DynMapNoCSIM : A Dynamic Mapping SIMULATOR for Network on Chip based MPSoC

[1, 3] Mohammed Kamel Benhaoua, [2]AmitKumar Singh, [1]Abou El Hassan Benyamina, [3]Pierre Boulet
[1] Department of Computer Science, University of Oran, LAPECI, Algeria
[2] Department of Computer Science, University of York, UK
[3] University Lille1, LIFL, CNRS, UMR 8022, F-59650Villeneuve d'Ascq, France
{Mohammed-Kamel.Benhaoua, pierre.boulet}@liû.fr, amit.singh@york.ac.uk, benyamina.abouelhassen@univ-oran.dz

JDIM
Journal of Digital Information Management

ABSTRACT: *To fulfill the need of intensive embedded computations, architects have proposed Network-on-Chip based Multi-Processor Systems-on-Chip. Applications exploit the distinct features of the different types of processors in Multi-Processor Systems-on-Chip to optimize the performance metrics: overall execution time, energy consumption, resource usage, etc. Application designers often map statically the tasks on the processors. This static mapping cannot handle many kinds of applications such as those with dynamic workloads and one must use dynamic mapping when several applications run concurrently. As such, there has been an increased need for defining and developing simulation software for carrying out a simulation of the proposed dynamic Mapping heuristics to the NoC-based MPSoC architectures. In this paper, we present DynMapNoCSIM, a JAVA based Dynamic Mapping simulator for NoC-based MPSoC architecture, which builds upon the object-oriented modular design of the NoC-based MPSoC architecture components. Here we demonstrate the use of our proposed simulator by providing a conceptual detail. We have also presented all the features offered. Finally, we have validated the outputs of DynMapNoCSIM with the studies of existing and proposed Dynamic mapping strategies.*

Categories and Subject Descriptors
C.1.2 [Multiple Data Stream Architectures]; (Multiprocessors)]: **B.4 [Input/Output and Data Communications]:** chip carriers

General Terms: Mapping Simulator, Multiprocessors, Network Mapping

Keywords: Multi-processor Systems-on-chip (MP-SoCs), Network-on-chip (NoC), Heterogeneous Archi-Tectures, Dynamic Mapping Simulation

Received: 20 October 2014, Revised 27 November 2014, Accepted 30 November 2014

1. Introduction

The complexity of applications demands to transit from the System-on-Chip (SoC) based on a single processor to Multi-Processor System-on-Chip (MPSoC) which contains multiple processing elements (PEs) in the same chip. Eventually, the evolution of semiconductor technology permits us to integrate several processors in the same chip. Typically, there are two types of MPSoCs: homogenous and heterogeneous. A homogeneous MPSoC contains identical PEs [22], [23], whereas different types of PEs are integrated in a heterogeneous MPSoC [24], [25]. MPSoCs provide increased parallelism towards achieving high performance [21]. The Network-on-Chip (NoC) has been introduced as a power efficient and scalable interconnection to support communication amongst the PEs [1].

Modern embedded applications contain dynamic workload of tasks or applications that need to be loaded into the system at run-time, which needs efficient dynamic mapping techniques [9], [10], [11], [12], [13], [14], [15]. Such techniques find the placement of tasks on the MPSoC resources at run-time. The latest dynamic mapping approaches try to place the communicating tasks on nearest available PEs, i.e. close to each other in order

to reduce the communication overhead [26], [18], [19], [3]. The objective of this paper is to design and implement a simulation tool which allows to test different techniques of dynamic mapping and simulate traffic on a NoC. The simulator is developed at a high level of abstraction of the behavior of an MPSoC system that uses a network on chip for communication. This level can simulate the behavior of the mapping and execution application tasks. Then the Simulator based on computational models of architecture and application. The application is a set of tasks that communicate with each other, where all actions related to tasks and communications are known. Architecture based on a model which defines the different features and types of processing elements. The advantage of the simulator is that it will easily integrate algorithmic proposals and have the desired performance metrics, which are so far the execution time and energy consumption. The simulator supports two different type of platform: mono-tasks and multi-tasks platform.

The rest of the paper is organized as follows. Section 2 provides an overview of the related work. Section 3 describes the simulator detailed design. Experimental setup and the results are presented in Section 4. Section 5 concludes the paper and provides future research directions.

2. Related Work

Most of the existing work reported in the literature to solve the problem of mapping on MPSoC platform is static mapping techniques [3], [4], [5], [6], [7], [8], [34]. However, static mapping is not able to handle dynamic workload of tasks or applications that need to be loaded into the MPSoC at run-time. Dynamic (run-time) mapping techniques are required to handle the mapping of such workloads into the platform resources. The latest works reported in the literature handle the problem of run-time mapping of applications tasks onto NoC-based MPSoCs while optimizing for different performance metrics.

