Dynamic communications mapping in multi-tasks NoC-based heterogeneous MPSoCs platform

Mohammed Kamel Benhaoua*

Department of Computer Science,
Faculty of Engineering,
University of Oran1, Algeria
Email: kbenhaoua@gmail.com
*Corresponding author

Amit Kumar Singh

Department of Computer Science,
Faculty of Engineering,
University of York, UK
Email: amit.singh@york.ac.uk

Abstract: Multi-processor system-on-chip (MPSoC) has emerged as a solution to address the increased computational requirements of modern applications. The network-on-chip (NoC) has been introduced as a power-efficient and scalable communication infrastructure between processors. One important phase in architectural exploration in NoC-based MPSoC is the communications mapping. Mapping parallelised communications of tasks onto these MPSoCs can be done either by static or dynamic routing. Static communication mapping strategies find the fixed placement of communications like XY routing and hence, these are not suitable to achieve high overall performance. The number of tasks or applications executing in MPSoC platform can exceed the available resources, requiring multi-tasking platform. In this paper, we propose a newly dynamic communications mapping strategy for efficient communication between the PEs of MPSoC, where each PE support multiple tasks and shared memory is used for the communications between the tasks mapped in the same PE. The strategy considers efficient placement of communications in order to optimise the overall performance. Experimental results show that the proposed mapping approach provides significant performance improvements when compared to those using static routing.

Keywords: multi-processor systems-on-chip; MPSoCs; network-on-chip; NoC; heterogeneous architectures; dynamic mapping heuristics; routing algorithm; communications mapping; multi-tasks platform.

Reference to this paper should be made as follows: Benhaoua, M.K. and Singh, A.K. (2015) 'Dynamic communications mapping in multi-tasks NoC-based heterogeneous MPSoCs platform', *Int. J. High Performance Systems Architecture*, Vol. 5, No. 4, pp.240–251.

Biographical notes: Mohammed Kamel Benhaoua received his Engineer degree in Artificial Intelligence and Magister degree in Information Security and Networking from Oran University Computer Science Department, Algeria, in 2005 and 2009, respectively. He received his PhD degree from Lille1 University, France and Oran1 University, Algeria. His research interests include NoC-based MPSoC design, parallel processing, optimisation, design space exploration (DSE) and run-time mapping techniques for MPSoC.

Amit Kumar Singh received his BTech in Electronics Engineering from Indian School of Mines, Dhanbad, India, in 2006. Thereafter, he worked with HCL Technologies, India for a year and a half. He joined Nanyang Technological University (NTU), Singapore, in 2008 and worked at Centre for High Performance Embedded Systems (CHiPES), School of Computer Engineering, NTU, Singapore as a research student towards the completion of his PhD till January 2012. From February 2012 to August 2014, he was working with the Department of Electrical and Computer Engineering, National University of Singapore (NUS) as a Postdoctoral Researcher. Since September 2014, he has been working with Department of Computer Science, University of York, UK. His research interests include 2D and 3D network-on-chip (NoC)-based multiprocessor systems-on-chip (MPSoC), design space exploration (DSE) and run-time mapping techniques for MPSoC. He has published over 35 papers in leading related international journals/conferences.

1 Introduction

Intensive embedded systems use multi-processor systems-on-chip (MPSoCs), which provide increased parallelism towards achieving high performance (Jerraya et al., 2005), to cope with the limits of a single general purpose processor, increasing computational demands and performance requirements. An MPSoC (Singh et al., 2013; Bertozzi and Benini, 2004; Smit et al., 2004) contains multiple processing elements (PEs) in the same chip. The network-on-chip (NoC) has been introduced as a power efficient and scalable interconnection to support communication amongst the PEs (Benini and Mecheli, 2002; Moraes et al., 2004).

The designer has to map the tasks of the application onto the different processing resources of the MPSoC. Static mapping techniques define task placement at design-time, having a global view of the MPSoC resources. Such mapping techniques may use complex algorithms to better explore the MPSoC resources towards achieving optimised solutions (Zhang et al., 2002; Shin and Kim, 2004; Armin et al., 2007). However, static mapping is not able to handle the dynamic workload of tasks or applications that need to be loaded into the system at run-time. Dynamic (run-time) mapping techniques are required to handle these varying (dynamic) workloads (Chou et al., 2007; Chou and Marculescu, 2008; Mehran et al., 2008; Carvalho and Moraes, 2008; Wildermann et al., 2009). Such techniques find placement of tasks on the MPSoC resources at run-time. The latest dynamic mapping approaches try to place the communicating tasks on the nearest available PEs, i.e., close to each other in order to reduce the communication overhead (Carvalho et al., 2010; Singh et al., 2010). However, these approaches do not perform well when applications contain a large number of tasks that exceed the available resources. A multi-tasks platform is needed to solve the deadlock in Benhaoua et al. (2013a, 2013b, 2014a, 2014b, 2014c) with a well clusterisation. Further, most of the mapping works reported in the literature uses a deterministic routing method (Singh et al., 2010; Mehran et al., 2008; Wildermann et al., 2009; Carvalho et al., 2010; Holzenspies et al., 2008; Faruque et al., 2008; Manna et al., 2012; Silva et al., 2012). However, for a system that needs to handle dynamic workflow, using a dynamic routing method can lead to better results.

We present a dynamic communication mapping algorithm that reduces the communication costs by using the benefits of multi-tasking platform. Multi-tasking means that each PE can execute many tasks depending in memory capacities. A multi-tasks scheduler is implemented in each PE. The model used for the representation of applications is the master-slave model. This type of model is used to represent the applications that have parallel communicating tasks. The considered heterogeneous MPSoC platform contains two types of PEs: instruction set processors (ISPs) and reconfigurable areas (RAs), which execute software and hardware tasks, respectively. In the MPSoC, each PE supports multiple tasks. Existing techniques use static routing approaches to facilitate the communication. However, most of them do not focus on the adaptive routing (dynamic communications mapping). The proposed approach work in two cases: if the communication is between two different PEs then the method tries to find the path of communications that has the lowest load (widest bandwidth). Otherwise, a shared memory is used to communicate between intra-tasks (multi-tasks platform) resulting in optimised execution time and energy consumption. The obtained results show further improvements when compared to existing approaches.

The rest of the paper is organised as follows. Section 2 provides an overview of related work. Section 3 describes the model of considered MPSoC architecture. In Section 4, the proposed approach has been presented. Experimental setup and the results are presented in Section 5. Section 6 concludes the paper and provides future research directions.

2 Related work

Mapping of tasks and communications into the multi-tasks MPSoC platform requires finding the placement of tasks and communications into the platform resources in view of some optimisation criterions like reducing energy consumption, reducing total execution time and optimising occupancy of channels. Mapping can be accomplished by static (design-time) or dynamic (run-time) mapping techniques (Singh et al., 2013). Most of the existing works reported in the literature to solve the problem of mapping on MPSoC platform are static mapping techniques (Zhang et al., 2002; Shin and Kim, 2004; Vardi et al., 2009; Wang et al., 2010; Ghosh et al., 2009; Sahu and Chattopadhyay, 2013). Meta-heuristics like genetic approach (Lei and Kumar, 2003; Wu et al., 2003) and methods like tabu search (Manolache et al., 2005; Murali et al., 2006) and stimulated annealing (Marcon et al., 2005; Orsila et al., 2007) are presented. These techniques find fixed placement of tasks at design-time with a well-known computation and communication behaviour. 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 optimising for different performance metrics.

