Efficient Heuristics for Minimizing Communication Overhead in NoC-based Heterogeneous MPSoC Platforms

Amit Kumar Singh, Wu Jigang, Alok Prakash, Thambipillai Srikanthan (Senior Member, IEEE)
Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore
{amit0011, asjgwu, alok0001, astsrikan}@ ntu.edu.sg

Abstract

The number of tasks executing in MPSoC platform can exceed the available resources, requiring efficient run-time mapping strategies to meet the real-time constraints of the applications. This paper describes two new run-time mapping heuristics for mapping applications onto NoC-based Heterogeneous Multiprocessor Systems-on-Chip (MPSoC). The heuristics proposed in this paper attempt to map the tasks of an application in close proximity to each other so as to minimize the communication overhead. In addition, they have been shown to alleviate NoC congestion bottlenecks to maximize overall computation performance. Based on our evaluations to map applications with varying number of tasks onto an 8× 8 platform, we demonstrate that the new mapping heuristics are capable of reducing the total execution time, channel load and latency of applications when compared to state-of-the-art run-time mapping heuristics reported in the literature. Moreover, we show that the proposed heuristics are highly scalable and provide for high-speed realization justifying their applicability to complex MPSoC platforms.

Keywords: Mapping heuristic, multiprocessor system-on-chip (MPSoC) design, run-time mapping, NoC

1 Introduction

Single general purpose processor may be sufficient for small and less complex applications but the complexity of embedded software applications and their performance requirements have increased substantially. Thus, there is an inevitable need for high performance computing platforms. Thankfully, the significant advancements in Nanotechnology have made it feasible to integrate several embedded processors on a single chip creating a high performance multiprocessor system-on-chip (MPSoC). MPSoC is being increasingly used to meet the higher computing demands of the real world complex applications [1].

There are several issues while designing an MPSoC. The communication infrastructure is one of the important issues and it can be bus-based, point-to-point or Networks-on-Chip (NoCs)-based [2]. NoCs have several advantages over others, such as scalability and shorter wires, which minimizes power consumption. NoCs can integrate instruction set processors (ISPs), specialized processing elements like Digital Signal Processors (DSPs), FPGA fabric tiles, dedicated intellectual property cores (IPs) and specialized memories on a single chip to make a NoC-based Heterogeneous MPSoC in order to meet the ever-rising performance constraints [3] [4].

The applications can be mapped by static or run-time mapping techniques. Static mapping techniques [5] [6] cover only certain scenarios and find the best placement of tasks at design-time and hence these are not suitable for dynamic workloads. In Heterogeneous MPSoCs, task migration [3] [7] [8] is also used at run-time to improve the performance. Task migration relocates the tasks from one processing element to another processing element when a performance bottleneck is detected or when the workload needs to be distributed more homogeneously.

Embedded applications like multimedia and networking contain dynamic workload of tasks. So at any point, the number of tasks running on the MPSoC platform may exceed the available resources, requiring the tasks to be mapped at run-time to meet the real-time constraints.

This work describes two new run-time mapping heuristics based on our packing strategy and their performance evaluation for a NoC-based heterogeneous MPSoC. State-of-the-art run-time mapping heuristics do not perform well when applied to different scenarios. The new presented heuristics give better performance compared to state-of-the-art mapping heuristics. The MPSoC platform that we consider is almost similar to that described in [9]. The tasks of an application are mapped in close proximity within a particular region in order to reduce the communication overhead between the communicating tasks, to improve the performance.

The rest of the paper is organized as follows: Section 2 describes related work on task mapping. Section 3 describes the MPSoC architecture. In Section 4, we present our novel task mapping algorithms. Experimental setup and the results are presented in Section 5, with Section 6 concluding the paper.

2 Related Work

Several static (design-time) mapping techniques have been proposed to solve the problem of mapping tasks to their respective processing elements. Static mapping algo-

1074-6005/09 $25.00 © 2009 IEEE
DOI 10.1109/RSP.2009.18

rithms for NoC-based and bus-based MPSoCs are presented in [5] [6] and [10]. These mapping algorithms are not suitable for dynamic workloads.