Mehran et al [12] propose a Dynamic Spiral Mapping (DSM) technique for task mapping during run-time. Faruque et al. [20] propose a decentralized agent-based mapping approach targeting large NoC-based heterogeneous MPSoCs such as 32x64 systems. Carvalho et al. [14], [18] present heuristics for dynamic task mapping in two phases. The first phase finds placement of initial (starting) tasks of different applications in the MPSoC architecture, whereas the second phase uses different methods. In [18], the authors evaluate dynamic mapping heuristics and compare them with static mapping techniques such as simulated annealing and Taboo search. Singh et al. [28], [3] target heterogeneous MPSoC architecture containing software and hardware PEs. Their mapping heuristics map the communicating tasks of an application close to each other so as to minimize the communication overhead in order to improve the overall performance. In general, the works proposed in [14] and [18] are extended in [28] and [3] by employing a packing strategy that minimizes the communication overhead in NoC-based MPSoC platform. This entire works are validated by a cycle accurate tools. Our motivation is to develop a high level simulator tool to accelerate a conception flow.

Table 1 shows the popular simulators reported in the literature. All these simulators don't permit us to simulate our proposed heuristics for dynamic mapping applications in NoC- based heterogeneous MPSoCs.

GPNoCSim	2006	University of Bengladesh	Mesh2D, Tore, Fat tree	Java
BOOKSIM	2010	University of Stanford	Mesh2D, Tore, Fat tree	C++
Noxim	2010	University of Catagne	Mesh2D	C++
NS-2	1995	Lawrence Berkley National Laboratory	Mesh2D, Tore, Fat tree	C++
Darsim	2009	MIT	Mesh 2D, Mesh 3D	C++

Table 1. Simulators Synthesis

3. Detailed Design

To understand the features provided by the simulation platform, a Unified Modeling Language (UML) is presented.

Hereinafter, a static view of the components of the simulator and a global class diagram that explains the various interactions between these components are shown.

3.1 Packages Platform

The platform was performed on five key features, namely:

- Creating a NoC.
- Creating applications.
- Mapping of applications on the NoC.
- Routing communications.
- NoC simulation result and display performance.

Different packages providing these features are:

- **Package Architecture**

This package allows you to generate the architecture of a network on chip with 2D mesh topology. It is composed of CreateNOC class. The latter creates and describes all the tiles and physical links. Extended version generates a multi-tasks platform, cad each PE can execute multi-tasks by implementing a scheduler in each PE.

• **Package Application**
The Application package supports different applications from an XML file that will later be placed on the architecture. An application is described by a graph consisting of tasks and communication links between the different tasks.

• **Package Dynamic Mapping**
It contains different dynamic mapping heuristics for the placement of applications on a heterogeneous architecture. It contains a reference heuristics dynamic mapping and all our proposed heuristics [29], [30], [31], [32], [33], [34].

• **Package Routing**
Routing package allows routing communications between tasks. This package contains the routing algorithms, namely the static methods such as XY and dynamic as our dynamic routing algorithm MORA [32].

• **Package Simulation**
This package is responsible for simulating the mapping of the various tasks and routing their communications and all applications on the architecture created. The simulator class is the kernel of the simulator, it allows managing the various actions of a simulation.

• **Package Results**
This package has the function to display performance measures desired by the simulation.

3.1.1 Class Diagram
The diagram below Figure 2 presents the different classes of the simulator and the interactions between them.

3.2 Creating The Architecture
The proposed simulator allows to design a heterogeneous NoC architecture Mesh-2D. This architecture consists of several types of processors, ie GP, ASIC, FPGA, DSP. These can execute single or multi-tasking, it is called a Mono/Multi tasking. Figure 1 shows an example of an architecture that can be generated by the simulator. The architecture creates is divided into clusters.

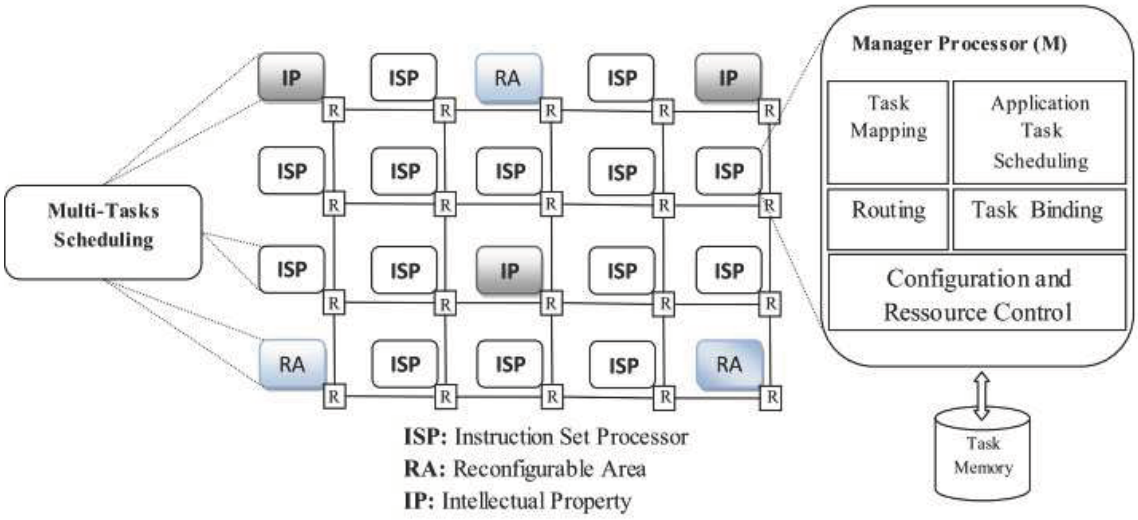


Figure 1. Heterogeneous MPSoC Architecture

• **Manager:**
One of the (GPs) processors of the NOC is used as a manager processor. This allows the control and centralized management of NOC. The main features provided by the manager are:

- **Tasks Mapping:**
Search the PE in the architecture following a mapping heuristic to allocate a task,

-**Tasks Scheduling:**
Defines the order of execution of tasks.

