World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:11, 2014

Heuristic for Accelerating Run-Time Task Mapping in NoC-Based Heterogeneous MPSoCs

M. K. Benhaoua, A. K. Singh, A. E. H. Benyamina, A. Kumar, P. Boulet

***Abstract*—**In this paper, we propose a new packing strategy to find a free resource for run-time mapping of application tasks to NoC-based Heterogeneous MPSoC. The proposed strategy minimizes the task mapping time in addition to placing the communicating tasks close to each other. To evaluate our approach, a comparative study is carried out for a platform containing single task supported PEs. Experiments show that our strategy provides better results when compared to latest dynamic mapping strategies reported in the literature.

***Keywords*—**Multi-Processor Systems-on-Chip (MPSoCs), Network-on-Chip (NoC), Heterogeneous architectures, Dynamic mapping heuristics.

I. Introduction

THE complexity of applications need to transit from the System on Chip (SoC) based on a single processor to Multi-Processor on Chip (MPSoC) that contains multiple processing elements (PEs) in the same chip. Effectively the evolution of semiconductor technology permits to integrate a several processor 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], [2]. The actual and futures embedded applications contain dynamic workload of tasks or applications that need to be loaded into the system at run-time, this need efficient dynamic mapping techniques [9]-[11], [13], [15]-[17]. 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 nearest available PEs, i.e. close to each other in order to reduce the communication overhead [26], [27], [18], [19], [3]. However, these approaches do not perform well when applications contain large number of tasks. Further, the latest works not focus in the minimization of time search of mapping. Our contribution is to minimize a time mapping of tasks at run-time. With a large number of tasks by application, the time of search deveining is so important. A newly Manhattan packing strategy is proposed that permits to explore and place the application tasks faster than the latest existing mapping strategies, resulting in optimized costs of tasks mapping. The model used for the representation of applications is the master-slave model. This type of model is used to represent the applications which have parallel communicating tasks. The heterogeneous MPSoC platform considered is Mono-task each resource permit to execute only one task. The platforms contain two types of PEs: Instruction Set Processors (ISPs) and Reconfigurable Areas (RAs), which execute software and hardware tasks respectively. The physical platform contains 64 PEs arranged as an 8x8 mesh. The platform is divided into nine virtual clusters which permit us to launch nine applications in parallel. For the dynamic mapping of tasks, state-of-the-art mapping techniques reduce the communication costs by mapping the communicating tasks on nearest available PEs [12], [18], [28], [29]. The latest works use a packing strategy to realize this objective. However, most of them don't focus on minimization of search (mapping) time. In our proposed Manhattan packing strategy, we search not only to place the communicating tasks on nearest available PEs but also to minimize a search time. The Manhattan packing strategy show significant performance improvements when compared to latest mapping approaches.

The rest of the paper is organized as follows. Section II provides an overview of the related work. Section III describes the model of considered MPSoC architecture. In Section IV, the proposed Manhattan packing strategy it's presented. Experimental setup and the results are presented in Section V. Section VI concludes the paper and provides future research directions.

II. Related Work

Most of the existing works reported in the literature to solve the problem of mapping on MPSoC platform are static mapping techniques [3]-[8]. 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

Benhaoua Mohammed Kamel is with the Department of Computer Science, University of Oran, Oran, Algeria, BP 1524 El-Mnaouer and University Lille 1, LIFL, CNRS, UMR 8022, F-59650 Villeneuve d'Ascq, France (corresponding author phone: 00213794160366; e-mail: Mohammed-Kamel.Benhaoua@lifl.fr).

Amit Kumar Singh is with Department of Computer Science, University of York, UK (e-mail: amit.singh@york.ac.uk)

Benyamina Abou el Hassen is with Department of Computer Science, University of Oran, Oran, Algeria, BP 1524 El-Mnaouer(e-mail: benyamina.abouelhassen@univ-oran.dz).

Akash Kumar is with Department of Electrical and Computer Engineering, National University of Singapore, Singapore (e-mail: akash@nus.edu.sg).

