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

Mohammed Kamel Benhaoua[1, 3], AmitKumar Singh[2], Abou El Hassan Benyamin[3]
AkashKumar[2], Pierre Boulet [1]
[1]University Lille1, LIFL, CNRS, UMR 8022
F-59650Villeneuve d'Ascq, France
[2]Department of Electrical and Computer Engineering
National University of Singapore, Singapore
[3]Department of Computer Science
University of Oran, Algeria
{Mohammed-Kamel.Benhaoua, pierre.boulet}@lifl.fr, {eleaks, akash}@nus.edu.sg
benyamina.abouelhassen@univ-oran.dz

Journal of Digital Information Management

ABSTRACT: *MultiProcessor Systems on Chip (MPSoC) has emerged as a solution to adress the incremental Computational requirements for future applications. The Network-On-Chip (NoC) has been introduced as a power-efficient, scalable inter communication, interconnection mechanism between processors. One important phase in architectural exploration in NOC-based MPSOC is the mapping. The application and architectural are represented by processing model, application task graph and architectural graph respectively. Mapping parallelized tasks of applications onto these MPSoCs can be done either at design-time (static) or at run-time (dynamic). Static mapping strategies find the best placement of tasks at design-time and hence these are not suitable for dynamic workload and seem incapable of run-time resource management. The number of tasks or applications executing in MPSoC platform can exceed the available resources, requiring efficient run-time mapping strategies to meet with this constraints. In this paper, we propose a new packing strategy to find free resources for run-time mapping of application tasks on NoC-based Heterogeneous MPSoCs. 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. Experiments show that our strategy provides better results when compared to latest dynamic mapping strategies reported in the literature.*

Categories and Subject Descriptors
C.1.2 [Multiple Data Stream Architectures]; [B.4.3 Interconnections (Subsystems)]: Physical structures

General Terms:
Multiprocessors, Network-on-chip

Keywords: Multi-Processor Systems-on-Chip (MP-SoCs), Network-on-Chip (NoC), Heterogeneous Architectures, Dynamic Mapping Heuristics

Received: 10 May 2014, Revised 28 June 2014, Accepted 4 July 2014

1. Introduction

The complexity of applications demand to transit from the System-on-Chip (SoC) based on a single processor to Multi-Processor System-on-Chip (MPSoC) that 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 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]. However, these approaches do not perform well when applications contain large number of tasks. Further, the latest works do not focus on the minimization of search time to find a mapping. One of our contributions in this paper is a mapping strategy that tries to minimize mapping time of tasks at run-time. For applications containing large number of tasks, the search time becomes large and thus reducing the search time is of paramount importance. Towards achieving the same, we have proposed a Manhattan packing strategy 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 permits to execute only one task on each resource. 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]. 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 2 provides an overview of the related work. Section 3 describes the model of considered MPSoC architecture. In Section 4, the proposed Manhattan packing strategy is presented. Experimental setup and the results are presented in Section 5. Section 6 concludes the paper and provides future research directions.

2. 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], [4], [5], [6], [7], [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 mapping of applications tasks onto NoC-based MPSoCs while optimizing for different performance metrics.

Mehran at al [12] propose a Dynamic Spiral Mapping (DSM) technique for task mapping during run-time. Faruque at al. [20] propose a decentralized agent-based mapping approach targeting large NoC-based heterogeneous MPSoCs such as 32 ´ 64 systems. Carvalho at 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 at 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.

3. Heterogeneous MPSoC Architecture

Figure 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 Figure 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 that of [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. Application Task 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