-**Routing communications:**
Determines which route to take between two communicating tasks (master/slave).

-**Update of the platform:**
Updated in real time tiles and communications links.

3.2.1 Class Diagram Package Architecture
The following diagram Figure 3 illustrates the different classes and characteristics that make up the Architecture package.

• **ProcessorElement:**
The *ProcessorElement* class represents processors. Each processor is characterized by:

- Id: Identifier of the processor,

- type: This argument represents the type of processor (GP, FPGA, ASIC, DSP, ..),

- x,y: are the coordinates of the line and column in the NOC,

- state: variable that determines whether the processor is available or not,

- memory: memory of a processor,

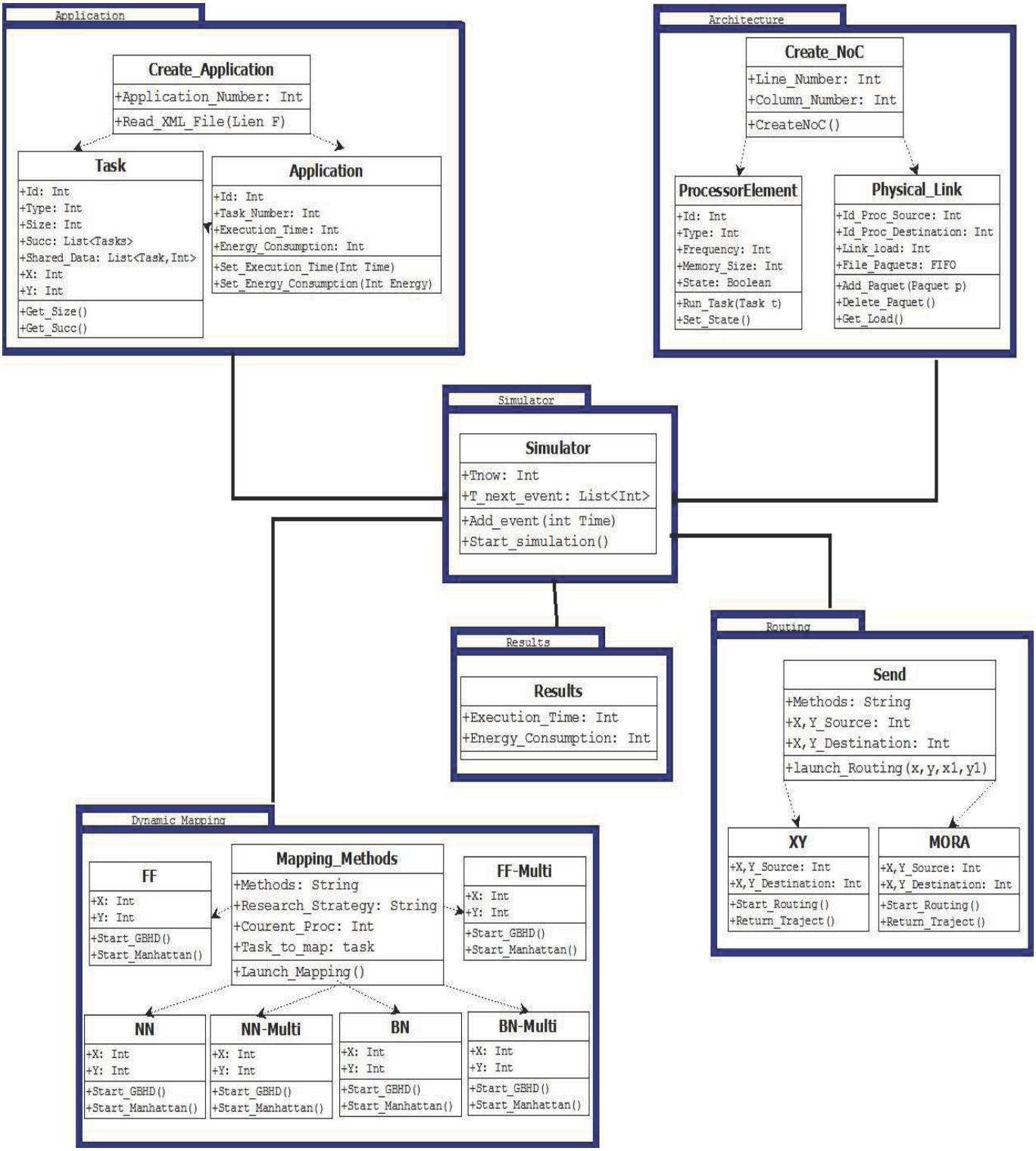


Figure 2. Heterogeneous MPSoC Architecture

- **Frequency:** is the number of clock cycles per unit of time (second, ms, ect),

- **Energy:** is the energy consumed during the execution of a clock cycle (in units of energy "joul, mj, ect")

- **Mode:** variation in the frequency of execution.