Carvalho et al. [9] present heuristics for run-time task mapping in NoC-based heterogeneous MPSoCs. Tasks are mapped on the fly, according to the communication requests and the load in the NoC links. The performance of the mapping heuristics with dynamic workloads, targeting NoC congestion minimization to optimize the NoC performance is investigated.

Nollet et al. [11] describe a run-time task assignment heuristic for efficiently mapping the tasks in a multiprocessor systems-on-chip containing FPGA fabric tiles. With the presence of FPGA fabric tiles, algorithm is capable of managing a configuration hierarchy and this improves the task assignment success rate and quality.

Smit et al. [4] present a run-time task assignment algorithm to map the task-graph on a heterogeneous MPSoC platform. The algorithm maps a task before all other task that needs a scarce resource by taking availability of resources into account. In [12] authors have presented efficient heterogeneous multi-core architectures for streaming applications and run-time mapping of these applications onto these multi-core architectures.

Holzenspies et al. [13] present a run-time spatial mapping technique to map the streaming applications onto a heterogeneous MPSoC. The mapping technique contains four steps and is applied to a HIPERLAN/2 receiver example that takes less than 4 ms to run on an ARM926 running at 100 MHz.

Faruque et al. [14] describe run-time agent based distributed application mapping techniques for NoC-based heterogeneous MPSoCs. To map applications first a cluster negotiation algorithm is used to find the most suitable virtual cluster for each application and then a mapping algorithm is used to map the tasks of each application into their corresponding virtual clusters.

In [15] and [16] run-time mapping techniques to map the tasks onto MPSoC platforms are presented. The MPSoC platform in [15] is homogeneous (all IPs of same type) while in [16], it is heterogeneous.

In [9], five mapping heuristics, *First Free (FF), Nearest Neighbor (NN), Minimum Maximum Channel Load (MMC), Minimum Average Channel Load (MAC) and Path load (PL)* with their performance evaluation are described. Authors in [9] have combined *NN* search strategy and *PL* computation approach to find the best neighbor (*BN*) among all the nearest neighbors. For each mapping z, *PL* is computed from equation 1, where $rate_{c(i,j)}$ and $rate_{c(j,i)}$ are the rates in the individual channels, from the master to the new slave and the rates of the channels in opposite direction. This latest heuristic (*BN*) is presented in [17]. *NN* and *BN* are taken for performance comparison with our mapping heuristics.

$$\cos t_z = \sum rate_{c(i,j)} + \sum rate_{c(j,i)} \quad (1)$$

3 System Description

The MPSoC architecture model used in this work contains a set of processing nodes which interact via a communication network composed of routers (*R*) as shown in figure 1. Processing nodes may support either software or hardware task. Software tasks execute in instruction set processors (*ISPs*) and hardware tasks execute in reconfigurable logics (*reconfigurable area-RA*) or in dedicated *IPs*. The *RA* allows run-time mapping of hardware tasks through dynamic reconfiguration. The communication network uses message passing protocol for inter-task communication.

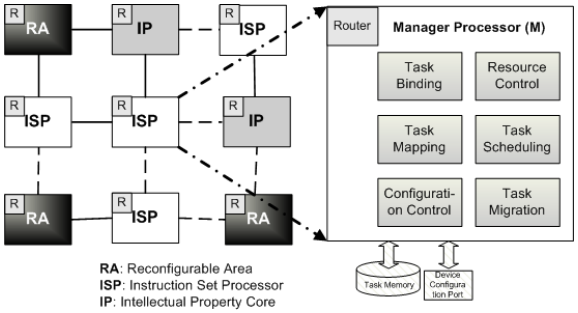


Fig 1: Conceptual Heterogeneous MPSoC Architecture

One of the processing node is used as the Manager Processor (*MP*) which is responsible for task scheduling, task binding, task placement (mapping), task migration, resource control and reconfiguration control. The *MP* starts the initial task of each application and new tasks are loaded into the MPSoC from the *task memory* when a communication to them is required and they are not already mapped.