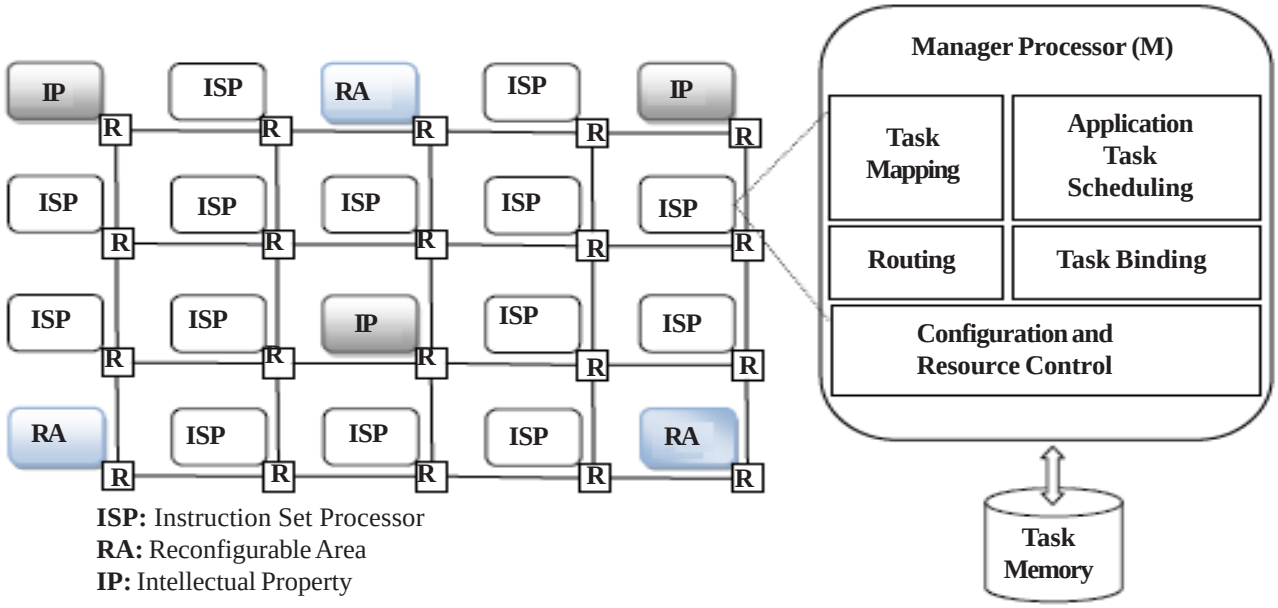

Figure 1. Heterogeneous MPSoC Architecture

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.

4. 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.

4.1 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 2 (a). describes an application having initial, software and hardware tasks along with the edges (E) connecting these tasks. Figure 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 (Figure 2 (b)). To transmit and receive messages by a task, deterministic XY routing algorithm is used.

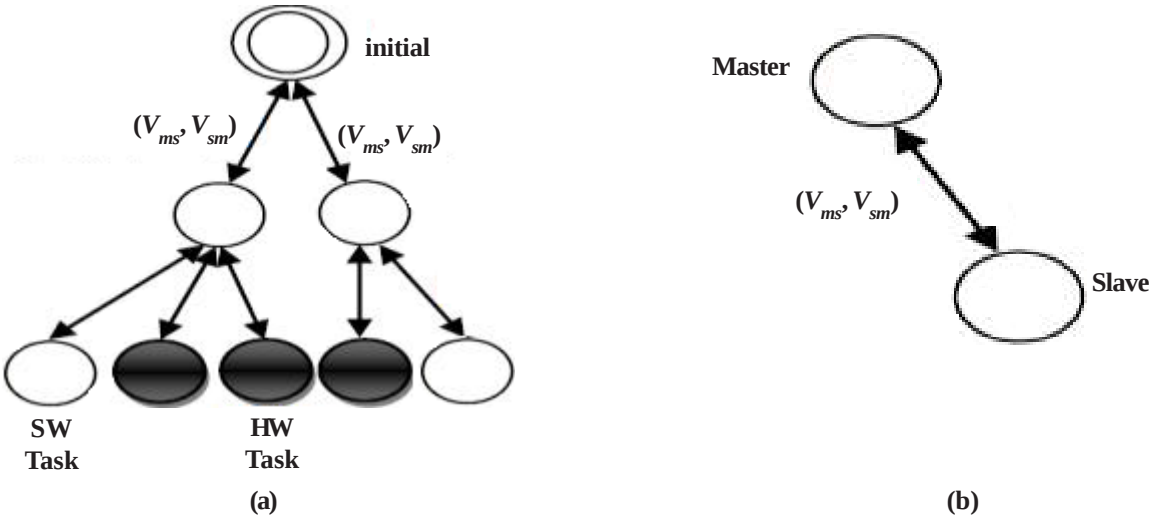

Figure 2. Application task graph modeling and Master-Slave pair

4.2 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.

4.3 Mapping

The task mapping is represented as $mpg\ (t_i \rightarrow p_i)$, where task t_i of an application is mapped onto tile *pi* 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 on the MPSoC architecture.

4.4 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 Figure 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.

4.5 Proposed Manhattan Packing Strategies

Before describing the proposed Manhattan packing strategies, we demonstrate an example mapping by the Packing-based Nearest Neighbor (PNN) strategy ([3]) 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 Figure 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 Figure 4 (a). The limitations of the PNN approach 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 searching sequence is mentioned as 1 to 20. The proposed approach reduces the mapping time for applications with large number of tasks.