• **PhysicalLink:** a physical link has the following arguments:

- **Id:** link identifier,

- **Id_proc_source:** Id of the source processor,

-**Id_proc_destination:** Id of the destination processor,

-**Load:** represents the size of the data must pass through this link.

3.3 Creating Applications

An application is represented by oriented task graph $TG = (T, L)$, where T is the set of tasks of an application and L

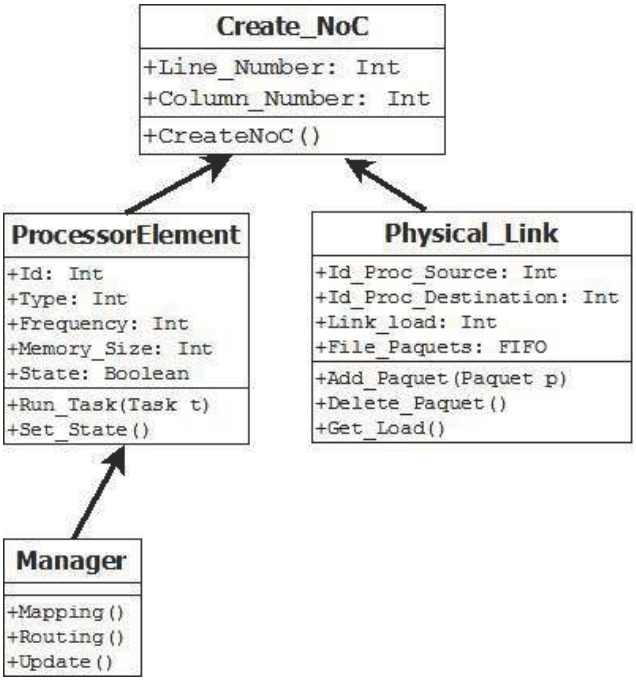


Figure 3. Class Diagram Package Architecture

set of links that connect the tasks in this application (Figure 4. (a)). Each application has a single initial task and several software and hardware tasks. The connection between two tasks is a master-slave connection (Figure 4.(b)). Beginning the task of the application is the initial task, and has no master. The links contain respectively data shared between each master-slave.

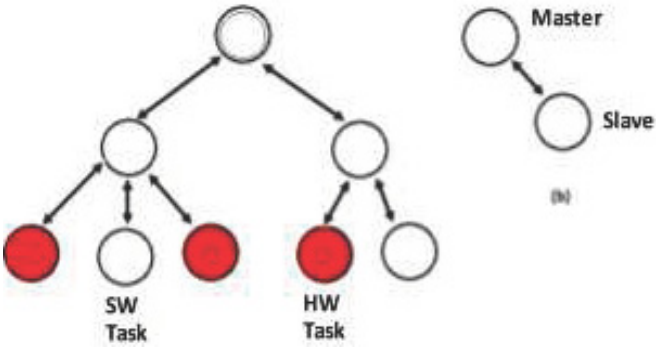
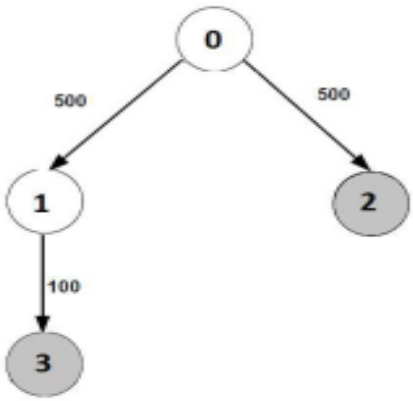


Figure 4. Class Diagram Package Architecture




```
<?xml version = "1.0" encoding = "iso-
8859-1" ?>
<Application_List>
<Application>
   <Task>
     <id> 0 </id>
      <Type_Size>
         <size_by_type Type = "1" Size =
                        "300" />
      </ Type_Size >
            <succ>
           <id_succ id = "1" data = "500"
                                        />
           <id_succ id = "2" data = "500"
                                        />
            </succ>
</Task>
<Task>
          <id> 1 </id>
          < Type_Size >
          <size_by_type Type = "1" Size =
                                "300" />
          </ Type_Size >
             <succ>
            <id_succ id="3" data = "100"/
                                        >
          </succ>
</Task>
<Task>
           <id> 2 </id>
           < Type_Size >
              <size_by_type Type = "2"
                          Size = "300" />
           </ Type_Size >
                 <succ>
                 </succ>
</Task>
<Task>
           <id> 3 </id>
           < Type_Size >
               <size_by_type Type = "2"
                         Taille = "300"/>
           </ Type_Size >
             <succ>
             </succ>
</Task>
</Application>
```

Figure 5. Example of application

3.3.1 Representation of Applications

The list of applications is represented by a given input XML file. Figure 5 shows an application and its representation in XML.

3.3.2 Class Diagram Application Package

The following diagram Figure 6 shows the different classes in the package application.

· **Application:** The main attributes of the application are:

- **TasksList:** the list of tasks that make up the application,

- **ExecutionTime:** the overall execution time of the application,

- **EnergyConsumption:** the energy consumed during its execution.