Wildermann et al. (2009) evaluate the benefit of using a run-time mapping heuristic, which decreases the communication overhead. A neighbourhood cost function has been used to reduce the communication costs. Holzenspies et al. (2008) investigate another run-time spatial mapping technique to map streaming applications onto heterogeneous MPSoCs, aiming at reducing the energy consumption. Schranzhofer et al. (2010) suggest a dynamic mapping strategy based on pre-computed template mappings (defined at design-time), which are used to define

placement of newly arriving tasks to the PEs at run-time. Chou et al. (2007) use a NoC platform with multiple voltage levels. Their mapping technique is based on a region selection algorithm that minimises the communication energy consumption. The communication energy is decreased by 50%. The mapping technique is applicable to homogenous NoC platforms. In Chou and Marculescu (2008), the authors incorporate the user behaviour information in the resource allocation process. This allows the system to respond better to real-time changes and adapt to user needs dynamically. Ost et al. (2013) propose a dynamic power aware mapping technique that minimises energy consumption by 16% in the best case. Mehran et al. (2008) propose a dynamic spiral mapping (DSM) technique for task mapping during run-time. The placement of a task is searched in a spiral path from centre to the boundary of the network architecture so that the communicating tasks can be placed close to each other. It also attempts to reduce the execution time by reducing dynamic mapping time, reconfiguration time and task migration time.

Faruque et al. (2008) propose a decentralised agent-based mapping approach targeting large NoC-based heterogeneous MPSoCs such as 32 × 64 systems. The proposed heuristic maps the applications in a decentralised manner using an agent-based approach. Multiple agents are used to perform resource management. There are two types of agents: global agent (GA) and cluster agents (CAs). The whole platform is partitioned into small clusters and each CA has updated knowledge of its cluster resources. The GA keeps global information about all the clusters. The agents negotiate with each other to find PEs suitable for mapping a task. The agent-based mapping reduces monitoring traffic and computational effort for the mapping process, compared to the centralised approaches.

Carvalho and Moraes (2008) 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 (e.g., first free – FF, nearest neighbour – NN, minimum maximum channel load – MAC and path load – PL) to find the placement for rest of the tasks on the fly according to the communication requests and the loads in the NoC links. NoC channel load, congestion and packet latency gets reduced when employing different methods. The NoC-based target MPSoC architecture contains PEs each supporting a single task. In Carvalho et al. (2010), the authors evaluate dynamic mapping heuristics and compare them with static mapping techniques such as simulated annealing and taboo search.

Singh et al. (2010) target heterogeneous MPSoC architecture containing software and hardware PEs. In the architecture, among the available processing nodes, one processing node acts as a manager processor that is responsible for task binding, task mapping, task migration, resource control and reconfiguration control. The resource status is updated at run-time and the manager processor keeps track of the information about resource occupancy. Their mapping heuristics map the communicating tasks of an application close to each other so as to minimise the communication overhead in order to improve the overall performance. The heuristics in Singh et al. (2010) examine the available resources prior to recommending the adjacent tasks on the same PE. The mapping process is accomplished in two phases. First, initial tasks are mapped at the centre of the clusters that are obtained by partitioning the NoC into regions. Thereafter, the communicating tasks are requested and mapped by their proposed mapping approaches. In general, the works proposed in Carvalho and Moraes (2008) are extended in Singh et al. (2010) by employing a packing strategy that minimises the communication overhead in NoC-based MPSoC platform. The heuristics in Singh et al. (2010) are further extended in (Kaushik et al., 2011) to make them both the computation and communication aware.

An energy-aware heuristic for dynamic task mapping, named lower energy consumption based on dependencies-neighbourhood (LEC-DN) has been presented in Mandelli et al. (2011). The main cost function here is not only the distance in hops between communicating tasks, but also the proximity in the number of hops and the communication volume among the tasks, since the number of transmitted flits defines the communication energy. When target task has only one communicating task that has already been mapped, LEC-DN uses the NN search in a spiral fashion. On the other hand, if there are more than one communicating tasks that are already mapped, it searches for a PE inside the bounding box defined by the position of such task depending on the communication volume. In Weichslgartner et al. (2011), a dynamic decentralised application-driven and resource-aware mapping has been proposed, where tasks can be embedded incrementally with an already mapped predecessor task. This is a self-embedding approach that is fully decentralised and autonomous. The major contributions in the work of Maqsood et al. (2015) are summarised as follows:

- A detailed quantitative analysis of the selected dynamic task mapping heuristics is provided under same environment, using same assumptions, and system models (Singh et al., 2010).
- An extension to CPNN (Singh et al., 2010) heuristic is proposed. The proposed heuristic aims to reduce the communication cost and energy consumption by migrating communicating tasks that are already mapped using CPNN from lightly loaded PEs to the other PEs that can accommodate those tasks.
- Formal verification and modelling of the proposed technique is provided using high level Petri nets, satisfiability modulo theories, and Z3 solver.

Most of the existing works use a static routing algorithm such as XY method (Singh et al., 2010; Mehran et al., 2008; Mandelli et al., 2011; Holzenspies et al., 2008; Carvalho et al., 2010; Faruque et al., 2008).

Benhaoua et al. (2014a) in their paper have proposed a new packing strategy to find free resources for run-time

mapping of application tasks on NoC-based heterogeneous MPSoCs.

However, for a system that has a dynamic workflow, using a dynamic routing method can lead to significant performance improvements. Benhaoua et al. (2014c) has proposed a dynamic multi-objective routing algorithm working in mono-tasks platform. Some reference heuristics in the literature has been implemented that employ XY routing algorithm, and these heuristics have been used with our newly proposed dynamic multi-objective routing algorithm. In Benhaoua et al. (2014c), the results obtained reduce significantly execution time and energy consumption. The latest works reported in the literature use a multi-tasks platform. In this paper, we propose a dynamic communications mapping in multi-tasks NoC-based heterogeneous MPSoC platform.

3 Mapping problem and reference mapping heuristics

3.1 Application task graph

An application task graph is represented as an acyclic directed graph $TG = (T, E)$, where T is set of all tasks of an application and E is the set of all edges in the application. Figure 1(a) describes an application having initial, software and hardware tasks along with the edges E connecting these tasks. Figure 1(b) shows the master-slave pair (communicating tasks). The starting task of an application is the initial task that has no master. Each task is associated with following attributes: task identifier t_{id}, task type t_{type} (hardware, software, initial), and task execution time t_{exec} on supported PE types. Edge set E contains all the edges along with the communicating tasks connected by the edges. Each edge Figure 1(b) has following attributes: it's master task identifier *mtid* representing the connected master task, slave task identifier st_{id} representing the connected slave task, the data volume sent from master to slave V_{ms}, and data volume sent from slave to master V_{sm}. If there are multiple slave tasks communicating with a master task, the slave tasks are requested to be mapped in the order of their assigned task identifier number. These slave task identifiers are assigned at design-time to optimise performance, based on the communication overhead, connections (edges) between the master and slave tasks and memory capacities of each PEs. For example, let a master task identifier is 0 and its four communicating tasks' identifiers are 1, 2, 3 and 4, then first the task with identifier number 1 gets requested. We look for highlight all the requested tasks and mapping them into the same PE or in different PEs. To transmit and receive messages by a task, our routing algorithm chooses the shortest trajectory and PL that disposes a low load from one node to another node in the MPSoC architecture.