4.5.1 Manhattan Packing-based Nearest Neighbor (MPNN)

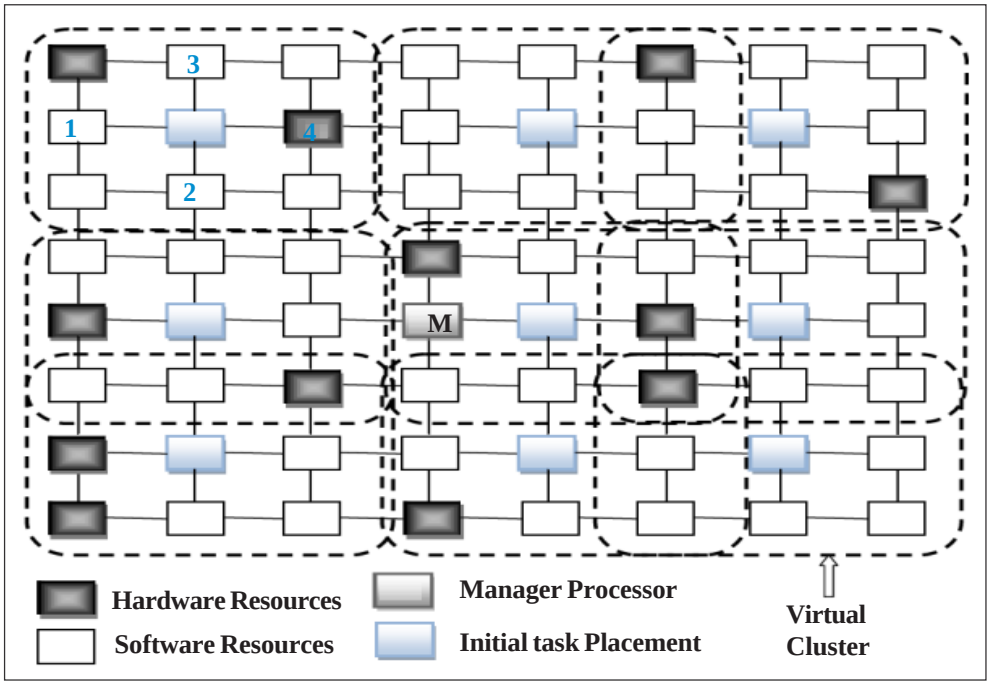

Figure 3. Initial tasks placement for mapping 9 applications simultaneously

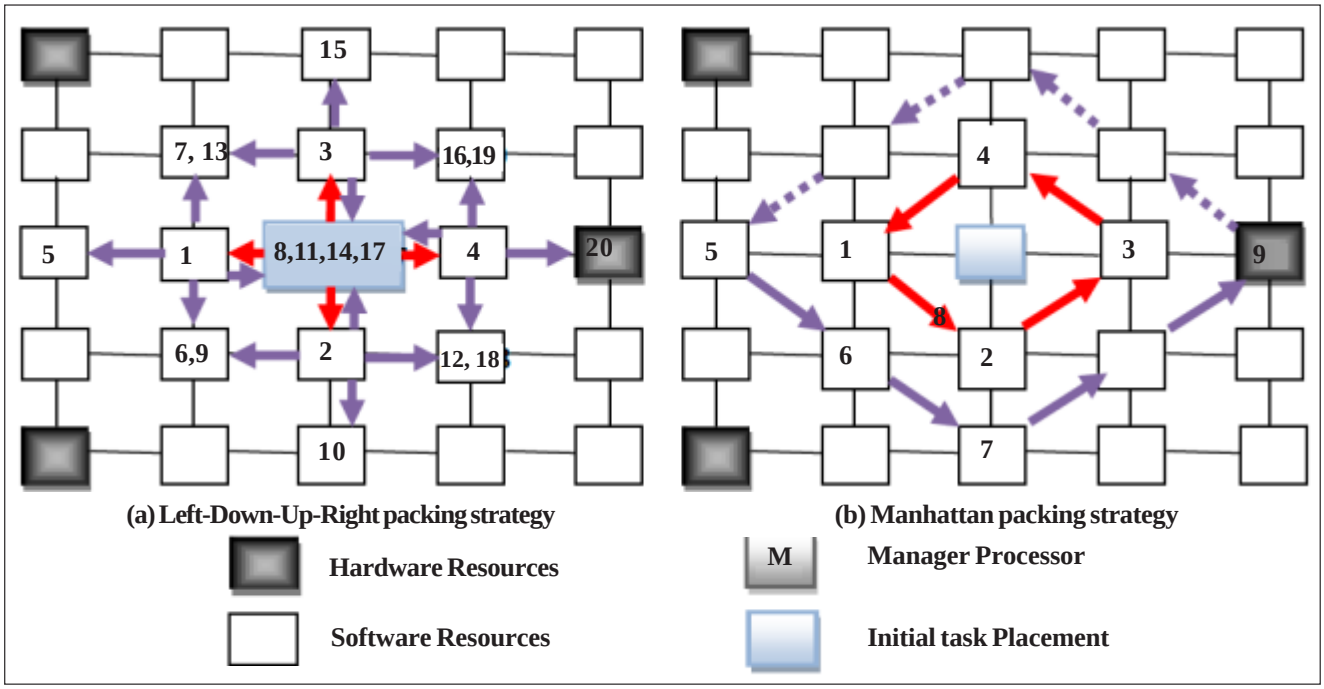

Figure 4. Nearest Neighbor with packing strategies

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 Figure 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 Figure 4(a). In contrast, our MPNN strategy needs to evaluate only 9 (search number) PEs.

The MPNN heuristic has been introduced in Algorithm 1. For mapping a requested task, the algorithm takes the *requestedTask*, 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.

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

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

Algorithm 1: MPNN Heuristic

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

Algorithm 2: Search Algorithm

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. 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.

4.5.2 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 imposingminimum 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.

5. 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.

5.1 Experimental Set-up

This section describes the experimental set up used. All the applications are modeled as in Figure 2(a), with initial, hardware 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 4.1 The NoC is modeled as in Figure 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.

The experiments are performed for different scenarios:

• **Scenario 1:** Applications *Multi-Window Display* (MWD), *Video Object Plane Decoder* (VOPD), *Pecture-In Picture* (PIP) as shown in Figure 5. The MWD, VOPD and PIP contain 12, 15 and 8 tasks, respectively.

• **Scenario 2:** Multiple MPEG-4 Applications. The application contains 13 tasks (Figure 9).

• **Scenario 3:** Four application sets: 1st set - each application having 5 tasks, 2nd set - each application having