Pierre Boulet is with University Lille 1, LIFL, CNRS, UMR 8022, F-59650 Villeneuve d'Ascq, France (e-mail: pierre.boulet@lifl.fr).

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. The existing approaches encounter large exploration time to find placement of tasks. The proposed Manhattan strategy performs faster exploration with objectives to place communicating tasks on nearest available PEs. Mapping heuristics Nearest Neighbor (NN) and Best Neighbor (BN) presented in [18] along with the packing strategy in [3] are taken for evaluation and performance comparison with our proposed Manhattan packing strategy.

III. Heterogeneous MPSOC Architecture

Fig. 1 shows the model of the heterogeneous MPSoC architecture used in this work. The architecture contains a set of different processing elements (PEs) which interact via a communication network [1]. The PEs can be of varying types such as instruction set processors (ISPs), reconfigurable logics (reconfigurable area-RA), dedicated intellectual properties (IPs), etc. Tasks to be executed onto the PEs are categorized 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 [14], as shown in Fig. 1. 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. The inter-task communication is supported by a message passing mechanism similar to used in [26].

In the MPSoC architecture, one PE is used as the Manager Processor (MP) that is responsible for task binding, task placement (mapping), application task scheduling, communication routing, resource control and reconfiguration. *Task binding* is required before task mapping in case of heterogeneous MPSoCs. For each task, the binding process defines the PEs types onto which the task can be mapped and executed. For example, software tasks will be mapped and executed on ISPs, whereas hardware tasks on RAs and IPs. *Task placement* step identifies the location of a PE in the architecture in order to allocate a task. *ApplicationTask scheduling* defines execution order of initials tasks of applications on clusters. *Communication routing* defines the mechanism to be used for routing data from one PE to another. *Resource control* is maintained by updating the resources status at run-time in order to provide the MP with accurate information about the resource occupancy. The updated resource information helps the MP to take better mapping decision at run-time, which is based on the PEs and NoC links usage at that time. The configuration overhead results are used to simulate the *configuration control* process. The MP knows only the initial tasks of the applications. The initial task of each application is started by the MP and new communicating tasks are loaded into the MPSoC platform at run-time from the *task memory* when a communication to them is required and they are not already mapped.

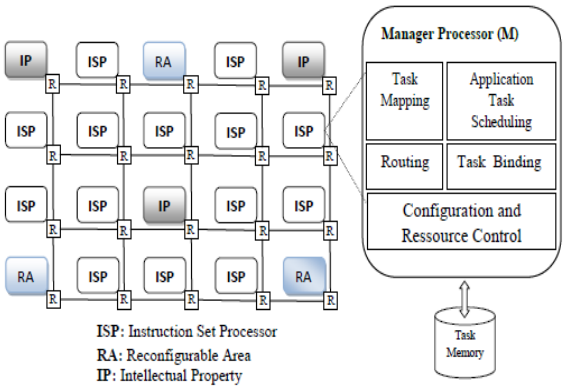

Fig. 1 Heterogeneous MPSoC Architecture

IV. Proposed Mapping Approach

This section describes our proposed run-time mapping approach. The proposed approach tries to minimize the search time for finding free resource able to execute a given task in order to optimize the task mapping process. The proposed Manhattan packing strategy reduces global computational time and energy consumption for dynamic mapping of tasks significantly. We introduce definitions for representing the application models, architecture model and mapping of applications to the architecture. The initial tasks mapping strategy is described to explain the concept of the clustering. Finally, our Manhattan packing strategy for tasks placement is elaborated.

A. 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. Fig. 2 (a) describes an application having initial, software and hardware tasks along with the edges (*E*) connecting these tasks. Fig. 2 (b) shows the master-slave pair (communicating tasks). The starting task of an application is the initial task that has no master. Edge set *E* contains all the edges along with the communicating tasks connected by the edges (Fig. 2 (b)). To transmit and receive messages by a task, deterministic XY routing algorithm is used.