3.2 NoC-based heterogeneous MPSoC architecture graph

Figure 2 shows the model of the multi-tasks heterogeneous MPSoC architecture used in this work. The architecture contains a set of different multi-tasks PEs that interact via a communication network (Benini and Mecheli, 2002). The PEs can be of varying types such as ISPs, reconfigurable logics (RA), dedicated intellectual properties (IPs), etc. Each PE integrates multi-tasks scheduling. Tasks to be executed onto the PEs are categorised as software and hardware tasks, which normally implement simple and compute intensive functions, respectively. Software tasks execute in ISPs and hardware tasks execute in RAs or dedicated IPs. ISPs execute software tasks efficiently. Induction of RAs in the platform provides flexibility to hardware at a similar level to the ISPs programmability. However, higher reconfiguration overheads of RAs need to be taken into account. The communication network required to facilitate communication amongst PEs is arranged in a 2D mesh topology (Carvalho and Moraes, 2008), as shown in Figure 2. Network communication protocol follows wormhole packet switching, handshake control flow, input buffers and deterministic XY routing algorithm. In XY routing, the packets are first transferred in X-direction and then in Y-direction in order to transfer them from the source PE to the destination PE.

Figure 1 Application task graph modelling and master-slave pair

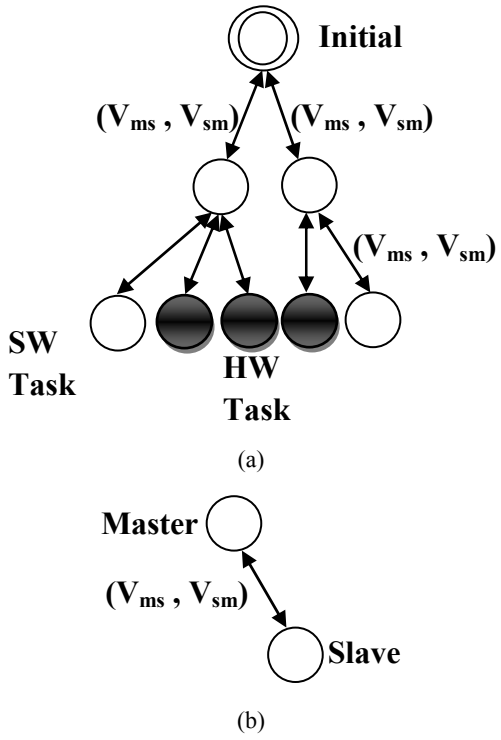
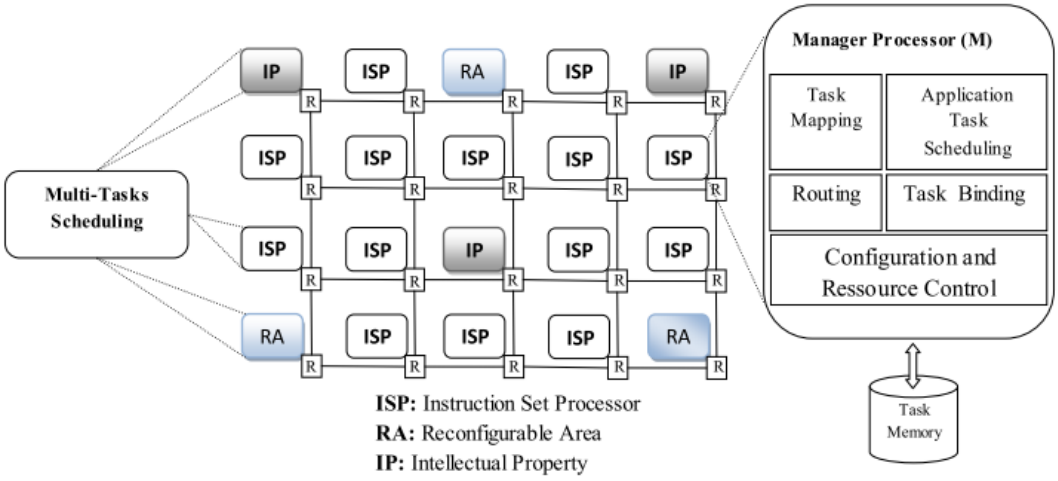


Figure 2 Multi-tasks heterogeneous MPSoC architecture (see online version for colours)



In addition to XY routing, our dynamic routing algorithm has also been incorporated, which will be detailed in the next section. The inter-task communication is supported by a message passing mechanism similar to the one used in Carvalho and Moraes (2008).

A multi-tasks NoC-based heterogeneous MPSoC architecture is a directed graph $AG = (P, V)$, where P is the set of tiles and V represents the physical channels between the tiles. Each tile in P has following attributes: the tile identifier p_{id}, the tile address p_{add} that is used to receive packets sent from some other tile, the tile type p_{type} (hardware, software, initial). Each physical channel keeps the channel width information in packets and percentage usage of available bandwidth in order to facilitate efficient transmission of data.

3.3 Reference mapping heuristics

We will use the mapping heuristics FF, NN and best neighbour (BN) proposed in Carvalho et al. (2010) and Singh et al. (2010) as the reference heuristics to be used for the comparative study.

3.3.1 NN heuristic

The NN mapping algorithm assigns the first task to the selected initial PE. It considers only the proximity of an available resource to execute a given task. Later, the requested task is mapped to the available PEs that have the minimum distance to the already mapped PEs, with reference to initial position. The distance is calculated based on the hop count starting from the hop count of 0 and going up to the maximum hop distance (hop distance = 0 to max hop count).

3.3.2 BN heuristic

The NN heuristic considers only the proximity of an available resource to execute a given task. The BN heuristic combines the search strategy of NN with the path load (PL) computation approach. Unlike NN that stops the evaluation when the first free supported PE is found, BN evaluates all the free supported PEs at each hop distance. For all the supported PEs, their imposed path loads in the channels used for communications are computed and the PE with minimum PL is chosen for final allocation (mapping) in order to get the best neighbor from the available neighbors. The channel (link) that needs to transfer minimum number of packets is the one having minimum PL. If a mapping is found with the PEs in the current hop distance, then the evaluation process is stopped for higher hop distances. The other steps of BN heuristic are similar to that of NN heuristic.

