created by the *CTL Capture actor*, during system startup. This CTL structure is illustrated in Figure 3(b). 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(c) shows a possible partial pre-mapping of the application (Figure 3(a)) onto a 2x2 NoC-based MPSoC, which is composed of three PEs that are able to execute up to 3 tasks each. Note that one of the PEs is responsible for managing the MPSoC, including the mapping process.

During the system startup, the initial tasks are mapped according to the positions chosen by the designer, and a first execution of the *premap* occurs. In the example illustrated in Figure 3, the initial task t_A is mapped onto PE 01. The Mapper Actor selects a task t_j in the set $C(t_A)$ *iff* t_j does not have any communication with higher volume than its communication with t_A. In the example $C(t_A) = \{t_B, t_C, t_D\}$, $C(t_B) = \{t_A\}$, $C(t_C) = \{t_A\}$, $C(t_D) = \{t_E, t_A, t_D\}$, as t_B and t_C are the ones with higher communication with t_A, and they only communicate with this task, they are pre-mapped with t_A onto PE 01. During t_A execution, t_B and t_C are required to be mapped. As they were already pre-mapped, LEC-DN is not executed and they are effectively mapped (assignment of the lifelines to the PEs). Next t_D is required to be mapped. As this task was not pre-mapped, the LEC-DN chooses PE 00, which is the nearest PE to t_A (it could also be PE 11 in this case). As t_D "opened" a new PE, the *premap* is executed. In this case $C(t_D) = \{t_E, t_A, t_F\}$, $C(t_E) = \{t_G, t_D\}$, $C(t_A) = \{t_B, t_C, t_D\}$, $C(t_F) = \{t_D, t_H\}$. The *premap* evaluates the task that will be mapped with t_D according to $C(t_D)$ order. Thus, the first task to be evaluated will be the one with highest communication with t_D, that is, t_E. Next t_F is evaluated, and it is pre-mapped with t_D because the communication volume t_F–t_D ($m6(50)$) is higher than t_H–t_F ($m7(25)$). PE 00 finished the *premap* with two tasks. The third task slot can be used later by LEC-DN.

The implementation of the *premap* clustering method is described in Algorithm 2. The method begins assigning to the set T_i, the list of all the tasks that t_i communicates with (line 2). Then nT_i receives the non-mapped tasks of T_i (line 4). The next step evaluates each task d_i from nT_i, to choose which tasks will be pre-mapped onto p_i. This evaluation (line 7–17) happens while the PE has less than *TASK_PER_PE* (the maximum number of tasks supported per PE) mapped/pre-mapped tasks onto it, or if all possible tasks in nT_i were already evaluated. For each task d_i, the first task h_i of its CTL D_i is obtained (line 11). So, the communication volume of h_i is compared to the communication volume t_i (line 12) to verify if d_i communicates with the highest volume with t_i. In an affirmative case, d_i is pre-mapped onto p_i, also increasing the p_i task number of mapped/pre-mapped tasks (line 13-16). Otherwise, if available, another task from nT_i is evaluated.

ALGORITHM 2: Premap Method Pseudocode

Input: The PE p_i, the task t_i mapped onto p_i
Output: A set of tasks pre-mapped onto p_i

```
1. // T_i contains the tasks which ti communicates with
2. T_i ← list (T)
3. // nT_i contains all tasks in T_i which were not mapped.
4. nT_i ← non-mapped_tasks(T_i)
5. // Get the first task in the nT_i
6. d_i ← first (nT_i)
7. while tasks(p_i)<TASKS_PER_PE or !end(nT_i) do
8.    // D_i contains the tasks which d_i communicates with
9.    D_i ← list(d_i)
10.   // Get the first task h_i (with highest communication volume) in D_i
11.   h_i ← first(D_i)
12.   if comm_volume(h_i) > comm_volume(t_i) then
13.      // pre-map d_i into p_i
14.      premap(d_i, p_i)
15.      // increase the number of mapped/pre-maped tasks into p_i
16.      tasks(p_i)++
17.   end if
18.   // Get the next task in the nT_i
19.   d_i ← next(nT_i)
20. end while
```

The theoretical complexity of *premap* is O(*TASKS_PER_PE** | PE |). The loop started in line 7 executes, in worst case, *TASKS_PER_PE** | PE | times, since it is possible to have a given task connected to all other tasks. The maximum number of tasks is defined by the number of processors, | PE |, multiplied by the number of tasks each processor can simultaneously execute, *TASKS_PER_PE*. In this case, $|nT_i| = TASKS\_PER\_PE^*$ | PE | . In practice, as previously mentioned, each task communicates with few others tasks, resulting in small execution time.