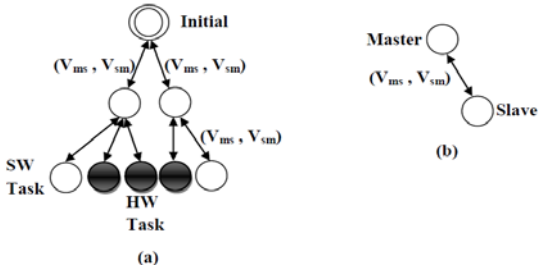

Fig. 2 Application task graph modeling and Master-Slave pair

B. NoC-Based Heterogeneous Mpsoc Architecture

A 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.

C. Mapping

The task mapping is represented as $mpg(t_i \rightarrow p_i)$, where task t_i of an application is mapped onto tile p_i in the MPSoC architecture. The application mapping considers mapping of all the application tasks onto different PEs in the architecture while optimizing for some performance metrics. Multiple applications mapping involves allocation of tasks from different applications in parallel while performing optimization for each of the application. This work considers simultaneous mapping of multiple applications to the MPSoC architecture.

D. Initial Tasks Mapping

The initial task mapping has a significant impact on the performance of the run-time mapping. The initial tasks are considered as software tasks and thus are mapped on software resources (ISPs). The initials tasks of applications are placed in a distributive way in the whole architecture. Towards this, the architecture is partitioned into multiple distributed clusters and the initial tasks are placed at the center of the clusters, as shown in Fig. 3. Such placement of initial tasks of different applications facilitates the mapping of tasks of each application close to each other within a particular region (cluster). This reduces communication costs as communicating tasks of each application get mapped in close proximity. The frontiers of clusters are virtual and thus the common regions could be shared by the tasks of different applications. After the initials tasks of each application are placed, communication requests are sent to the communicating tasks in order to find their placement. Next, we introduce the reference heuristics and our approach to be used to find the placement of the tasks.

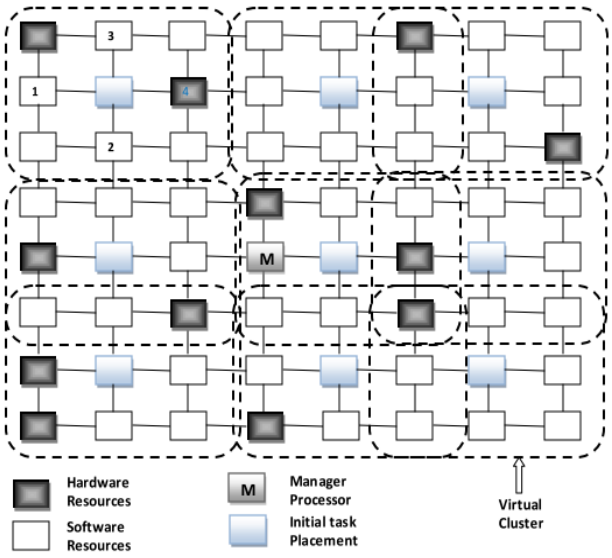

Fig. 3 Initial tasks placement for mapping 9 applications simultaneously

E. Proposed Manhattan Packing Strategies

Before describing the proposed Manhattan packing strategies, we demonstrate an example mapping by the PNN strategy to highlight its limitations, which has been considered as one of the reference mapping heuristic.

To map a requested task, firstly, the task is tried to be mapped on the PEs around the node making the request at hop distance of one. The PEs are searched in the sequence of left, down, top and right, denoted as 1, 2, 3 and 4, respectively in Fig. 4 (a).This way, first, left and down side PEs are searched to find the placement. If neither left nor down side PE is able to execute the task, then task is tried to be mapped on the top or right side PE according to the above defined sequence. The same strategy is followed from lower to higher hop distances until a free supported PE is found. Each application follows above defined strategy to map the requested tasks on the MPSoC platform resources. The rest of the tasks get mapped as shown in Fig. 4 (a). The limitations of the approach PNN are observed when the number of tasks in the considered application is large. Most of the existing and reference works do not consider applications with large number of tasks. In case of large number of tasks in the application, the PNN does not perform well in terms of the mapping time. Additionally, for the appropriate position of the hardware resource in the platform, PNN incurs high mapping time as it has to search for 20 PEs before reaching the hardware PE, as shown in Fig. 4 (a). The searching sequence is mentioned as 1 to 20. The

proposed approach reduces the mapping time for applications with large number of tasks.