- **Task:** Each task is associated with the following attributes: **Id:** Identifier of the task, **type:** the type of processor on which the task can be executed (GP, FPGA, etc.), **size:** the size of the task by type (number of unit data "*byte, bit* .."),

- **cycles:** the required number of cycles in the execution of the task depending on the type,

- **Shared-data:** the size of shared data with each slave (in unit data),

- **x, y:** the coordinates of the processor on which the task will be placed.

3.3.3 Sequence Diagram:

The sequence diagram, Figure 7 illustrates the interaction between the master and slave tasks.

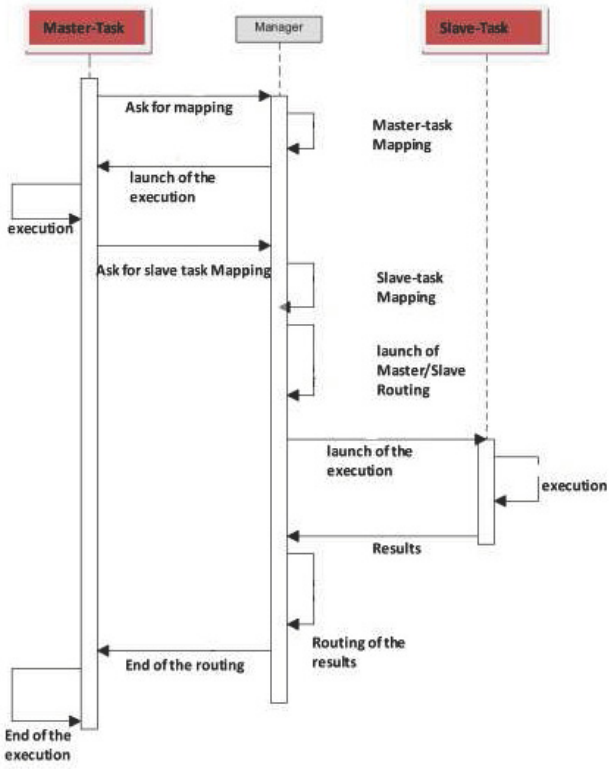


Figure 7. Sequence Diagram illustrating the interaction between master/slave

3.4 Simulation Model

In this section we explain the interest and the concept of simulation is by definition an imitation or representation of a system by another. As such, several aspects of the simulation are simplifications of the real system or from deduction made from studies on similar systems.

When a system is simulated, it is represented as a set of modules which are interconnected to each other. The manner in which these modules are executed and interact with each other is called a model of simulation. Among the simulation models can be distinguished, discrete event simulation and simulation with fixed step.

The simulator which we present in this paper is based on discrete event approach which is described in the following section.

3.4.1 The Discrete Event Simulation:

In discrete event simulation, only the points where the system state changes over time are represented. In other words, the system is modeled as a sequence of events, that is to say the instants at which the system state changes occur. Events such as the arrival of entities, the service start of an entity, a recovery service for a resource where each product at a time. In discrete event simulation, there is a global queue which stores the list of events that will occur. The advantage of this technique is down to the fact that the system is seen as being composed of a collection of events, each event running instantly with respect to the simulation clock, changing the state of object modeled, selecting and scheduling the successor event that will happen in the future. The major drawback of this approach is that the programmer must explicitly

Event	Status Change	Event programmed
Arrival of applications	Add applications	Mapping initial tasks
Mapping initial tasks	Making the cluster occupied Making the middle PE of the cluster occupied	Start task execution
Start task execution	Making PE occupied	End of execution
End of execution	Release the PE	Mapping slaves tasks
Mapping slaves tasks	Assigned the PE	Start routing
Start routing	Edit the load links	End routing
End routing	Edit the load links	Start execution of the slave task

Table 2. Lists of Events

Time	Event
0	Arrived applications 1 and 2 Static mapping of initial tasks of applications 1 and 2 Started on the two initial tasks
400	Request to map the slave task of application 1 Dynamic mapping of the slave task
500	Placement of the slave task (application 1) finish Start routing (master to slave)
600	Request to map the slave task of application 2 Dynamic mapping of the slave task
700	Mapping of the slave task completed of the application 2 Start routing (master to slave)
3100	End routing between master and slave task (application 1)Start execution of the slave task
3300	End routing between master and slave task (application 2)Start execution of the slave task
3600	End of execution of slave task (application 1)Start routing (from slave to master)
3700	End of execution of slave task (application 2)Start routing (from slave to master)
4100	End routing between the slave and his master task (application 1) Resume execution of the master task
4300	End routing between the slave and his master task (application 2) Resume execution of the master task
5000	End of execution of the initial task (master task) (application1) End of execution of application 1
5300	End of execution of the initial task (master task) (application2) End of execution of application 2

Table 3. Example of a running of the simulation

partition a model of events, which makes the structure of the model difficult to understand.

3.4.2 The Basic Components Of The Simulation Model:

Whatever the simulated system must include three main components:

- **Entities:** are objects with specific attributes that affect change in the system state,

- **Queues:** Entities generally expect in a queue to be served,

- **Resources:** Treat entities queue. These resources are free when they are available for the treatment or occupiedwhen processing entities.