The *premap* method and the LEC-DN were integrated into the *Mapper Actor*, which executes the mapping process according to the diagram illustrated in Figure 4. When a task t_i is requested to be mapped, the Mapper Actor initially checks if there is some available PE in the system. If there is no available PE, the task t_i 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 pre-mapped. 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

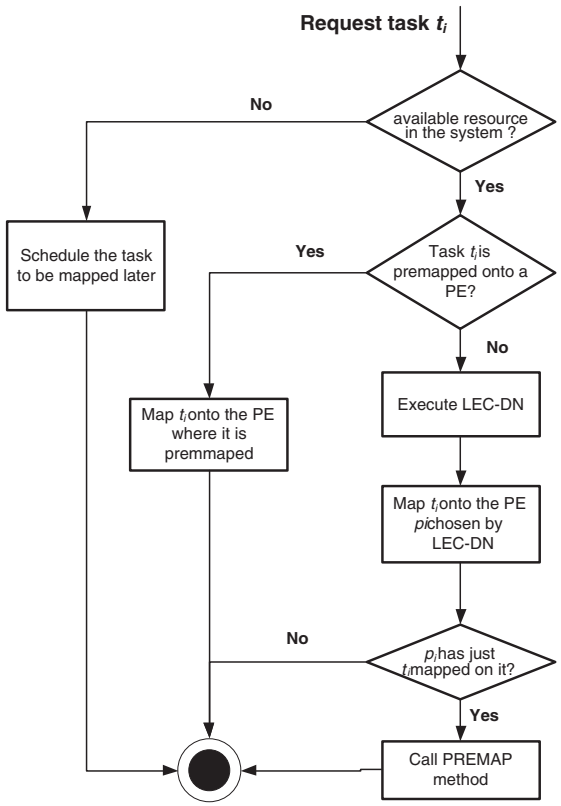

Fig. 4. Integration of the *premap* and LEC-DN into the *Mapper Actor*.

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

5. RESULTS

This section illustrates the use of the multi-layer model-based approach to evaluate different system configurations.

—*Application layer*. Six applications were modeled: (i) VOPD (Video Object Plan Decoder), with 12 application blocks, transmitting 30 fps [Milojetic et al. 2009]; (ii) MWD (Multi-Window Display) with 12 application blocks [Marcon et al. 2008]; (iii) Automotive application, with 10 application blocks [Määttä et al. 2008]; (iv) Circuit Application, a synthetic application with 4 application blocks; (v) Image Segmentation, with 6 application blocks; (vi) MPEG4 decoder, with 12 application blocks and transmitting 30 fps [Milojetic et al. 2009].

—*Mapping layer*. Three mono-task dynamic heuristics: NN, DN, LEC-DN; four multi-task heuristic: NN (multi-task), CNN (Communication-aware Nearest Neighbor) [Singh et al. 2010], PNN (Packing-based Nearest Neighbor) [Singh et al. 2010] and the proposed Premap-DN. For the multi-task heuristics, the number of tasks per PE varies among 3 to 5.

—*Platform layer*. The set of fixed platform parameters are 2D-mesh topology, XY routing algorithm, 32-bit flit size, packets with 128 flits and handshake control flow. The NoC power model was calibrated using X-FAB XCMOS (extended CMOS) 180 nm 1.8 V process, adopting clock gating and a 250 MHz clock frequency. The design space exploration varied only the NoC dimensions. It would be also possible to vary the buffer depth, the routing algorithm, and the packet size, as explored in Ost et al. [2009, 2010]. The platform used to evaluate the mapping heuristics is configured as follows: 7×6 (41 general PEs and one manager PE) for mono-task mapping and 3×5 (14 general PEs and one manager PE) for multi-task mapping.