1. Manhattan Packing-Based Nearest Neighbor (MPNN)

To map a requested task, the MPNN firstly tries to find the placement at hop distance of one in the similar manner as that of PNN. The searching sequence for the PEs is denoted as 1, 2, 3 and 4 in Fig. 4 (b). For the second hop, the PEs are searched in the sequence of Left, left-Down, Down, Right-Down, Right, Top-Right, Top and Top-Left, denoted by different numbers at two hop PEs starting from 5. The similar search strategy is followed for higher hop distances as well. To map a requested hardware task, the PNN strategy needs to evaluate 20 (search number) PEs in order to find the hardware PE that can support the task, as shown in Fig. 4 (a). In contrast, our MPNN strategy needs to evaluate only 9 (search number) PEs.

The MPNN heuristic has been introduced in Algorithm 3. For mapping requested task, the algorithm takes the *requested Task*, X&Y position (coordinate) of the PE that executes the master task and platform size (NoC-limit) as input and provides X-Y coordinate of the PE (X' & Y') that can execute the task. A variable *mapped* is used to test whether the task has been mapped or not, and is initially assigned as false. Hop is use to determine distance between the master and requested task PEs.

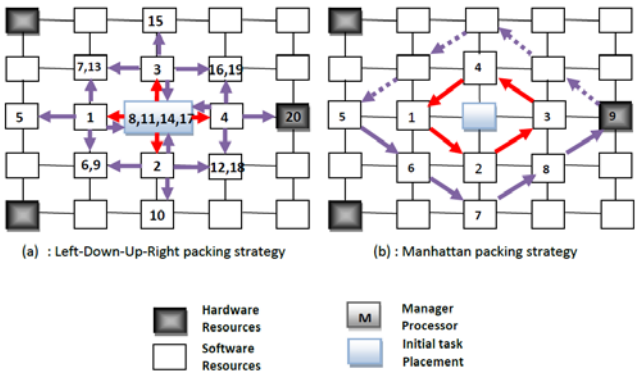

Fig. 4 Nearest Neighbor with packing strategies

The mapping for the task is searched from hop 1 to NoC-limit and is stopped as soon as a free supported PE is found. For hop=1, the MPNN evaluates PEs in the sequence of Left, Down, Right and Top. For hop equals to 2 and onwards, the evaluation sequence is Left, left-Down, Down, Right-Down, Right, Top-Right, Top and Top-Left, as described in the algorithm. For each evaluation, the MPNN calls search function presented in Algorithm 2, i.e. similar search as that of PNN is applied. This function tests whether the position of the PE is within the NoC-limit and the PE can support the task. The function returns position of the PE if *1)* the PE is within the NoC-limit, *2)* the types of task and PE are the same and *3)* the PE is free. Additionally, the function assigns *mapped* to *true* to indicate that a resource able to execute the requesting task is found.

```
Input: NoC-limit, requestedTask, X, Y
Output: X', Y'
1: mapped ← False; hop ← 1;
2: while (hop<NoC-limit) and (mapped=false) do
4:       search (requestedTask, X, Y-hop) //search Left
6        for (i=1; i<hop; i++) do //search Left Down
7:         IF (mapped=false) then
8:         search (requestedTask, X+i, Y-(hop-i));
9:       end for
11:      IF (mapped=false) then //search Down
12:        search (requestedTask, X+hop, Y)
14:      for (i=1; i<hop; i++) do //search Right Down
15:        IF (mapped=false) then
16:          search (requestedTask, X+(hop-i), Y+i)
17:      end for
19:      IF (mapped=false) then //search Right
20:          search (requestedTask, X, Y+hop)
22:      for (i=1; i<hop; i++) do //search Top Right
23:        IF (mapped=false) then
24:          search (requestedTask, X-i, Y+(hop-i))
25:      end for
24:      IF (mapped=false) then //search Top
25:          search (requestedTask, X-hop, Y)
27:      for (i=1; i<hop; i++) do //search Top Left
28           IF (mapped=false) then
29:          search (requestedTask, X-(hop-i), Y-i)
30:      end for
31:  hop++;
32: end while
```