3.3.3 FF heuristic

FF is the simplest mapping algorithm. For task placement, the algorithm looks for the first available PE and assigns the task to it. The tasks are mapped in a sequential order, starting from the initial position of (0, 0) which represents 0th row and 0th column in the mesh NoC. The algorithm keeps on looking for the available PEs until the position reaches the boundary of NoC. If no PE is found, then it looks for the PEs in the next row. In every iteration, the mapping is completed only if PE is found or if all the PEs have been evaluated (Maqsood et al., 2015).

3.3.4 PL heuristic

Computes the load in each channel used in the communication path. PL computes the cost of the communication path between the source task and each one of the available resources. The selected mapping is the one with minimum cost.

4 Proposed dynamic communication mapping algorithm in multi-tasks NoC-based heterogeneous MPSoCs platform

The reference heuristics including most of the existing dynamic task mapping approaches (e.g., Carvalho et al.,

2010; Singh et al., 2010; Carvalho and Moraes, 2008) use static XY communication mapping to facilitate communication amongst the communicating tasks once they are mapped onto the PEs otherwise in the same PE. Example of such a routing is shown in Figure 3(a). The figure shows an example of tasks mapping into NoC. Each PE can execute more than one task. If the tasks are mapped in the same PE, a shared memory is used for the communication. Otherwise, NoC links are used where two communicating tasks are mapped on different PEs (source and destination) and they need to communicate with each other. The values mentioned adjacent to the links represent the volumes present in the links, i.e., the number of packets to be transmitted through the links. Figure 3(a) indicates that in order to transfer a token from the source PE (PE that execute the tasks number 9) to the destination PE (PE that execute the tasks number 10), the packet is first transferred 2 hop distances in X direction and then 2 hop distances in Y direction while following the XY routing mechanism. The packets are sent one by one in the same direction created by the first packet. In Figure 3(a), the first chosen link in X direction has volume of (150) that is more than the volume (110) present in Y direction. Similarly, the second chosen link in the X direction has more volume than that of the link in Y direction (250 vs. 80). This mechanism routes the packets through a path that incurs high communication costs due to high volumes present in the links chosen for communication, resulting in high communication costs. Thus, choosing such communication paths may incur high communication time and energy consumption. In order to provide efficient communication between the source and destination nodes, an efficient routing strategy needs to be developed. The routing strategy should be able to choose the links with lower volumes at run-time. Figure 3(b) describes an example for the operation of the proposed dynamic communication mapping algorithm presented in Algorithm 1. Unlike the static communication mapping, our proposed dynamic communication mapping chooses an efficient routing path where the packets are transferred by the links having the lowest loads. The direction to be taken from source to destination PE follows different paths depending upon the location of the PEs and loads in the paths. If x-coordinate of the source X_{source} is less than the x-coordinate of the destination X_{dest}, then the trajectory (path) will be *up* to *down*; otherwise *down* to *up*. For down to up, if y-coordinate of the source Y_{source} is less than the y-coordinate of the destination Y_{dest}, then the path will be *left* to *right*, else *right* to *left*. For all the different paths, the algorithm chooses the link direction that has the lowest load. For example, Algorithm 2 shows how the lowest loaded link is found in the case of *Up*_to_*Down*-*Left*_to_*Right*. Depending upon the load values present in the links, the algorithm chooses left to right ($X' = X_{source}$, $Y' = Y_{source+1}$) or *up* to *down* link, which has lower loads. Similar approach as that of Algorithm 2 is followed for other cases when *up*, *down*, *left* and *right* are contained in the calling function. In the case when Y_{source} and Y_{dest} are the same (i.e., in the same column), the direction is *up* to *down* or *down* to *up* and there is no evaluation to get the load values on the link. The direction is automatically taken in one of the two directions. Similarly, if X_{source} and X_{dest} are the same (i.e., in the same row) then the link chosen and the direction is *left* to *right* or *right* to *left*. This kind of links selection towards the destination PE facilitates to choose the lowest loaded links. Once a chosen link becomes more loaded, another less loaded link is chosen for the packet transmission if the source and destination PE are not in the same row or column. Otherwise, the same link gets used. For all the communicating tasks, the packets to be transferred use the same strategy.

Algorithm 1 Dynamic communication mapping algorithm

Input: X_{source}, Y_{source}, X_{dest}, Y_{dest}
Output: X', Y'

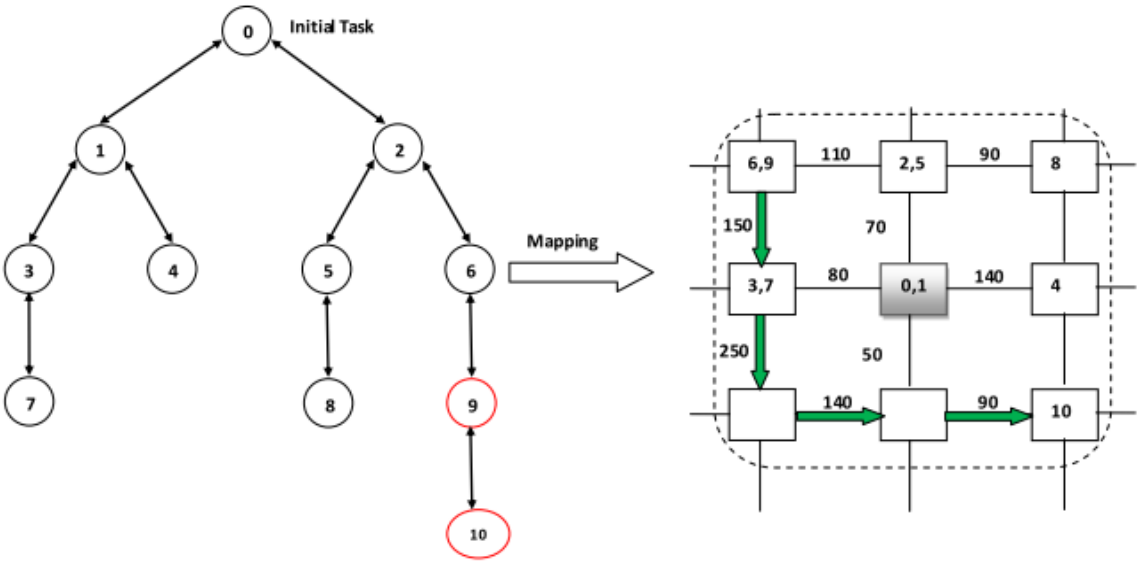
```
if X_source < X_dest then //Up to Down
    if Y_source < Y_dest then
        Up_to_Down-Left_to_Right(X_source, Y_source)
    else
        Up_to_Down-Right_to_Left(X_source, Y_source)
    end if
else // Down to Up
    if Y_source < Y_dest then
        Down_to_Up-Left_to_Right(X_source, Y_source)
    else
        Down_to_Up-Right_to_Left(X_source, Y_source)
    end if
end if
if X_source = X_dest then //in the same row
    Right_to_Left-Left_to_Right(X_source, Y_source)
end if
if Y_source = Y_dest then // in the same column
    Up_to_Down-Down_to_Up(X_source, Y_source)
end if
```