The adopted applications were chosen since several authors in the NoC research field employ them as benchmarks due to their characteristics. For instance, the MPEG4 application contains as a relevant feature a high degree of connections: 2 of the 12 tasks are connected to upto 7 other tasks. The VOPD application presents smaller inter-task dependency when compared to MPEG4. The Automotive and the Segmentation Image applications present several parallel communications. Circuit and MWD present a dataflow behavior. These applications are grouped in four scenarios, with applications executing simultaneously for one second, allowing at least one application iteration (e.g., one video decoding) of each application.

—Scenario A. 30 tasks: MPEG4, VOPD, and Image Segmentation.
—Scenario B. 40 tasks: MPEG4, VOPD, MWD and Circuit.
—Scenario C. 34 tasks: MPEG4, MWD, Image Segmentation and Circuit.
—Scenario D. 38 tasks: MPEG4, VOPD, Automotive and Circuit.

5.1. Communication Energy Consumption Evaluation

Table II presents the communication energy consumption in the NoC (routers and links), with the gain w.r.t the NN heuristic. The first column of Table II presents the maximum number of tasks that each processor may receive. The single and multi-task NN mapping heuristic is used as reference.

The first analysis concerns the advantage of the multi-task mapping to reduce the communication energy consumption. As expected, the multi-task mapping reduces the communication energy consumption compared to the single-task mapping, due to the fact that the number of communications (messages exchanged) through the NoC decreases. For instance, comparing the single-task NN to the multi-task implementation (first column of Table II), the observed reduction is on average 39.5%, 53.3%, and 53.7% for 3, 4, and 5 tasks per processor, respectively. The communication energy would be zero if all tasks were mapped in the same PE. However, this is not possible, since each PE may execute a limited set of tasks simultaneously, and this would penalize the application execution time.

The second analysis concerns the single-task mapping (*1 task/PE*). As can be observed, the LEC-DN performs similarly to NN, with a reduction of 4.36% in scenario B (with 40 tasks to be mapped in 41 PEs). The small difference observed between the heuristics is explained by the choice of the initial tasks in *virtual regions* (as explained in Section IV), reducing the resource sharing among tasks.

The third analysis concerns the multi-task mapping. As the MPSoC has 14 PEs to execute tasks (NoC size equal to 3×5 in the multi-task evaluation), it is possible to execute simultaneously up to 42, 56, and 60 tasks for 3, 4, and 5 tasks per processor, respectively. Observing the lines *3 tasks/PE* in the table, the mapping heuristics are stressed in this case due to small search space (number of PEs almost equal to the number of tasks to be executed). On average, the proposed *Premap-DN* reduced the energy consumption by 8.5%, reducing the energy consumption in the best case by 16.9%. Increasing the number of tasks per processor reduces the advantages of

Table II. Evaluation of the Communication Energy consumption (nJ) and the Gain/Loss (%) when Compared to the NN Heuristic