Fig. 5 MPNN Heuristic (Algorithm 3)

```
Input: requestedTask, NoC-limit, X, Y
Output: X', Y'
1: if (0<=X<=NoC-limit) && (0<=Y< =NoC-limit)) then
2:   if PE[X][Y].isfree&& (requestedTask.type = PE[X][Y].type) then
3:              mapped←true
4:              X'← PE.X
5:              Y'← PE.Y
6:    end if
7:  end if
```

Fig. 6 Search Algorithm (Algorithm 2)

2. Manhattan Packing-Based Best Neighbor (MPBN)

The search space of MPBN strategy is similar to MPNN. In contrast to MPNN that maps the task on the first free supported PE, the MPBN evaluates all the PEs at the same hop distance and then selects the one imposing minimum path load. If a free supported PE is found within the current hop distance, then the evaluation for higher hops is discarded and the algorithm gets terminated.

V. Experimental Set-Up and the Results

For the implementation, we have used JAVA as high level programming language, which has enabled us to quickly compare various algorithms.

A. Experimental Set-Up

This section describes the experimental set up used. All the applications are modeled as in Fig. 2 (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. The NoC is modeled as in Fig. 3 with initial tasks supported PEs at the middle position in each cluster. We have realized a heterogeneous platform that comprises 64

processors: 12 hardware, 51 software, and one manager processor. The manager is responsible for finding placement of the applications' tasks, task configuration, platform resources update and communications routing. The platform uses a Network-on-Chip (NoC) as a communication support, which is responsible for data transfer between the tasks. Manager processor knows only the initial tasks. When initial tasks start their execution, the slave tasks are mapped dynamically, according to the communication request. Concerning the applications, we have used XML to describe the application task graph. The processing time of tasks depends on the type and capacity of processor. We can vary several parameters through an input configuration file (parameters file) that contain all the parameters such us platform configurations, choice of dynamic mapping heuristic, etc.

B. Experimental Results

Results obtained from our proposed Manhattan heuristics MPNN and MPBN are compared with existing run-time mapping heuristics PNN and PBN. Our heuristics show lower mapping (search) time.

1. Case Study: Real-Life Applications

Total execution time comprises of mapping time (the time to find the placement), configuration time, communication time, computation time and waiting time when no resource is free in the platform. The adaptation of packing strategy in the mapping process facilitates mapping of communicating tasks in close proximity and thereby reducing the communication time. It has also been observed that the mapping time contributing to total execution time gets reduced when employing the Manhattan heuristics because the search space to find the placement of a task is minimized. Fig. 7 shows the total execution time taken for executing 10 applications considered for scenarios 1 and 2 when different heuristics are applied. A couple of observations can be made from the figure. First, the proposed approaches MPNN and MPBN reduce the execution time when compared to the PNN and PBN, respectively. These observations show that our approaches reduce the execution time with respect to existing approaches even when existing routing approach XY is employed. In scenario 1, our proposed approaches MPNN and MPBN reduce the total execution time when compared with PNN and PBN. In scenario 2, our proposed approaches MPNN and MPBN reduces the total execution time when compared to PNN and PBN. The reduction is more than the scenarios 1 as application MPEG-4 contains 13 tasks and one of the masters contains 7 slaves. The proposed approaches will provide better results when the applications containing more number of tasks and slaves per master are considered.