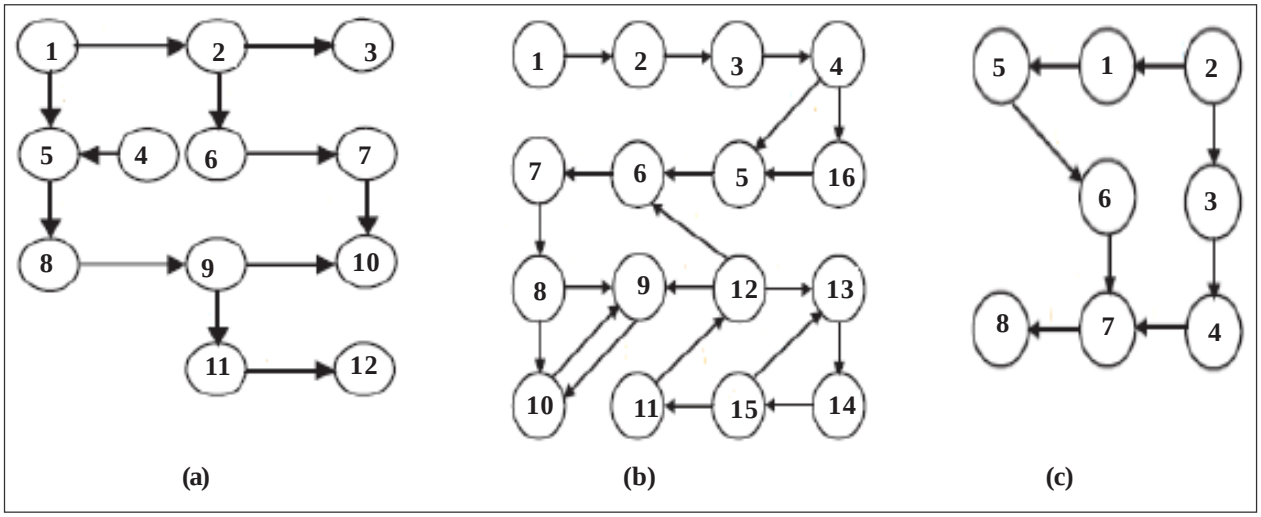

Figure 5. Applications (a) MWD, (b) VOPD, (c) PIP

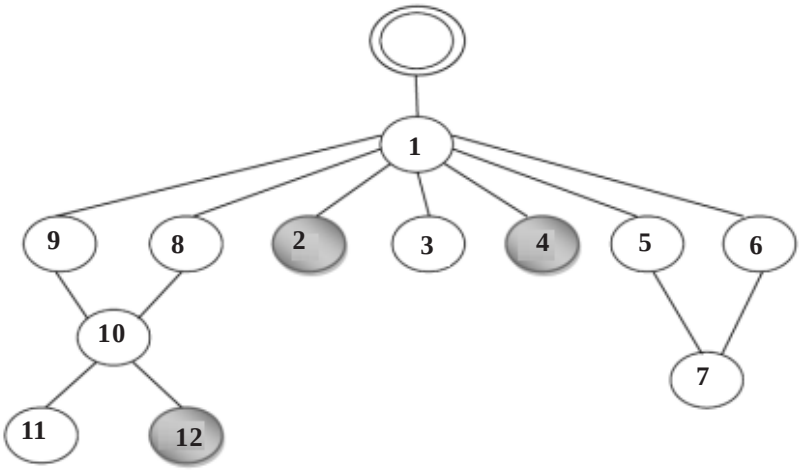

Figure 6. An MPEG-4 Application

10 tasks, 3rd set - each application having 15 tasks, and 4th set - each application having 20 tasks.

For each scenario, we try to map and execute a total of 10 applications, whereas any number of applications can be considered. The platform is divided into nine (9) 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

For large size application, the tasks at the leaf of application graph can get mapped far from the initial task. The mapping time for a task increases with the number of hops as a large number of PEs needs to be evaluated. This calls for an efficient technique that can minimize the mapping time and our approach provides a solution towards