SCENARIO A									
HEURISTIC	NN	DN	LEC-DN	CNN		PNN		PREMAP-DN	
1 task/PE	17,255	19,279	−11.37%	16,993	1.52%	-	-	-	
3 tasks/PE	9,490	-	-	9,233	2.71%	9,186	2.71%	8,855	6.69%
4 tasks/PE	8,273	-	-	9,219	−11.43%	9,141	−11.43%	7,847	5.15%
5 tasks/PE	8,155	-	-	8,837	−8.36%	8,758	−8.36%	8,155	0.00%
SCENARIO B									
HEURISTIC	NN	DN	LEC-DN	CNN		PNN		PREMAP-DN	
1 task/PE	16,864	16,129	4.36%	16,129	4.36%	-	-	-	
3 tasks/PE	9,717	-	-	8,527	12.25%	10,868	−11.85%	8,074	16.91%
4 tasks/PE	7,030	-	-	7,868	−11.92%	7,727	−9.91%	6,881	2.12%
5 tasks/PE	7,282	-	-	7,498	−2.97%	7,730	−6.15%	7,126	2.14%
SCENARIO C									
HEURISTIC	NN	DN	LEC-DN	CNN		PNN		PREMAP-DN	
1 task/PE	11,885	12,640	−6.35%	12,010	−1.05%	-	-	-	
3 tasks/PE	8,466	-	-	7,627	9,91%	7,941	6.20%	7,654	9.59%
4 tasks/PE	6,647	-	-	6,772	−1,88%	6,475	2.59%	6,475	2.59%
5 tasks/PE	6,121	-	-	6,324	−3,32%	5,965	2.55%	5,965	2.55%
SCENARIO D									
HEURISTIC	NN	DN	LEC-DN	CNN		PNN		PREMAP-DN	
1 task/PE	17,553	16,797	4.31%	17,996	−2.52%	-	-	-	
3 tasks/PE	10,215	-	-	11,304	−10.66%	8,709	14.74%	10,166	0.48%
4 tasks/PE	7,245	-	-	8,621	−18.99%	8,389	−15.79%	7,204	0.57%
5 tasks/PE	7,635	-	-	8,320	−8.97%	7,991	−4.66%	7,387	3.25%

Premap-DN, since the search space increases and simpler heuristics in this case may also produce good mapping solutions. Note that in all cases, the *Premap-DN* is similar or outperforms *NN*. Different behavior is observed with heuristics CNN/PNN, which present worse results than NN in most cases. The reason CNN presents worse results is the cost function of maping new tasks, which requires empty PEs or PEs with tasks belonging to the same application. The worst results observed with the heuristic PNN come from the packing strategy and the corresponding search method.

Therefore, the proposed *Premap-DN* effectively reduces the communication energy, compared to other state-of-the-art heuristics. It is important to highlight that the *Premap-DN* presents the higher communication energy consumption reduction when the number of available resources is small (e.g., up to 3 tasks/PE), due to the proposed resource reservation strategy.

5.2. Latency Evaluation

Latency is an important metric related to the mapping heuristics. A non-optimized mapping may increase the latency and energy consumption due to distance between tasks and to the congestion induced inside the NoC.

As in the communication energy consumption evaluation, the multi-task mapping also reduces the latency when compared to the single-task mapping. In scenarios A and C, the latency reduction is superior by 20% for 4 and 5 tasks per processor.

The obtained latency results were similar for all evaluated scenarios and heuristics. Only PNN and the proposed *Premap-DN* achieved latency reduction (average and accumulated) superior to 1% when compared to NN. For the *Premap-DN*, in scenario A the latency reduction w.r.t NN was 4.7% and 4.5% for 3 and 4 tasks/PE, respectively,

Table III. Latency in Clock Cycles for the Premap-DN Heuristic for the 4 Simulated Scenarios where

	Scenario A		Scenario B		Scenario B		Scenario D	
PREMAP-DN	Av. L.	Ac. L.	Av. L.	Ac. L.	Av. L.	Ac. L.	Av. L.	Ac. L.
3 Task/PE	4,124	222,734	3,239	171,712	3,109	183,485	2,914	177,780
4 Task /PE	3,857	208,308	3,211	170,223	2,826	166,785	2,866	174,847
5 Task /PE	3,863	208,644	3,215	170,445	2,656	156,749	2,863	174,668

Note: Av. L. means Average Latency, *Ac. L.* Accumulated Latency.