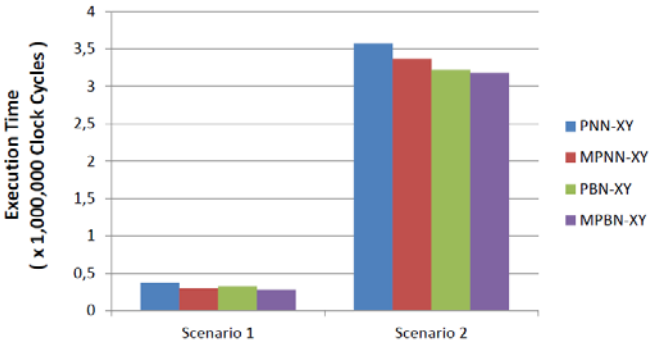

Fig. 7 Execution Time comparison of PNN and PBN with MPNN and MPBN for scenario 1 and scenario 2

2. Performance Evaluation for Large Size Applications

We have evaluated the performance for large size applications considered in Scenario 3. Four sets of applications are considered, where each set contains 10 applications with 5, 10, 15 and 20 tasks. Fig. 8 shows the execution times for four application sets considered in Scenario 3. Observations can be made from Fig. 8, the Manhattan approaches MPNN and MPBN show further reduction. Second, the reduction in execution by our approach over the existing approach increases as the number of tasks in the considered applications is increased. This is due to the fact that existing approaches encounter large search time to find mappings for tasks, whereas our approach finds the mappings in lesser time. The difference in search time by existing and our approaches increases with the number of tasks in considered applications. Therefore, our approach provides more savings in total execution time for large size applications. Fig. 9 shows energy consumption for four application sets considered in Scenario 1. It can be observed that the reduction in energy consumption by our approach over the existing approach increases as the number of tasks in the considered applications is increased. Thus, our approach provides better savings for large size applications.

3. Search Number of Different Applications

We have computed the complexity of different heuristics in terms of number of searches to be performed for mapping all the tasks in an application set. It has already been demonstrated that the number of searches by the proposed Manhattan strategy is less than the existing strategies, for example 9 vs. 20 as shown in Fig. 4. Table I shows the number of searches by different heuristics for four application sets, where each set contains 10 applications. Each application in the four sets contains 5, 10, 15 and 20 tasks, respectively. A couple of observations can be made from Table I. First, the number of searches is greatly reduced by the Manhattan strategies MPNN and MPBN when compared to PNN and PBN, respectively. Second, the proposed Manhattan strategies show higher reduction in the number of searches for applications with large number of tasks. Thus, the Manhattan strategies reduce the complexity and further reduction is expected for applications with higher number of tasks. The complexity comparison of different algorithms in terms of

algorithm execution time has shown that our approaches have reduced complexity.

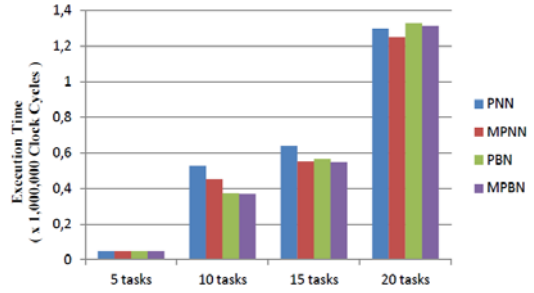

Fig. 8 Execution Time of 10 applications for four applications sets (Scenario 3), where each application contains 5, 10, 15 and 20 tasks

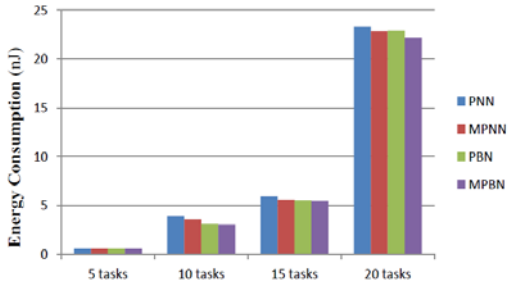

Fig. 9 Energy Consumption of 10 applications for four applications sets (Scenario 3), where each application contains 5, 10, 15 and 20 tasks

TABLE I
NUMBER OF SEARCHES FOR ALL THE TASKS IN DIFFERENT APPLICATION SETS FOR SCENARIO 3 WHEN EMPLOYING EXISTING AND MANHATTAN STRATEGIES

	Apps-5tasks	Apps-10tasks	Apps-15tasks	Apps-20tasks
PNN	1690	10370	16360	46650
MPNN	1540	7950	12470	43210
PBN	2280	10800	17200	51130
MPBN	2260	9800	15130	48150