This work focuses on *resource control*, *task binding* and *task placement (mapping)*. The *MP* takes the mapping decision according to the PE and NoC use. The resources status is updated at run-time to provide the *MP* with accurate information about the resource occupancy. There are three queues, one for each type (i.e. hardware, software and initial) of task and *task scheduling* is based on queue strategies. If there are no free resources in the system the task enters into the queue and waits until this condition changes.

4 Proposed Algorithms

This section discusses our packing strategy and two run-time task mapping heuristics based on it.

4.1 Definitions

Definitions necessary to explain our run-time mapping heuristics based on packing strategy are as follows:

Definition 1: An *application communication task graph* is an acyclic directed graph *ATG = (T, E)* as shown in figure 2 (a), where *T* is set of all tasks of an application and *E* is the set of all edges in the application. Out of all the tasks present in *T*, one task is the initial task. The initial task has no master, so it can not be the end point of any edge in *E*. All the elements present in *E* belongs to a pair of communicating tasks as a master-slave pair as in figure 2 (b). *E* is represented as $(t_{idm}, t_{ids}, (V_{ms}, R_{ms}, V_{sm}, R_{sm}))$, where t_{idm} represents the master task identifier, t_{ids} represents the slave task identifier; V_{ms} and R_{ms} are the data volumes and data rate sent from master to slave respectively; V_{sm} and R_{sm} are the data volumes and data rate sent from slave to master respectively. The message rate is described as percentage of available link bandwidth.

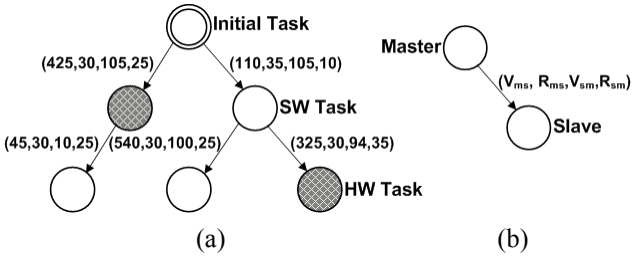


Fig 2: Application Modeling and Master-Slave pair

Definition 2: A NoC-based heterogeneous MPSoC architecture is a directed graph *AG = (P, V),* where *P* is the set of tiles p_i and $v_{i,j} \in V$ presents the physical channel between two tiles p_i and p_j. A tile $p_i \in P$ consists of a router, a network interface, a heterogeneous processing element, local memory and a cache.

Definition 3: The *application mapping* is represented by $mpng : t_i (\in T) \mid\rightarrow p_i (\in P)$ to map the tasks of the application onto the NoC-based heterogeneous MPSoC.

4.2 Initial Task Mapping

Initial tasks can only be mapped onto software processing elements as they are software tasks. The way in which initial tasks of each application are mapped has a significant impact on the performance of run-time mapping. The initial tasks can be mapped in two different ways. In one way, the initial tasks are mapped on the first free position found in the network. This may cause the initial tasks to be placed very close to each other. Now, when rest of the tasks of different applications are mapped, the applications need to share the same NoC region, resulting in longer waiting time for a resource to become free for task mapping, and increased channel congestion. In the second way, virtual clusters are found by partitioning the NoC into regions. One initial task is placed into each virtual cluster. This work considers the clustering approach.

The MP does not know the whole application graphs. It knows only the initial tasks. When initial tasks start their execution, the slave tasks are mapped dynamically, according to the communication request, sent to the MP. A run-time mapping heuristic is required to map these new tasks. In next sub-sections our packing strategy and run-time mapping heuristics based on packing are presented.

4.3 Our Packing Strategy

The state-of-the-art run-time mapping heuristics do not perform well when applied to different scenarios. The mapping heuristics (NN, BN) considered in this work for comparison are not so efficient for scenarios, where number of tasks in an application is varied. The mapping heuristics developed with our packing strategy give better performance for all the scenarios when compared with the NN and BN heuristics.