By analogy in our system, these components are represented by:

- Applications

- Queuing applications,

- All PEs and physical links.

3.4.3 The list of events simulator:

Table 2 lists some events, change of state as well as scheduled events that are managed by the simulator. To better understand the sequence of events during the simulation, the Table 3 shows an example of the simulation of the mapping of two applications in parallel. Each of these two applications is composed of two tasks (initial task and the slave task)

3.4.4 Initialization Execution Parameters :

This initialization is done through a text file that contains the following parameters:

3.4.5 Performance Measures

The simulator which we implemented can simulate a several dynamic mapping techniques on a NOC architecture to measure and compare the performance of

these techniques in terms of execution time and energy consumption.

5. Experimental setup and the results

This work was realized with java us tool language. Figure 8 show the graphical interfaces simulator. Phase 1 is the settings interface. It contains all the possible input parameters. The second phase how to configure the architecture (size of the platform, number of PEs, position of the different PEs, type of the PEs ...etc). Phase 3 is for creating all applications graphs needed. A phase 4 is the interface that displays the graphs results measurement in term of energy consumption and execution time. Finally the phase 5 and 6 shows the details of the simulation and Gant graph respectively.

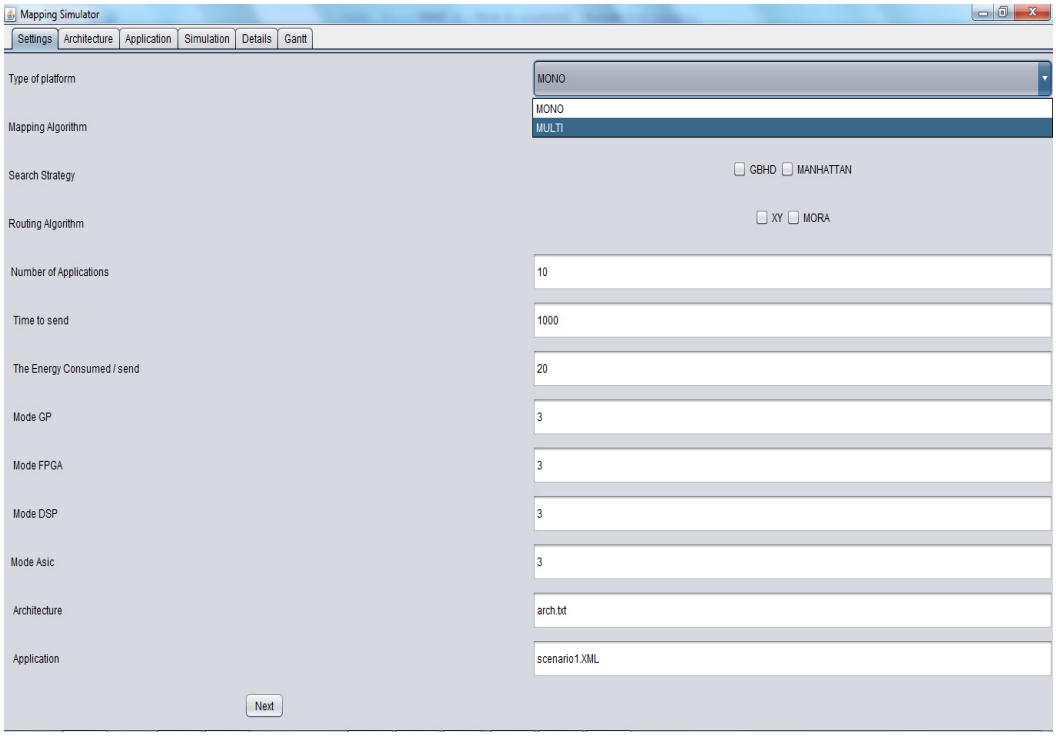


Figure 8. Graphical Interfaces of the Simulator

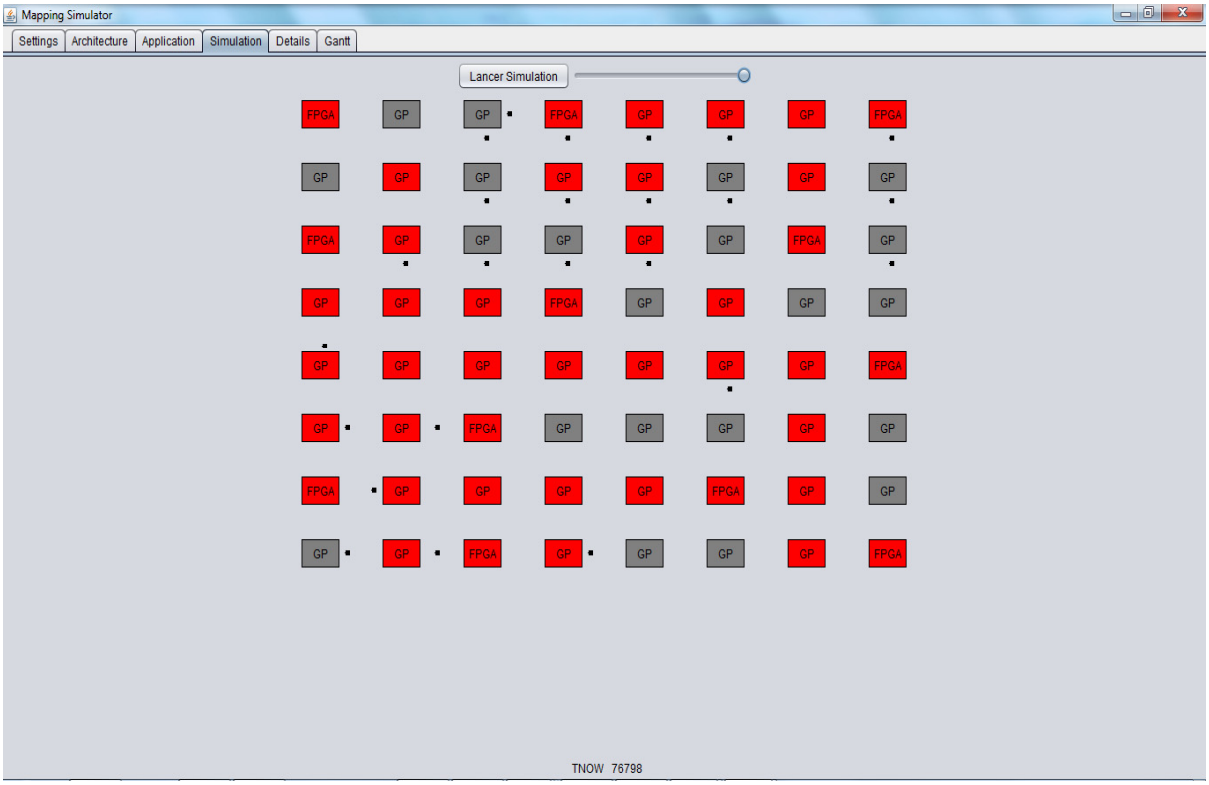


Figure 9. Graphical Interfaces of An Example Of Running Simulation

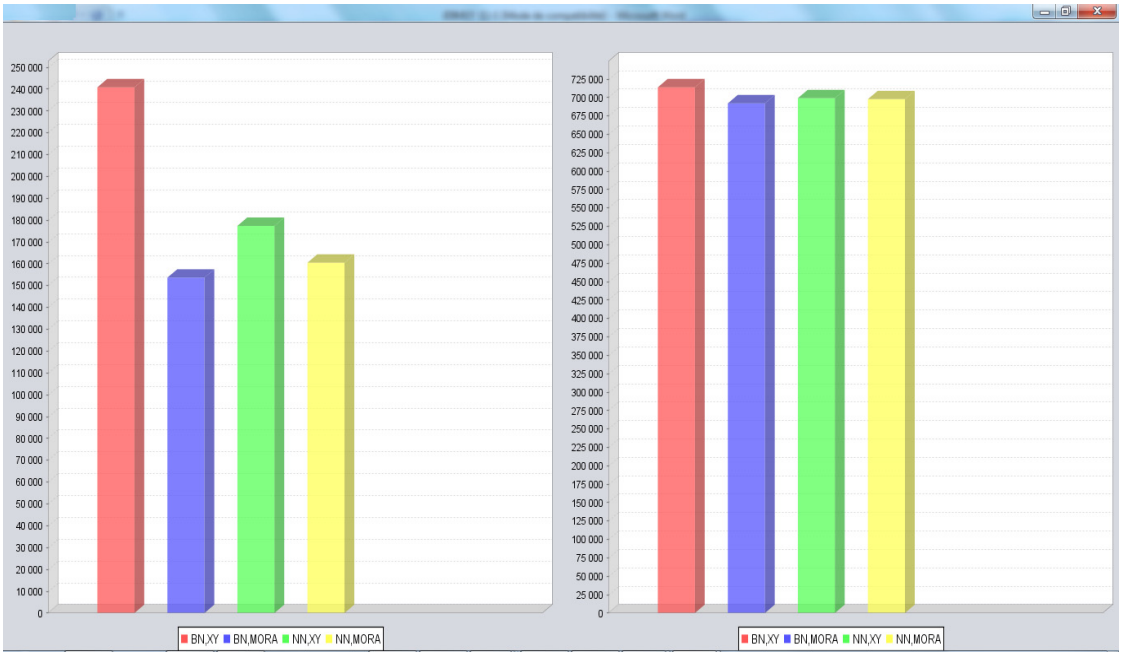


Figure 10. Graphical Interfaces of displays results of an example of running simulation

Figure 9 show an example of runtime tasks mapping of an example. We can view the occupation of the PEs and the transfer of the packets between the PEs in the links in all the platform resources. We can have at runtime the display results of the measurement performance needed such us execution time and energy consumption. This results are shown in figure 10.

5. Conclusion

The lack of tools to perform the dynamic placement of applications on heterogeneous MPSoCs based NoC platforms, simulate traffic and evaluate the performance of systems based on this structure pushed us to develop a new simulator.

This paper is dedicated to the design details of various features of this new simulator and some tests that were presented at the end of the paper. However, due to its object-oriented design, this tool is easily extensible and can be enhanced and complemented by other features being integrated in the design phase of MPSoCs based NoCs..

References

[1] Benini, L., Mecheli, G. D. (2002). *Networks on chips: a new SoC paradigm, Computer*, 35 (1) 70–78.