Algorithm 2 Up_to_Down-Left_to_Right

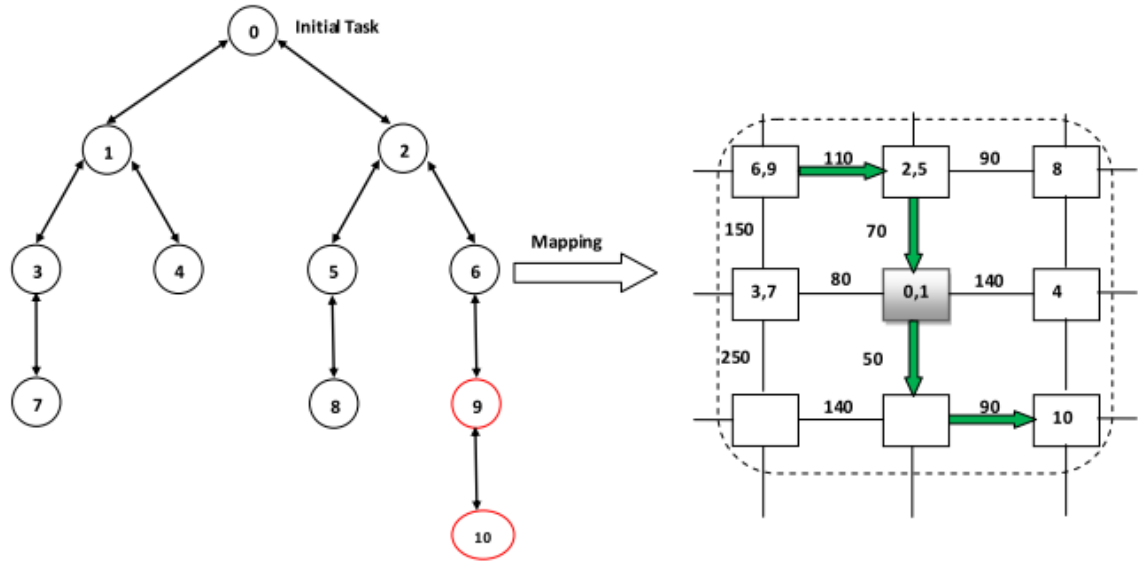
Input: X_{source}, Y_{source}
Output: X', Y'

```
if get_value_Link(X_source, Y_source + 1) <
   get_value_Link(X_source + 1, Y_source) then
    X' ← X_source
    Y' ← Y_source + 1 //Left to Right
else
    X' ← X_source + 1 //Up to Down
    Y' ← Y_source
end if
```

Figure 3 Static and dynamic communication mapping in multi-tasks heterogeneous MPSoC architecture, (a) static communication mapping (b) proposed dynamic communication mapping (see online version for colours)



(a)



(b)

4.1 Computing overall execution time and energy consumption

In this subsection, we describe the mathematical formulas used for the computation of energy consumption and execution time.

4.1.1 Energy consumption computation

The ways for calculating various energy consumption values are introduced subsequently.

Energy consumption for software tasks (in the same software resource). With multitasking the resource can execute more than one task depends in memory capacities:

$$TEC_{tn}^{s} = \sum_{i=1}^{n} EC_{inst}^{s} * T_{st}^{inst} \quad (1)$$

We calculate de energy consumption by instruction for each software instruction $(EC_{inst}^{s} * T_{st}^{inst})$ for all the software tasks (TEC_{tn}^{s}) mapped in the same software resource.

Energy consumption for hardware tasks (in the same hardware resource). With multitasking the resource can execute more than one task depends in memory capacities:

$$TEC_{tn}^{h} = \sum_{i=1}^{n} EC_{inst}^{h} * T_{ht}^{inst} \quad (2)$$

We calculate de energy consumption by instruction for each hardware instruction $(EC_{inst}^{h} * T_{ht}^{inst})$ for all the hardware tasks (TEC_{tn}^{h}) mapped in the same hardware resource.

If the communicating tasks are not mapped in the same resource, then the links of the NoC are used for the

communication. Energy consumption for one packet sending from t_i to t_j with rate R:

$$EC^{R}_{s^{tj}_{ti}} = EC^{R}_{s} * \frac{Q^{i}_{j}}{R} \tag{3}$$

Energy consumption for waiting data in the link for sending data from t_i to t_j:

$$EC_{w} = EC^{d}_{w} * \frac{Q^{i}_{j}}{R} \tag{4}$$

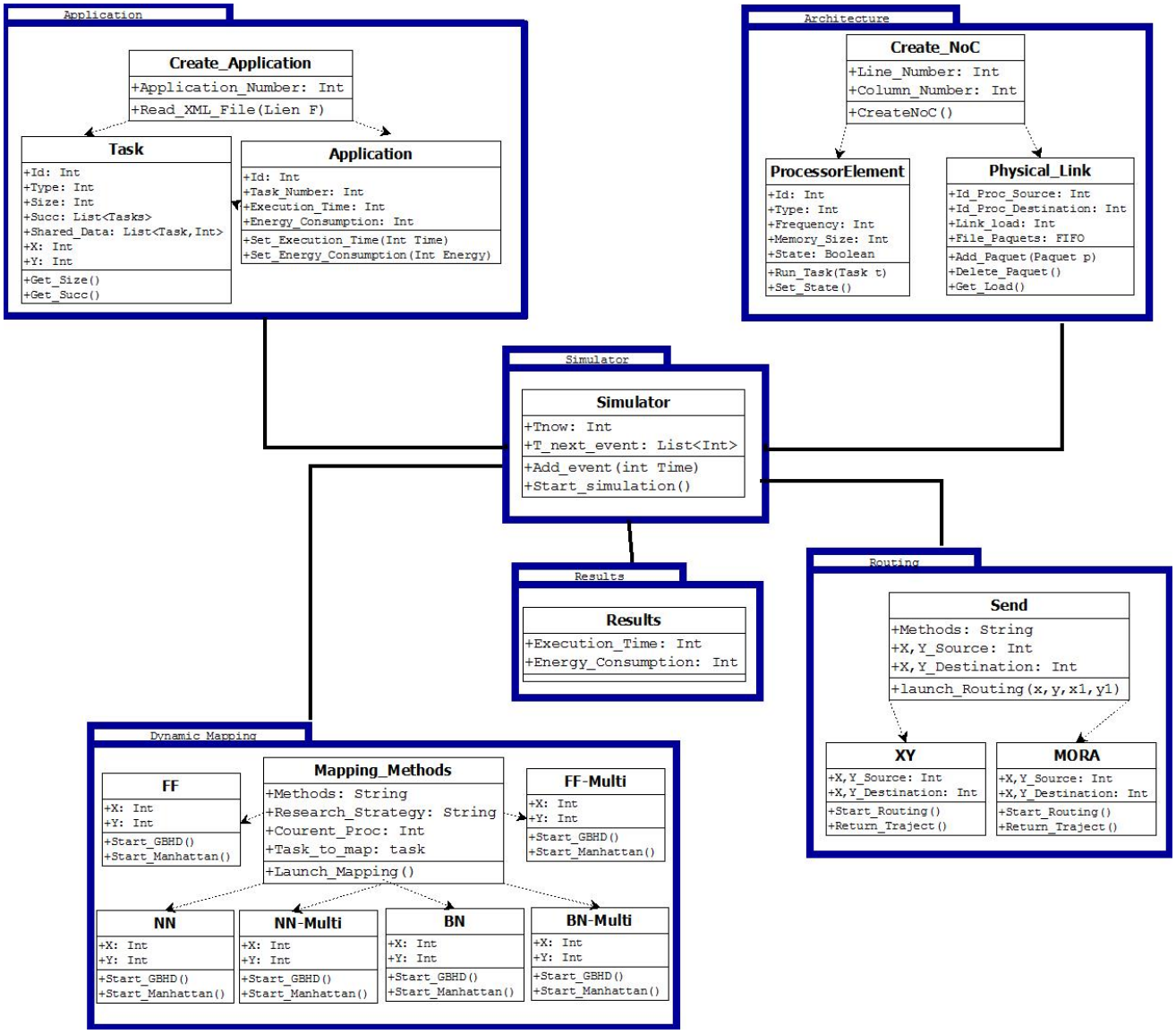
Finally, total energy consumption is calculated as follows:

$$EC_{total} = \sum EC_{w} + \sum EC^{R}_{s^{tj}_{ti}} + \sum TEC^{s}_{tn} + \sum TEC^{h}_{tn} \tag{5}$$

4.1.2 Execution time computation

The execution time for a task t_i (software or hardware) is the summation of time taken to find a mapping for the task (T^{ti}_{map}) and its communications (we do not consider the execution time of the communicating tasks mapped in the same resource that use the shared memory $T^{com^{ti}_{master}}_{map}$ and $T^{com^{master}_{ti}}_{map} = \varepsilon$), configuration time for task t_i (T^{ti}_{upload}) and execution time of the tasks ($T^{ti(s/h)}_{exe}$) in the multitasking configured resources (software resource or hardware resource). The mapping time of communications (communicating tasks mapped in different PEs) consists of time taken to map communications from the t_i's master to t_i and t_i to t_i's master. The execution of t_i is not finished until the execution of all of its slaves is not finished. As the slaves execute in parallel, the one taking the maximum time contributes to the execution time of t_i. The application execution time consists of communication time in addition to the above mention timings.

Figure 4 Packages of the simulator and the interactions between them (see online version for colours)



The execution time for every task is calculated in recursive manner as follows:

$$T_{exe}^{ti} = T_{upload}^{ti} + T_{exe}^{ti(s/h)} + T_{map}^{ti} + T_{map}^{com_{master}^{ti}} + T_{map}^{com_{ti}^{master}} + \sum_{n=1}^{slaves-master} MaxT_{exe}^{tn} \quad (6)$$

The overall execution time is calculated as the maximum execution time amongst all the applications running in parallel.

$$T_{Totalexec} = \max_{i=0\ to\ appl-numb} T_{exe}^{app} \quad (7)$$

5 Validation by simulation

To compare the mapping of tasks and communications heuristics, we have used our high-level simulator (Benhaoua et al., 2015) written in Java that provides results quite fast as compared to cycle accurate simulator. Figure 4 shows the class diagram of our tool DynMapNoCSIM (Benhaoua et al., 2015), 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.

5.1 Experimental setup

This section describes the experimental set up used. All the applications are modelled as in Figure 1. (a), with initial tasks, hardware tasks and software tasks. The values present on the edges represent the volume of data to be sent and received by the master as explained in definition application task graph. The NoC is modelled as in Figure 2 with initial tasks supported PEs at the middle position in each cluster. We have using our simulator (Benhaoua et al., 2015) to realise a heterogeneous platform that comprises 64 processors: 12 hardwares, 51 softwares, and one manager processor. Our simulator permits us to create and simulate any platform. Our choice is carried on a platform of 8 × 8 processors. For positioning types of processors on the platform, we have choosing the same architecture used in the work of Singh et al. (2010). The manager is responsible for finding placement of the applications' tasks, task configuration, platform resources update and communications routing. The platform uses a NoC as a communication support, which is responsible for data transfer between the tasks when the communicating tasks are mapped in different PEs. Manager processor knows only the initial tasks. When initial tasks start their execution, the slave tasks are mapped dynamically, according to the communication request. The processing time of tasks depends on the type and capacity of PE. We can vary several parameters through an input configuration file (parameters file) that contain all the parameters such as platform configurations, choice of dynamic mapping heuristic, routing method, memory capacities, scheduling methods in the PE (multi-tasking), etc. Each PE contains a scheduler for multi-tasking. The experiments are performed for different scenarios:

- Scenario 1: Applications generated by *task graph for free* (TGFF) as shown in Figure 5 (3-4 level, 1-3 son). Applications contain a maximum of 9 tasks.
- Scenario 2: Applications *multi-window display* (MWD), *video object plane decoder* (VOPD), *picture-in-picture* (PIP) as shown in Figure 6, and multiple MPEG-4 applications as shown in Figure 7. The MWD, VOPD, PIP and MPEG-4 contain 12, 15, 8 and 13 tasks, respectively.

For each scenario, we try to map and execute a total of ten applications, whereas any number of applications can be considered. The platform is divided into nine clusters and thus nine applications can be mapped and executed initially and one application has to wait until one of the first nine has not finished. Multiple instances of the same application are considered to take a total of ten applications in each scenario. The data volume in different scenarios has been varied. The applications with varying number of tasks are considered to see how far (in terms of number of hops) the tasks of the same application can get mapped. We must have an adaptative routing method in order to minimise the costs of communications. In the current work, software and hardware resources execute multi-tasks, to solve the deadlock problem caused by the mono-task platform when the number of tasks is less than the number of PEs.

5.2 Experimental results

Using a static communication mapping can influence the costs of communications. By employing the proposed dynamic communication mapping combined with the benefits of the multi-tasks platform, we can reduce the communication time significantly. In the case when the communicating tasks are mapped in different PEs, in our approach, the packets can take more than one trajectory (path) to facilitate for faster communication, resulting in reduced communications costs.

Figure 5 Applications generated by *Task Graph For Free* (3-4 Level, 1-3 Son) (see online version for colours)