In our packing strategy, all tasks of an application are tried to be mapped close to each other within a particular region. This strategy is applied to all the applications to be mapped on to the MPSoC platform as shown in figure 3. These regions can also be called as virtual clusters. The initial task (starting task) of each application is mapped at right-top position within the virtual clusters. New incoming tasks of an application are mapped to left or down side processing element (PE) around the node (PE) making the request. If neither left nor down side PE is able to execute the requested task, only then the task is tried to be mapped on the top or right side PE. The same strategy is followed for each application.

The packing strategy tries to map each application within a particular virtual cluster with initial task positions as specified above. The strategy tries to map the communicating tasks of an application close to each other within a virtual cluster in a compact manner, in order to avoid the communication overhead between the communicating tasks.

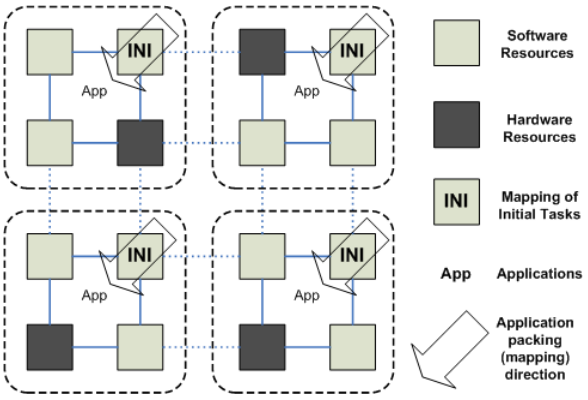


Fig 3: Initial tasks placement for mapping (packing) applications

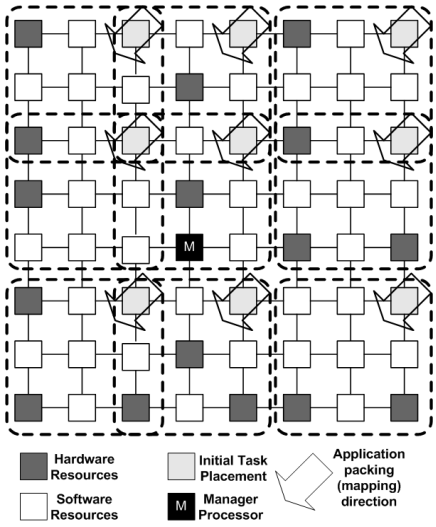


Fig 4: Our 8 x 8 NoC Model. Dashed lines denote the cluster limits

Execution time of an application depends on computation time and communication time. By packing strategy we are able to reduce the communication overhead, thus reducing the communication time and hence the execution time. *Average packet latency* depends on the distance between the source and target PEs and the congestion in the communication path. As packing strategy maps the tasks close to each other, reducing the distance between source and target PEs, resulting in reduced *latency*. *Channel load* also gets reduced as it depends on the communication overhead.

4.4 Packing based run-time Mapping Algorithms

Our run-time mapping heuristics are motivated by the packing strategy as discussed in section 4.3. The given heuristics are light-weight in terms of execution cycles, channel load and latency.

4.4.1 Algorithm 1

This algorithm is based on packing strategy along with the search space (circular search space) of *NN* heuristic. Similar to *NN* heuristic, a free node able to execute the requested task around the node making the request is searched. First the neighbors at hop distance one are searched. If node(s) at hop distance one is(are) not able to execute the requested task then nodes at hop distance two are put in the search space and so on. The search space goes on up to the NoC limit (step 6 in Algorithm 1). In our heuristic same search strategy along with the packing strategy is applied as explained in Algorithm 1.

In order to map multiple applications at a time, algorithm 1 is applied for each application. First, suitable clusters for applications are found and initial tasks are mapped as in figure 4. Then new coming tasks (requested tasks) are mapped dynamically, by applying packing strategy along with *NN* search strategy explained in algorithm 1.