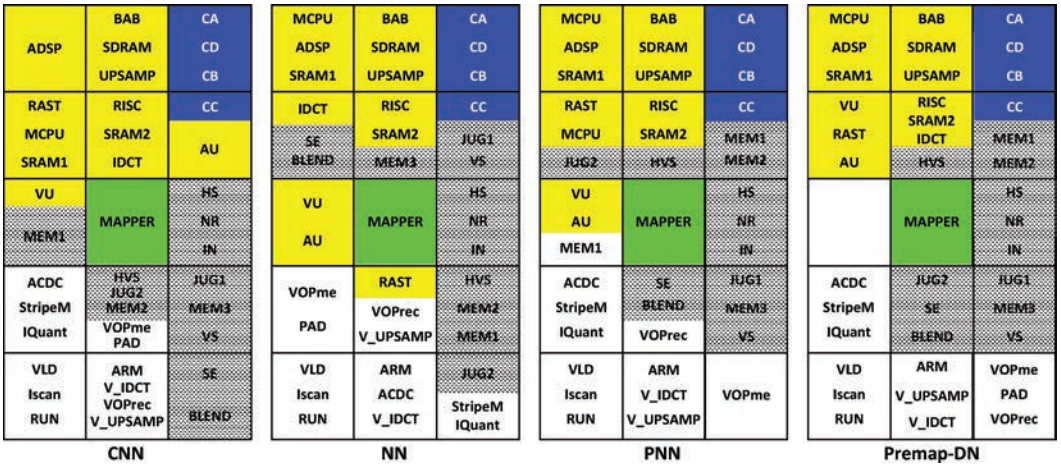

Fig. 5. Multi-task mapping in the 3x5 MPSoC, considering up to 3 tasks/PE executing simultaneously. Blue: circuit application; Yellow: MPEG4; White: VOPD; Gray: MWD. The center PE is the manager processor, which executes the *Mapper Actor*.

and in scenarios B and D the reduction was 1.9% and 1.4%, respectively, for 3 tasks/PE. Such latency reduction is converted using less power consumption when *Premap-DN* is employed, since it uses as cost function the communication energy involved in the data transfer/reception between tasks, during the mapping process. For the PNN, in scenario D (3 tasks per processor), the latency reduction was 2.9% (note that this latency reduction induced a reduction in the communication energy consumption). Table III illustrates the latency values for the *Premap-DN* heuristic.

This result was expected, since all heuristics minimize the number of hops between tasks. However, it demonstrates clearly how important is to consider not only the number of hops in the mapping heuristic, but also the communication energy, as in the proposed *Premap-DN,* to obtain optimized mappings.

5.3. Application Distribution Evaluation

The mapping quality can also be evaluated by the application distribution, as illustrated in Figure 5, corresponding to Scenario B with 3 tasks/PE.

In this scenario, PEs may receive more than 3 tasks, due to the dynamic behavior of the mapping. When a given task finishes it execution, the PE may receive a new task. As shown in Figure 5, Premap-DN groups the communicating tasks belonging to each application in continuous regions, resulting in smaller communication energy and latency compared to other heuristics. It is also important to show that the Premap-DN reduces the number of used PEs, allocating 8 PEs instead of 9. The final mapping of heuristics CNN, NN, and PNN spread some tasks of MPEG4 and MWD applications, reducing the performance of such applications.

Table IV. Energy Consumption (EC, in nJ) and Execution Time (ET, in clock cycles)

	Scenario A		Scenario B		Scenario C		Scenario D	
H	CNN	Premap-DN	CNN	Premap-DN	CNN	Premap-DN	CNN	Premap-DN
EC	9,179,970	9,221,348	9,431,786	8,848,600	7,269,261	7,326,460	12,640,634	8,803,531
ET	18,720,437	18,044,285	16,194,666	16,132,761	18,772,808	18,032,187	19,925,839	16,264,489

Note: Using CNN and Premap-DN dynamic mapping heuristics considering Scenarios A, B, C, and D, with up to three tasks per processor.

5.4. Validation of the Premap-DN in an RTL NoC-based MPSoC Platform

The proposed Premap-DN and CNN heuristics were integrated and validated in a homogeneous NoC-based MPSoC, described in synthesizable VHDL [Carara et al. 2009], named HeMPS. Both Premap-DN and the CNN were evaluated considering two performance metrics: energy consumption and execution time [Mandelli et al. 2011b].

The same 4 scenarios were used to evaluate the mapping heuristics. The multi-layer model-based approach has a module that generates C code from the UML modeled applications to the HeMPS platform [Ost et al. 2011], which uses an MPI-like API for message passing. Due to the higher execution time of the RTL simulation, the four scenarios were simulated for less than 0.2 seconds. Table IV shows the energy consumption (nJ) and the execution time of both CNN and Premap-DN.