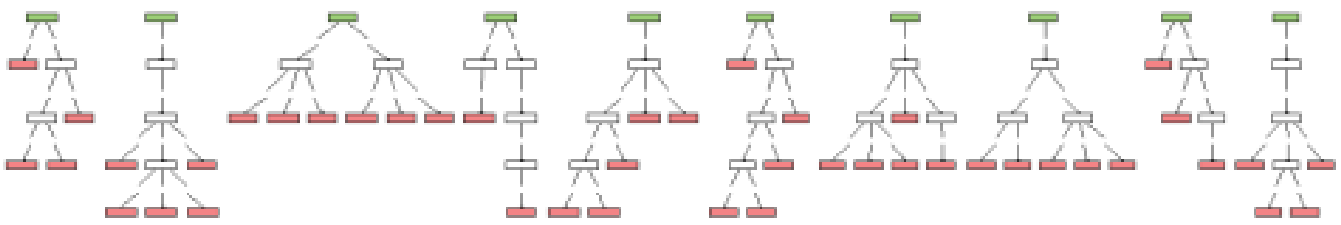
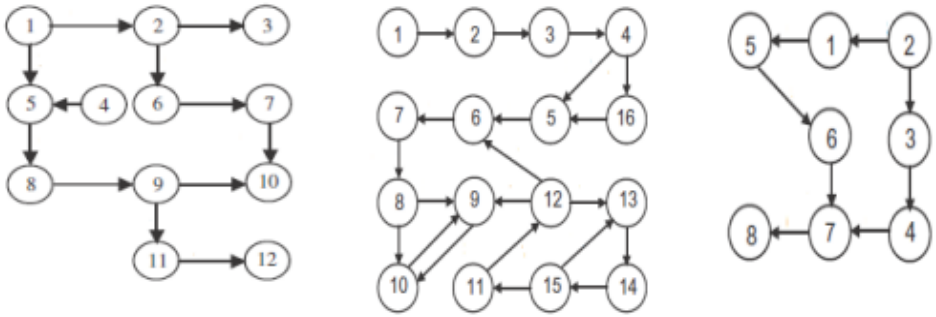


Figure 6 Applications *MWD*, *VOPD*, *PIP*



(a) (b) (c)

Figure 7 MPEG-4 applications

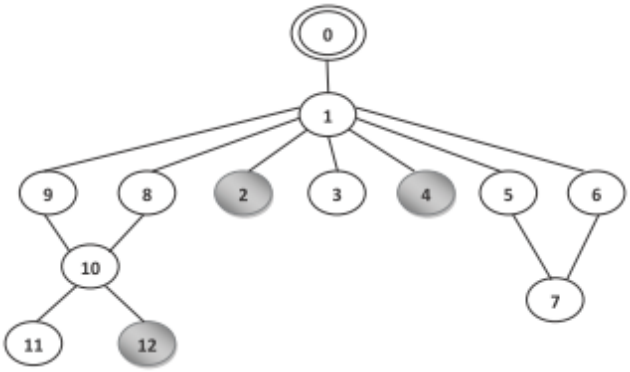
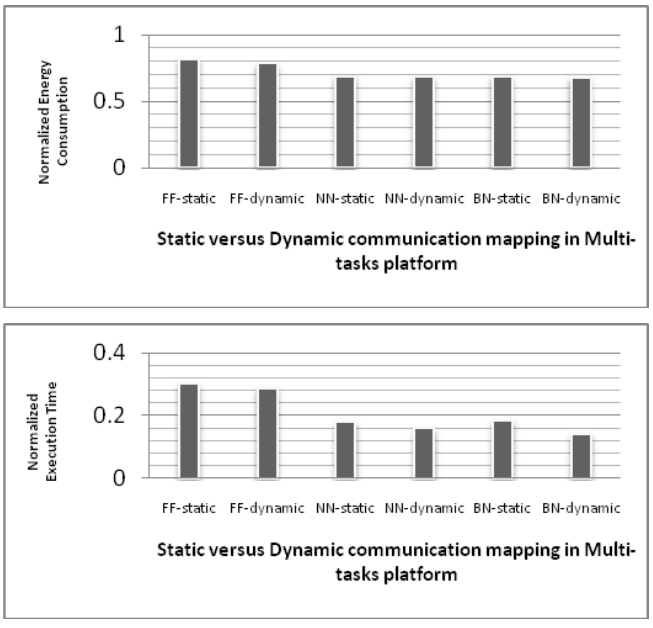
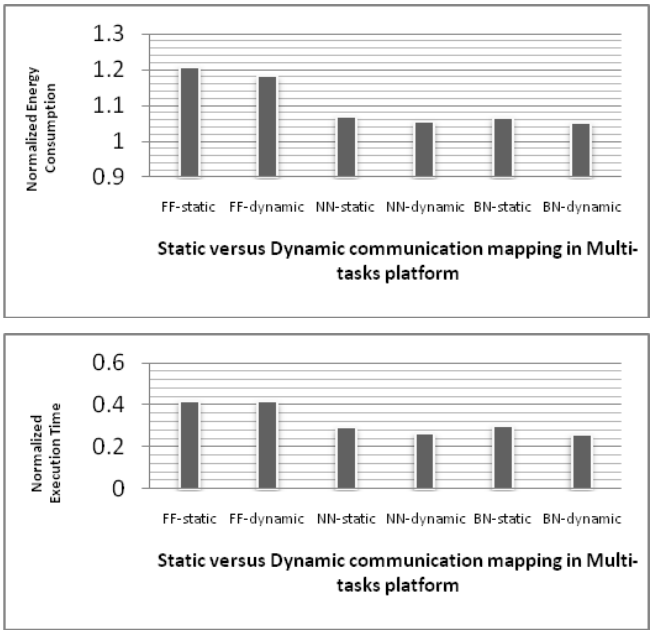


Figure 8 Execution time and energy consumption comparison of FF, NN and BN employing static and dynamic (our proposed) communication mapping for scenario 1



Graphs in Figure 8 show the normalised total execution time and energy consumption for executing ten applications considered in scenario 1 when different heuristics are applied to map the applications on the multi-tasks NoC-based heterogeneous MPSoC platform. The heuristics are applied by employing both XY and our approaches to see the impact on the total execution time and energy consumption. It can be observed that a reduction in the execution times and energy consumption is achieved when the routing approach is changed from static to our approach for all the considered heuristics. This is due to the dynamic adaptation of the paths by our approach. Therefore, the results suggest that our proposed approach should be applied with mapping heuristics in order to achieve better performance.

Figure 9 Execution time and energy consumption comparison of FF, NN and BN employing static and dynamic (our proposed) communication mapping for scenario 2



In order to evaluate the performance improvement on realistic applications by employing our approach over static, we have performed similar experiments by considering the applications of scenario 2. Figure 9 shows the total execution time and energy consumption for executing ten applications considered in scenario 2 when different heuristics are applied to map the applications on the multi-tasks NoC-based heterogeneous MPSoC platform. Similar results as that of scenario 1 can be observed in scenario 2 as well when the routing approach is changed from static to our dynamic communication mapping for all the considered heuristics. These observations show that our

proposed routing approach reduces the execution time and energy consumption for different kinds of application scenarios and can be considered as a potential candidate for efficient routing strategy. Otherwise, using multitasks platform optimise consequently the performance of the system (consumption energy, computational time) compared to mono-task platform.

6 Conclusions and future directions

This paper presents a mapping approach that performs communication mapping. When tasks are mapped in the same PE, a shared memory is used to communicate between tasks, resulting in optimised execution time and energy consumption. Also, the approach maps the communications between the tasks mapped in different PEs. To reduce the communication costs, a dynamic communication mapping algorithm has been proposed to map the communications. Experiments have shown significant reduction in total execution time and energy consumption when compared to heuristics employing static routing and mono task platform. In future, we plan to consider task migration to balance the loads on the processors and the monitoring.

References

Armin, M. et al. (2007) 'Spiral: a heuristic mapping algorithm for network on chip', *IEICE Electronic Express*, Vol. 4, No. 15, pp.478–484.