The task is mapped in the same manner at each hop distances and platform resources are updated when a task gets mapped. If none of the PEs in the NoC are able to execute a task, then it is placed in its corresponding queue and waits for a resource to become free that can execute the task.

Algorithm 1: Run-time mapping

Input: *ATG(T,E)*, A*G(P,V)*
Output: *mpng* (mapping *ATG(T,E)* → A*G(P,V)*)
type (t_i): type of task (HW, SW or INI)
type(p_i): type of tile (HW, SW or INI)
NFR[type]: number of free resource(s) of type *type* in NoC
(1) Find a suitable cluster for the application (from figure 4)
(2) Map the initial task (INI) at right-top position in cluster
(3) **for** all $t_i \in T$ (except INI, already mapped)
(4) **for** all unmapped t_i that is requested
(5) **if** (*NFR[type*(t_i)*]* != 0)
(6) **for** hop_distance = 1 to NoC limit
(7) Select left and down side node(s) (near requesting node)
(8) **if** (node(s) supported)
(9) Select first free supported node $p_i \in P$ to map t_i
(10) insert(p_i to *mpng*); update(resources by *mpng*)
(11) **wait** and go back to (4) if new task is requested
(12) **else**
(13) Select right and up side node(s) (near requesting node)
(14) **if** (node(s) supported)
(15) Select first free supported node $p_i \in P$ to map t_i
(16) insert(p_i to *mpng*); update(resources by *mpng*)
(17) **wait** and go back to (4) if new task is requested
(18) **end for**
(19) **else**
(20) insert(t_i to Queue(*type(ti)*))
(21) **wait** until *NFR[type*(t_i)*]* != 0 (updated at run-time)
(22) **if** (*NFR[type*(t_i)*]* != 0)
(23) **release**(t_i from Queue(*type(ti)*)) and go back to (6)
(24) **end for**
(25) **end for**

4.4.2 Algorithm 2

This algorithm is combination of the above algorithm (Algorithm 1) and path load (*PL*) computation approach.

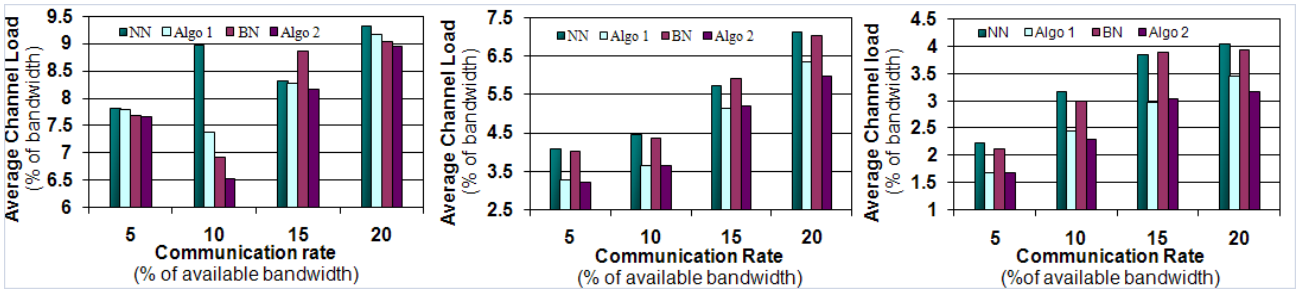
Algorithm 2: Run-time mapping

In algorithm 1, path load computation is incorporated by replacing the lines (9) and (15) both by:

- Calculate path load for node(s) (Eq. 1, Section 2)
- Select node p_i with minimum path load