VI. CONCLUSION AND FUTURE DIRECTIONS

This paper presents a mapping approach that performs mapping in two different phases. The first phase employs newly proposed Manhattan-based mapping strategies that try to map the application tasks in close proximity in order to reduce the communication costs. The Manhattan strategy reduces the mapping time for each task. The second phase maps the communications between the tasks. To reduce the communication costs, a multi-objective routing algorithm (MORA) has been proposed to map the communications. Experiments have shown that the proposed Manhattan-based mapping strategies along with the MORA show significant reduction in total execution time and energy consumption when compared to existing approaches. In future, we plan to consider embedded applications with growing complexity and analyze execution of increasing number of applications at the same time. We also plan to consider multi-task supported software and hardware processors in the MPSoC, where each processor will be able to support several tasks depending upon the processor memory capacity. Additionally, task migration to balance the loads on the processors will be considered.

REFERENCES

[1] L.Benini and G. D. Mecheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35 , Issue: 1, pp. 70–78, 2002.
[2] D.Bertozzi and L.Benini, "A network-on-chip architecture for gigascale systems-on-chip," *Circuits and Systems Magazine, IEEE*, vol. 4 , Issue: 2, pp. 18–31, 2004.
[3] A.Singh, T.Srikanthan, A.Kumar, and W.Jigang, "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms," *Journal of Systems Architecture*, vol. 56 , Issue: 7, pp. 242–255, 2010.
[4] Y.Zhang and al, "Task scheduling and voltage selection for energy minimization," in *Design Automation Conference, 2002. Proceedings. 39th*, 2002.
[5] D.Shin and J.Kim, "Power-aware communication optimization for networks-on-chips with voltage scalable links," in *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on*, 2004.
[6] F.Vardi, S.Saeidi, and A.Khademzadeh, "Crinkle: A heuristic mapping algorithm for network on chip," *IEICE Electronics Express*, vol. 6 , Issue: 24, pp. 1737–1744, 2009.
[7] Carvalho and al., "Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs," in *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, 2009.
[8] Smit and al., "Run-time mapping of applications to a heterogeneous SoC," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2004.
[9] Holzenspies, "Mapping streaming applications on a reconfigurable MPSoC platform at run-time," in *System-on-Chip, 2007 International Symposium on*, 2007.
[10] CL.Chou and R. Marculescu, "Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, 2007.
[11] CL.Chou and R. Marculescu, "User-aware dynamic task allocation in networks-on-chip," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008.
[12] A. Mehran, A. Khademzadeh, S. Saeidi, "DSM: A heuristic dynamic spiral mapping algorithm for network on chip," *IEICE Electronics*, vol. 5 , Issue: 13, pp. 5–13, 2008.
[13] Marcelo, "Multi-task dynamic mapping onto NoC-based MPSoCs," in *SBCCI '11 Proceedings of the 24th symposium on Integrated circuitsand systems design*, 2011.
[14] E.Carvalho and F.Moraes, "Congestion-aware task mapping in heterogeneous MPSoCs," in *System-on-Chip, 2008. SOC 2008. International Symposium on*, 2008.
[15] S.Wildermann, T.Ziermann, and J.Teichet, "Run time mapping of adaptive applications onto homogeneous NoC-based reconfigurable architectures," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009.
[16] Holzenspies, J.Hurink, J.Kuper, and G.Smit, "Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (MPSOC)," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008.
[17] A.Schranzhofer, C.Jian-Jia, L.Santinelli, and L.Thiele, "Dynamic and adaptive allocation of applications on MPSoC platforms," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, 2010.
[18] E.Carvalho, N.Calazans, and F.Moraes, Dynamic task mapping for MPSoCs*IEEE Design Test of Computers*, vol. 27 , Issue: 5, pp. 26–35, 2010.
[19] A.K.Singh and al, "Eficient heuristics for minimizing communication overhead in NoC-based heterogeneous MPSoC platforms," in *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*, 2009.
[20] M.Faruque, R.Krist, and J.Henkel, "Adam: Run-time agent-based distributed application mapping for on-chip communication," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, 2008.
[21] A. Jerraya et al., Guest editors' introduction: multiprocessor systems-on-chips, Computer 38 (7) (2005) 36–40.