The results reported in Table IV show that in scenarios A and C, CNN presents a small reduction in the energy consumption when compared to *Premap-DN*, 0.45% and 0.79%, respectively. In scenarios B and D, the *Premap-DN* presents a reduction in the energy consumption when compared to CNN, equal to 6.18% and 30.36% respectively. The total execution time is reduced when using *Premap-DN* in all scenarios, with a higher reduction in scenario D, 18.37%.

Scenario D penalizes the CNN heuristic due to the (i) number of tasks, 38, which restrains the search space and (ii) the automotive application having tasks with long execution times, sending and receiving a high volume of data. As mentioned previously, CNN heuristic maps new tasks in empty PEs or PEs with tasks belonging to the same application. As some tasks take longer to finish, this decreases the number of available PEs that can receive other application tasks when the CNN heuristic is employed. Thus, if there is no available PE, the CNN heuristic increases the execution time of a given application, since new application tasks will only be mapped when a PE is released.

The two last evaluations are concerned with the speedup provided by the abstract model and the accuracy of the power model. The simulation of scenario A (3 tasks/PE) for one second in the HeMPS platform (NoC modeled in VHDL and PEs modeled in cycle-accurate SystemC) took 7 hours, while the same scenario using our unified-model took on average 30 minutes (Xeon 64 bits, quadcore, 6 GB RAM). Considering that real embedded applications may run for seconds or minutes, we claim that the adoption of the present approach increases the evaluation speed, since different heuristics and application scenarios can be quickly explored. As the power model is the same for both platforms, abstract and RTL, the difference observed is less than 2%, in this case.

6. CONCLUSIONS AND FUTURE WORK

This paper presented a unified model-based framework, which employs high-level and accurate NoC models that enable rapid design space exploration. Beyond the intrinsic benefits of the modeling approach, this framework makes it possible to assess gains obtained in term of performance and power consumption resulting from the use of dynamic mapping techniques in MPSoC platforms. The unified model-based framework was used to comparatively evaluate an original mapping heuristic that

uses a pre-mapping stage to cluster multiple tasks before mapping clusters to processing elements. Results show that there exists a significant optimization potential in communication-related energy consumption that may be exploited by a range of dynamic task mapping techniques. Such techniques are of utmost importance in the frame of future multi-processor platform forecasted to exhibit hundreds of processors in the short term. In this context, our current work aims at analyzing the long-term effects of mapping decisions made in such systems, in a short time.

This work improved the quality of the dynamic mapping for NoC-based MPSoCs 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, while previous approaches consider only master-slave connections; (iii) using 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 MPSoCs. The metrics evaluated are end-to-end latency and communication energy consumption. Real and synthetic applications were evaluated. The results demonstrate that the multi-task approach reduces the energy consumption, since the approach reduces the distance among the tasks.

Future works include (i) evaluating the total MPSoC energy consumption, including the PEs consumption, (ii) evaluating strategies to decentralize the mapping, (iii) hotspots evaluation employing the rate-based power model, and (iv) the proposition of decentralized mapping, evaluating the trade-off of controlling the decentralized method per regions or applications.

REFERENCES

Almeida, G. M., Sassatelli, G., Benoit, P., Saint-Jean, N., Varyani, S., Torres, L., and Robert, M. 2009. An adaptive message passing MPSoC framework. *International Journal of Reconfigurable Computing*.

Carara, E., Oliveira, R., Calazans, N., and Moraes, F. 2009. HeMPS - A framework for NoC-based MPSoC generation. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*. 1345–1348.

Carvalho, E., Marcon, C., Calazans, N., and Moraes, F. 2009. Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs. In *Proceedings of the International Conference on System-on-chip (SoC)*. 87–90.

Chou, C-L. and Marculescu, R. 2007. Incremental run-time application mapping for homogeneous NoCs with multiple voltage Levels. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 161–166.

Chou, C-L. and Marculescu, R. 2008. User-aware dynamic task allocation in networks-on-chip. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 1232–1237.

Chou, C-L. and Marculescu, R. 2010. Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29*, 1, 78–91.