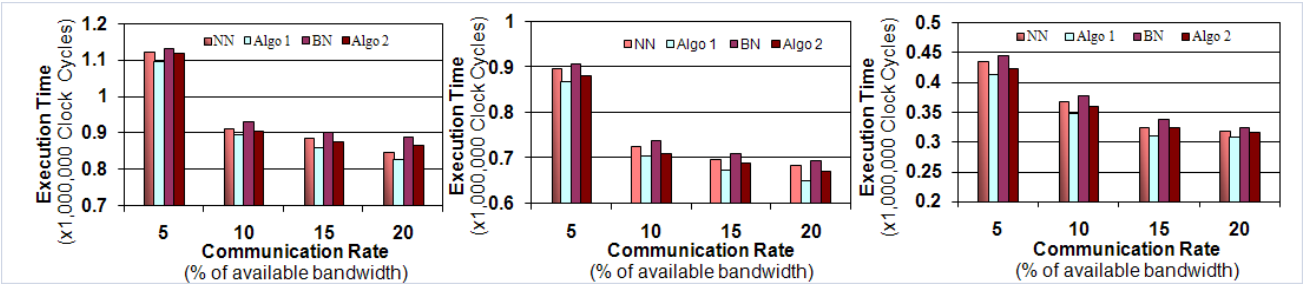
The rest of the lines remain same as in algorithm 1.

In addition to mapping the tasks in close proximity to avoid the communication overhead, this heuristic also tries to distribute the channel load by considering the path load,



(a) Applications having 10 task each (b) Applications having 7 tasks each (c) Applications having 4 tasks each

Fig 5: Average Channel Load comparison of Algorithm 1 and Algorithm 2 with NN and BN respectively



(a) Applications having 10 task each (b) Applications having 7 tasks each (c) Applications having 4 tasks each

Fig 6: Execution Time comparison of Algorithm 1 and Algorithm 2 with NN and BN respectively

resulting in reduced average channel load. Thus it is a congestion aware mapping heuristic.

5 Experiments and Results

All the experiments are performed by *ModelSim co-simulation* (*System-C* for applications and *VHDL* for the NoC). The results evaluated are *average channel load, total execution time* and *latency* of applications.

5.1 Experimental Setup

The simulation platform used for our experiments is similar to that in [9] and [17]. This section describes the experimental set up used.

All the applications are modeled as in figure 2 (a), with initial tasks, hardware tasks and software tasks. The values present on the edges represent the volume and rates of data to be sent and received by the master as explained in definition 4.1. The NoC is modeled as in figure 4 with initial tasks supported PEs at the top-right position in each cluster. Number of initial tasks may be different for different scenarios.

The experiments are performed for different scenarios. In each scenario 20 applications are taken with varying injection rate (% of available channel bandwidth). The results are shown for: (I) each application having 10 tasks (1initial, 6 software and 3 hardware tasks), (II) each application having 7 tasks (1 initial, 4 software and 2 hardware) and (III) each application having 4 tasks (1 initial, 2 software and 1 hardware tasks) simulation scenarios.

Each task transmits from 200 to 500 packets with size varying from 100 to 400 16-bit flits. The task processing time is fixed.

5.2 Experimental Results

Results obtained from our proposed heuristics are compared with the state-of-the-art run-time mapping heuristics.

5.2.1 Channel Load

The average channel load represents the NoC use. In our mapping heuristics, first heuristic (based on NN) does not consider traffic during mapping, but explores the proximity of communicating tasks. In second heuristic (based on BN) we consider the traffic as well during mapping, thus trying to distribute the channel load more uniformly. As we packed the communicating tasks as close as possible, hence they don't interfere with the channel of the other applications' tasks, resulting in reduced average channel load.

Graphs in figure 5 show that algorithm 1 and 2 present less average channel load as compared to NN and BN heuristics respectively. It can be observed that when number of tasks in each application is reduced, NN and BN do not perform well. In first scenario ((a)Applications having 10 tasks) algorithm1 and algorithm2 reduce the channel load by 2.74% and 3.8%, in second scenario (b) by 14.88% and 15.84% and in third scenario (c) by 21.60% and 22.01% when compared with NN and BN heuristics respectively. Thus, our heuristics performs better when number of tasks

in each application is less because of better packing of tasks.