[2] Bertozzi, D., Benini, L. (2004). A network-on-chip architecture for gigascale systems-on-chip, *Circuits and Systems Magazine, IEEE*.

[3] Singh, A. K., et al. (2010). Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms, *JSA* p. 242–255.

[4] Zhang, Y., et al. (2002). Task scheduling and voltage selection for energy minimization, in *DAC*.

[5] Shin, D., Kim, J. (2004). Power-aware communication optimization for networks-on-chips with voltage scalable links, *In:* CODES.

[6] Vardi, F., et al. (2009). Crinkle: A heuristic mapping algorithm for network on chip, *IEICE Electronics Express*, p, 1737–1744.

[7] Carvalho., al. (2009). Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs, in *SoC*.

[8] Smit., al. (2005). Run-time mapping of applications to a heterogeneous SoC, *In: SoC*.

[9] Holzenspies et al. (2007). Mapping streaming applications on a reconfigurable MPSoC platform at run-time, *In: SoC*.

[10] Chou, CL., et al. (2007). Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels, in *CODES*.

[11] Chou, CL., Marculescu, R. (2008). User-aware dynamic task allocation in networks-on-chip, *In: DATE*.

[12] Mehran, A., et al. (2008). DSM: A heuristic dynamic spiral mapping algorithm for network on chip, *IEICE Electronics*, p. 5–13.

[13] Marcelo, et al., (2011). Multi-task dynamic mapping onto NoC-based MPSoCs, *In: SBCCI*.

[14] Carvalho, E., Moraes, F. (2008). Congestion-aware task mapping in heterogeneous MPSoCs, *In: SoC*.

[15] Wildermann, S., et al. (2009). Run time mapping of adaptive applications onto homogeneous NoC-based reconfigurable architectures, *In: FPL*.

[16] Holzenspies et al., (2008). Run-time spatial mapping

of streaming applications to a heterogeneous multi-processor system-on-chip (MPSOC) *In*: *DATE*.

[17] Schranzhofer, A., et al. (2010). Dynamic and adaptive allocation of applications on MPSoC platforms, *ASP-DAC*.

[18] Carvalho, E., Calazans, N., Moraes, F. (2010). Dynamic task mapping for MPSoCs *IEEE Design Test of Computers*, p. 26–35.

[19] Singh, A. K., et al. (2009). Eficient heuristics for minimizing communication overhead in NoC-based heterogeneous MPSoC platforms, in *RSP*.

[20] Faruque, M., et al. (2008). Adam: Run-time agent-based distributed application mapping for on-chip communication, *In DAC*.

[21] Jerraya, A., et al. (2005). Guest editors' introduction: multiprocessor systems-on-chips, Computer 38 (7) 36–40.

[22] Kumar, A., et al. (2008). Multiprocessor systems synthesis for multiple use-cases of multiple applications on fpga, ACM ToDAES.

[23] Bertozzi, D., Benini, L. (2004). Xpipes: a network-on-chip architecture for gigascale systems-on-chip, *Circ. Syst. Mag. IEEE 4* (2) 18–31.

[24] Smit, L., et al. (2004). Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture, *In*: *FPT*.

[25] Kistler, M., et al. (2006). Cell multiprocessor communication network: built for speed, *IEEE Micro* 26 (3) 10–23.

[26] Carvalho, E., et al. (2007). Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs, in *RSP*.

[27] Sahu, P. K., Chattopadhyay, S., (2013). A survey on application mapping strategies for Network-on-Chip design, in *JSA*.

[28] Singh, A. K., et al. (2009). Mapping algorithms for NoC-based heterogeneous MPSoC platforms, in *DSD*.

[29] Benhaoua, M. K., et al. (2014). Heuristic for Accelerating Run-Time Task Mapping in NoC-based Heterogeneous MPSoCs,*Journal of Digital Information Management*.

[30] Benhaoua, M. K., et al., (2013). Heuristics For Dynamic Task And Communications Mapping In Noc-Based Heterogeneous Mpsocs, *The Mediterranean Journal of Computers and Networks,* 9 (4) *October* 2013.

[31] Benhaoua, M. K., et al. (2013). Heuristics for Routing and Spiral Run-time Task Mapping in NoC-based Heterogeneous MPSOCs, *IJCSI International Journal of Computer Science Issues,* 10 (4), No 1.

[32] Benhaoua, M. K., et al. (2014). Multi-Objective Routing Algorithm for dynamic communications mapping in NoC-based heterogeneous MPSoCs, in *META'2014 International Conference on Metaheuristics and Nature Inspired Computing, Oct. 27-31, 2014 in Marrakech, Morocco.*

[33] Benhaoua, M. K., et al., (2014). Heuristic for Accelerating Run-Time Task Mapping in NoC-based Heterogeneous MPSoCs, in *ICESIT 2014: International Conference on Embedded Systems and Intelligent Technology, Nov 25-26, 2014 in Dubai, UAE.*

[34] Benyamina, A., Boulet, P., Benhaoua, M. K. (2015). Static and Dynamic Mapping Heuristics for Multiprocessor Systems-on-Chip, *Computing in Research and Development in Africa, Springer International Publishing*, 229-247.