Faruque, M. A., Krist, R., and Henkel, J. 2008. ADAM: Run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of the Design Automation Conference(DAC)*. 760–765.

HA, S. 2008. Model-based programming environment of embedded software for MPSoC. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. 330–335.

Hölzenspies, P. K. F., Hurink, J. L., Kuper, J., and Smit, G. J. M. 2008. Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (MPSoC). In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 212–217.

Hosseinabady, M., and Nunez-Yanes, J. 2009. Run-time resource management in fault-tolerant network on reconfigurable chips. In *Proceedings of the Field Programmable Logic and Applications (FPL)*. 574–577.

Hu, J., and Marculescu, R. 2005. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems 24*, 4, 551–562.

Indrusiak, L. S., Thuy, A., and Glesner, M. 2007. Executable system-level specification models containing UML-based behavioral patterns. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, 301–306.

Indrusiak, L. S., Ost, L., Moraes, F. G., Maatta, S., Nurmi, J., Moller, L., and Glesner, M. 2010. Evaluating the impact of communication latency on applications running over on-chip multiprocessing platforms:

A layered approach. In *Proceedings of the International Conference on Industrial Informatics (INDIN)*. 148–153.

INDRUSIAK, L. S., AND SANTOS, O. 2011. Fast and Accurate Transaction-Level Model of a Wormhole Network-on-Chip with Priority Preemptive Virtual Channel Arbitration. In *Design, Automation and Test in Europe (DATE)*. 1–6.

JALIER, C., LATTARD, D., JERRAYA, A. A., SASSATELLLI, G., BENOIT, P., AND TORRES, L. 2010. Heterogeneous versus homogeneous MPSoC approaches for a mobile LTE modem. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 184–189.

KAHNG, A., LI, B., PEH, L., AND SAMADI, K. 2009. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 423–428.

KANGAS, T., KUKKALA, P., ORSILA, H., SALMINEN, E., HÄNNIKÄINEN, M., HÄMÄLÄINEN, T., RIIHIMÄKI, J., AND KUUSILINNA, K. 2006. UML-based multiprocessor SoC design framework. *ACM Transactions on Embedded Computing Systems 5*, 2, 281–320.

LEE, E. A. AND NEUENDORFFER, S. 2004. Actor-oriented models for codesign: Balancing re-use and performance. *Formal Methods and Models for System Design*. Kluwer Academic Publishers, Norwell, MA. 33–56.

LEE, H. G., OGRAS, U. Y., MARCULESCU, R., AND CHANG, N. 2006. Design space exploration and prototyping for on-chip multimedia applications. In *Proceedings of the Design Automation Conference (DAC)*. 1–6.

LEE S. E. AND BAGHERZADEH, N. 2009. A high level power model for Network-on-Chip (NoC) router. *Computers & Electrical Engineering 35,* 6, 837–845.

LI, K. 2010. A random-walk-based dynamic tree evolution algorithm with exponential speed of convergence to optimality on regular networks. In *Proceedings of the Conference Frontier of Computer Science and Technology (FCST)*. 80–85.

LU, S., LU, C., AND HSIUNG., P. 2010. Congestion- and energy-aware run-time mapping for tile-based network-on-chip architecture. In *Proceedings of the Frontier Computing, Theory, Technologies and Applications (FCTTA)*, 300–305.

MÄÄTTÄ, S., INDRUSIAK, L. S., OST, L., MÖLLER, L., NURMI, J., GLESNER, M., AND MORAES, F. 2008. Validation of executable application models mapped onto network-on-chip platforms. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems (SIES)*. 118–125.

MÄÄTTÄ, S., INDRUSIAK, L. S., OST, L., MÖLLER, L., NURMI, J., GLESNER, M., AND MORAES, F. 2009. Characterising embedded applications using a UML profile. In *Proceedings of the International Conference on System-on-Chip (SoC)*. 172–175.

MÄÄTTÄ, S., INDRUSIAK, L. S., OST, L., MÖLLER, L., NURMI, J., GLESNER, M., AND MORAES, F. 2010. Joint validation of application models and multi-abstraction network-on-chip platforms. *International Journal of Embedded and Real-Time Communication Systems (IJERTCS) 1*, 1, 86–101.