5.2.2 Total Execution Time

Total execution time for each task comprises of communication time and computation time. The allocation time (the time to find the placement (mapping time) and configuration time) is indirectly considered. Communication time dominates the entire execution time. Since with the packing strategy tasks are mapped in close proximity, so communication time gets reduced and so does the total execution time. It has also been seen that mapping time gets reduced because we have reduced the search space (explained in 4.4) to find the placement of a task. The graphs in figure 6 show that the total execution time also gets reduced for our algorithms (algo 1 and algo 2) when compared to the NN and BN heuristics.

5.2.3 Latency

The average packet latency depends on the distance between the source and destination PEs on which communicating tasks are mapped and the congestion in the communication path. Network congestion depends directly on the communication rate (% of available bandwidth). Table 1 presents the latency results for different heuristics for the first scenario. It can be seen that average packet latency for our algorithms gets reduced compared to NN and BN.

Scenarios	Rates	Average Packet Latency (Clock Cycles)			
		NN	Algo 1	BN	Algo 2
Applications having 10 tasks	5%	142	138	144	139
	10%	248	239	233	224
	15%	332	323	344	332
	20%	447	435	438	423

Table 1- Average Packet Latency measured in clock cycles

6 Conclusions

This paper details our packing strategy and two run-time mapping heuristics based on it in order to map the applications efficiently onto an 8 × 8 NoC-based heterogeneous MPSoC. First heuristic tries to map the tasks of an application in close proximity, reducing the communication overhead (communication time) between the communicating tasks. The second heuristic considers traffic in addition to the proximity of tasks while mapping, resulting in more uniformly distributed channel load. Our mapping heuristics reduce the average channel load by a large amount with a significant improvement in total execution time and latency, when compared to the mapping heuristics presented in [9] [17]. The improvements are clearly enunciated in the experiments and results section. As mentioned before the processors in this work can execute only one task at a time, so in future, we plan to extend the mapping heuristics to multi-tasking processors.

7 Acknowledgments

We thank Mr. Ewerson Carvalho (first author of the papers [9] and [17]) for providing us the simulation environment and helping us in explaining details via mails.

References

[1] Jerraya, A.; et al. Guest Editors' Introduction: Multiprocessor Systems-on-Chips. IEEE Computer, v.38(7), 2005.

[2] Benini, L. and Micheli, G. Networks on Chips: A new SoC paradigm. IEEE Computer, v.35(1), 2002.

[3] Nollet, V.; et al. Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles. DATE, 2005.

[4] Smit, L.; et al. Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. FPL, 2004.

[5] Hu, J.; Marculescu, R. Energy- and Performance-Aware Mapping for Regular NoC Architectures. IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, v.24(4), 2005.

[6] Marcon, C.; et al. Time and Energy Efficient Mapping of Embedded Applications onto NoCs. ASP-DAC, 2005

[7] Bertozzi, S.; et al. Supporting task migration in multiprocessor systems-on-chip; a feasibility study, DATE, 2006.

[8] Kalte, H.; et al. Context Saving and Restoring for Multitask ing in Reconfigurable Systems. FPL, 2005.

[9] Carvalho, E.; et al. Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs. Rapid System Prototyping (RSP), 2007.

[10] Ruggiero, M.; et al. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-onchip. DATE, 2006.

[11] Nollet, V.; et al. Run-time Management of a MPSoC Con taining FPGA Fabric Tiles. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 1, 2008.

[12] Smit, G.; et al. Multi-coreArchitectures and Streaming Applcations. SLIP, 2008.

[13] Holzenspies, P.; et al. Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSoC). DATE, 2008.

[14] Faruque, M. A. A.; et al. ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communica tion. DAC, 2008.

[15] Chou, C.-L. and Marculescu, R. Incremental run-time appli cation mapping for homogeneous NoCs with multiple voltage levels. Haraware/software Codesign and system synthesis (CODES+ISSS'07), 2007.

[16] Lei, T. and Kumar, S. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. Digital Systems Design (DSD), 2003.

[17] Carvalho, E.; Moraes, F. Congestion-aware task mapping in heterogeneous MPSoCs. System-on-Chip (SoC), 2008