Benhaoua, M.K. et al. (2013a) 'Heuristics for dynamic task and communications mapping in NoC-based heterogeneous MPSoSs', in *The Mediterranean Journal of Computers and Networks*, October, Vol. 9, No. 4, pp.135–146, ISSN: 1744-2397.

Benhaoua, M.K. et al. (2013b) 'Heuristics for routing and spiral run-time task mapping in NoC-based heterogeneous MPSOCs', in *IJCSI International Journal of Computer Science Issues*, July, Vol. 10, No. 4, pp.233–238, ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784.

Benhaoua, M.K. et al. (2014a) 'Heuristic for accelerating run-time task mapping in NoC-based heterogeneous MPSoCs', in *Journal of Digital Information Management*, Vol. 12, No. 5, pp.292–302.

Benhaoua, M.K. et al. (2014b) 'Heuristic for accelerating run-time task mapping in NoC-based heterogeneous MPSoCs', in *ICESIT 2014: International Conference on Embedded Systems and Intelligent Technology*, Dubai, UAE, 25–26 November.

Benhaoua, M.K. et al. (2014c) 'Multi-objective routing algorithm for dynamic communications mapping in NoC-based heterogeneous MPSoCs', in *META'2014 International Conference on Metaheuristics and Nature Inspired Computing*, Marrakech, Morocco, 27–31 October.

Benhaoua, M.K. et al. (2015) 'DynMapNoCSIM: a dynamic mapping simulator for network on chip based MPSoC', in *Journal of Digital Information Management*, Vol. 13, No. 1, pp.45–54.

Benini, L. and Mecheli, G.D. (2002) 'Networks on chips: a new SoC paradigm', *Computer*, Vol. 35, No. 1, pp.70–78.

Bertozzi, D. and Benini, L. (2004) 'Xpipes: a network-on-chip architecture for gigascale systems-on-chip', *Circ. Syst. Mag. IEEE*, Vol. 4, No. 2, pp.18–31.

Carvalho, E. and Moraes, F. (2008) 'Congestion-aware task mapping in heterogeneous MPSoCs', in *SoC*.

Carvalho, E., Calazans, N. and Moraes, F. (2010) 'Dynamic task mapping for MPSoCs', *IEEE Design Test of Computers*, Vol. 13, No. 1, pp.26–35.

Chou, C.L. and Marculescu, R. (2008) 'User-aware dynamic task allocation in networks-on-chip', in *DATE*.

Chou, C.L. et al. (2007) 'Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels', in *CODES*.

Faruque, M. et al. (2008) 'Adam: run-time agent-based distributed application mapping for on-chip communication', in *DAC*.

Ghosh, P. et al. (2009) 'Energy efficient application mapping to NoC processing elements operating at multiple voltage levels', in *NoCs*, pp.80–85.

Holzenspies, P.K.F. et al. (2008) 'Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (MPSOC)', in *DATE*.

Jerraya, A. et al. (2005) 'Guest editors' introduction: multiprocessor systems-on-chips', *Computer*, Vol. 38, No. 7, pp.36–40.

Kaushik, S., Singh, A.K. and Srikanthan, T. (2011) 'Computation and communication aware run-time mapping for NoC-based MPSoC platforms', in *SOCC '11: Proceedings of the 2011 IEE International System-on-Chip Conference*.

Lei, T. and Kumar, S. (2003) 'Algorithms and tools for network on chip based system design', in *Proc. ICSD*, p.163.

Mandelli, M. et al. (2011) 'Multi-task dynamic mapping onto NoC-based MPSoCs', in *SBCCI*.

Manna, K., Chattopadhaya, S. and Sengupta, I. (2012) 'An efficient routing technique for mesh-of-tree-based NoC and its performance comparison', *Int. J. High Performance Systems Architecture*, Vol. 4, No. 1, pp.25–37.

Manolache, S. et al. (2005) 'Fault and energy-aware communication mapping with guaranteed latency for applications implemented in NoC', in *Proc. of DAC*, pp.266–269.

Maqsood, T. et al. (2015) 'Dynamic task mapping for network-on-chip based systems', *JSA*, Vol. 61, No. 7, pp.293–306.

Marcon, C. et al., (2005) 'Time and energy efficient mapping of embedded applications onto NoCs', in *Proc. of ASP-DAC*, pp.33–38.

Mehran, A. et al. (2008) 'DSM: a heuristic dynamic spiral mapping algorithm for network on chip', *IEICE Electronics*, Vol. 5, No. 13, pp.5–13.

Moraes, F. et al. (2004) 'Hermes : an infrastructure for law area overhead packet-switching networks on chip', *Integr. VLSI J.*, Vol. 38, No. 1, pp.69–93.

Murali, S. et al. (2006) 'A methodology for mapping multiple use-cases onto networks on chips', in *Proc. of DATE*, pp.118–123.

Orsila, H. et al. (2007) 'Automated memory-aware application distribution for multi-processors-systems-on-chips', *JSA*, Vol. 53, No. 11, pp.795–815.

Ost, L. et al. (2013) 'Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach', *ACM Trans. Embedded Comput. Syst.*, Vol. 12, No. 3, p.75.

Sahu, P.K. and Chattopadhyay, S. (2013) 'A survey on application mapping strategies for network-on-chip design', *JSA*, Vol. 59, No. 1, pp.60–76.

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

Shin, D. and Kim, J. (2004) 'Power-aware communication optimization for networks-on-chips with voltage scalable links', in *CODES*.

Shin, D. and Kim, J. (2004) 'Power-aware communication optimization for networks-on-chips with voltage scalable links', in *Proc. of CODES and ISSS*, pp.170–175.

Silva Junior, L.D.R.S., Nedjah, N. and Mourelle, L.M. (2012) 'Static routing for applications mapped on NoC platform using ant colony algorithms', *Int. J. High Performance Systems Architecture*, Vol. 4, No. 1, pp.57–64.

Singh, A.K. et al. (2010) 'Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms', *JSA*, Vol. 56, No. 7, pp.242–255.

Singh, A.K., Shafique, M., Kumar, A. and Henkel, J. (2013) 'Mapping on multi/many-core systems: survey of current and emerging trends', in *DAC*.

Smit, L. et al. (2004) 'Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture', in *FPT*.

Vardi, F. et al. (2009) 'Crinkle: a heuristic mapping algorithm for network on chip', *IEICE Electronics Express*, Vol. 6, No. 24, pp.1737–1744.

Wang, X. et al. (2010) 'Power-aware mapping for network-on-chip architectures under bandwidth and latency constraints', *TACO*, Vol. 7, No. 1, p.6.

Weichslgartner, A. et al. (2011) 'Dynamic decentralized mapping of tree-structured applications on NoC architectures', in *NoCs*.

Wildermann, S. et al. (2009) 'Run time mapping of adaptive applications onto homogeneous NoC-based reconfigurable architectures', in *FPL*.

Wu, D. et al. (2003) 'Scheduling and mapping of conditional task graphs for the synthesis of low embedded systems', *DATE*, pp.1090–1095.

Zhang, Y. et al. (2002) 'Task scheduling and voltage selection for energy minimization', in *Proc. DAC*, pp.183–188.