MANDELLI, M., OST, L., CARARA, E., GUINDANI, G., GOUVEA, T., MEDEIROS, G., AND MORAES, F. 2011. Energy-aware dynamic task mapping for NoC-based MPSoCs. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*. 1676–1679.

MANDELLI, M., AMORY, A., OST, L., AND MORAES, F. 2011b. Multi-task dynamic mapping onto NoC-based MP-SoCs. In *Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI)*. 191–196.

MARCON, C., MORENO, E. I., CALAZANS, N. L. V., AND MORAES, F. 2008. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *IET Computers and Digital Techniques 2*, 6, 471–482.

MARCULESCU, R., OGRAS, U., PEH, L., JERGER, N., AND HOSKOTE, Y. 2009. Out-standing research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28*, 1, 3–21.

MILOJEVIC, D., MONTPERRUS, L., AND VERKEST, D. 2009. Power dissipation of the network-on-chip in multi-processor system-on-chip dedicated for video coding applications. *Journal of Signal Processing Systems 57*, 2 , 139–153.

MISCHKALLA, F., HE, D, AND MUELLER, W. 2010. Closing the gap between UML-based modeling, simulation and synthesis of combined HW/SW systems. In *Proceedings of the Design, Automation and Test in Europe (DATE')*. 1201–1206.

MOLNOS, A., AMBROSE, J. A., NELSON, A., STEFAN, R., COTOFANA, S., AND GOOSSENS, K. A 2010. Composable, energy-managed, real-time MPSOC platform. In *Proceedings of the Optimization of Electrical and Electronic Equipment (OPTIM)*. 870–876.

NGOUANGA, A., SASSATELLI, G., TORRES, L., GIL, T., SOARES, A., AND SUSIN, A. 2006. A contextual re-sources use: proof of concept through the APACHES platform. In *Proceedings of the Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 42–47.

OST, L., INDRUSIAK, L. S., GUINDANI, G., REINBRECHT, C., RAUPP, T., AND F. MORAES. 2009. A high abstraction, high accuracy power estimation model for networks-on-chip. In *Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI)*. 193–198.

OST, L., GUINDANI, G., INDRUSIAK, L. S., MÄÄTTÄ, S., AND MORAES, F. 2011. Exploring NoC-based MPSoC design space with power estimation models. *IEEE Design and Test of Computers 28*, 2, 16–29.

OMG. 2011. Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems.*http://www.omg.org/spec/MARTE/1.0/PDF/*.

PIMENTEL, A. D., THOMPSON, M., POLSTRA, S., AND ERBAS, C. 2008. Calibration of abstract performance models for system-level design space exploration. *J. Sig. Proc. Syst. 50*, 2, 99–114.

SCHRANZHOFER, A., CHEN, J.-J., AND THIELE, L. 2010. Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. *IEEE Trans. Indust. Info. 6,* 4, 692–707.

SMIT, L. T., HURINK, J. L., AND SMIT, G. J. M. 2005. Run-time mapping of applications to a heterogeneous SoC. In *Proceedings of the International Symposium on System-on-Chip (SoC)*. 78–81.

SINGH, A. K., SRIKANTHAN, T., KUMAR, A., AND JIGANG, W. 2010. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *J. Syst. Archite. 56,* 7, 242–255.

WEICHSLGARTNER, A., WILDERMANN, S., AND TEICH, J. 2011. Dynamic decentralized mapping of tree-structured applications on NoC architectures. In *Proceedings of the Networks on Chip (NoC)*. 201–209.

WILDERMANN, S., ZIERMANN, T., AND TEICH, J. 2009. Run time mapping of adaptive applications onto homogeneous NoC-based reconfigurable architectures. In *Proceedings of the Field-Programmable Technology (FPT)*. 514–517.

YE, T., BENINI, L., AND DE MICHELI, G. 2002. Analysis of power consumption on switch fabrics in network routers. In *Proceedings of the Design Automation and Conference (DAC)*. 524–529.

Received March 2011; revised August, October 2011; accepted December 2011