[22] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, N. Borkar, An 80-tile 1.28tflops network-on-chip in 65nm cmos, in: Solid-State Circuits Conference, 2007, pp. 98–589.
[23] D. Bertozzi, L. Benini, Xpipes: a network-on-chip architecture for gigascale systems-on-chip, Circ. Syst. Mag. IEEE 4 (2) (2004) 18–31.
[24] L. Smit et al., Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture, in: FPT, 2004, pp. 421–424.
[25] M. Kistler et al., Cell multiprocessor communication network: built for speed, IEEE Micro 26 (3) (2006) 10–23.
[26] E. Carvalho, N. Calazans, F. Moraes, Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs, IEEE International Workshop on Rapid system Prototyping (RSP), 2007, pp. 34–40.
[27] P.K Sahu and S.Chattopadhyay, A survey on application mapping strategies for Network-on-Chip design, Journal of Systems Architecture: the EUROMICRO Journal, Volume 59 Issue 1, January, 2013, pp 60-76.
[28] A.K. Singh, W. Jigang, A. Kumar, T. Srikanthan, Run-time mapping of multiple communicating tasks on MPSoC platforms, Procedia Computer Science, 2010, pp. 1019-1026.
[29] A.K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on multi/many-core systems: survey of current and emerging trends, Proceedings of the 50th Annual Design Automation Conference (DAC), 2013, pp. 1-10.

Benhaoua Mohammed Kamel received the engineer degree in artificial intelligence and a Magister degree in information security and networking from Oran university computer science department, Algeria, in 2005 and 2009, respectively. He is currently working toward the Ph.D. degree from Lille1 University, France and Oran University, Algeria. His research interests include NoC-based MPSoC design, parallel processing, optimization, design space exploration (DSE) and run-time mapping techniques for MPSoC.

Amit Kumar Singhreceived the B.Tech. degree in Electronics Engineering from Indian School of Mines, Dhanbad, India, in 2006. Thereafter, he worked with HCL Technologies, India for year and 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 post-doctoral 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 30 papers in leading related international journals/conferences.

BenyaminaAbbou el hassen graduated from Department of Computer Science, Faculty of Sciences, University of Oran, Algeria, where he received PhD degree in computer science in 2008. He is currently professor of computer science in the Univ. Es-senia ORAN, Sciences and Technologies, Algeria. He is head of the LAPECI team. His research works include parallel processing, optimization, design space exploration and Model Driven Engineering with the special focus on real-time and embedded systems.

Akash Kumar received the B.Eng. degree in computer engineering from the National University of Singapore (NUS), Singapore, in 2002. He received the joint Master of Technological Design degree in embedded systems from NUS and the Eindhoven University of Technology (TUe), Eindhoven, The Netherlands, in 2004, and received the joint Ph.D. degree in electrical engineering in the area of embedded systems from TUe and NUS, in 2009. Since 2009, he has been with the Department of Electrical and Computer Engineering, NUS. Currently, he is an Assistant Professor in the department. His research interests include analysis, architectures, design methodologies, and resource management of embedded multiprocessor systems. He has published over 40 papers in leading international electronic design automation journals and conferences.

Pierre Boulet was born in Lille, France, in 1970. He earned a DEA d'Informatique Fondamentale (Mastersdegree) in 1993 and a PhD of computer science in 1996 from the ÉcoleNormaleSupérieure de Lyon, France. He is currently professor of computer science in the Univ. Lille 1, Sciences et Technologies, France. His interests range from parallelism, compilation, embedded system co-design to model driven engineering and synchronous languages. He is currently investigating how to program time and energy aware mobile applications on post-Moore architectures. He is deputy director of the LIFL (computer science laboratory of Lille) and head of the DART team. He is a member of the HiPEAC EU network of excellence, and of the IEEE and ACM professional societies.