the same. In the current work, software and hardware resources can execute only one task. Multi-tasking hardware and software resources will be considered as future work.

5.2 Experimental Results

Results obtained from our proposed Manhattan heuristics MPNN and MPBN are compared with existing run-time mapping heuristics PNN and PBN reported in [3]. Our heuristics show lower mapping (search) time.

5.2.1 Case study: Real-life Applications

Total Execution Time Evaluation: 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 heuristic because the search space to find the placement of a task is minimized. Graphs in Figure 7 show 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

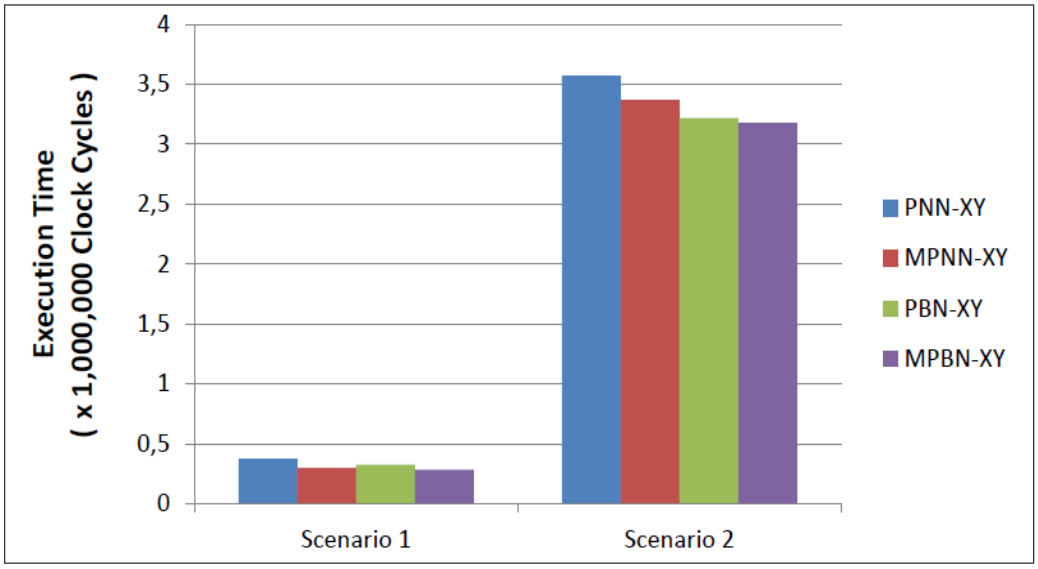

Figure 7. Execution Time comparison of PNN and PBN with MPNN and MPBN for scenario 1 and scenario 2

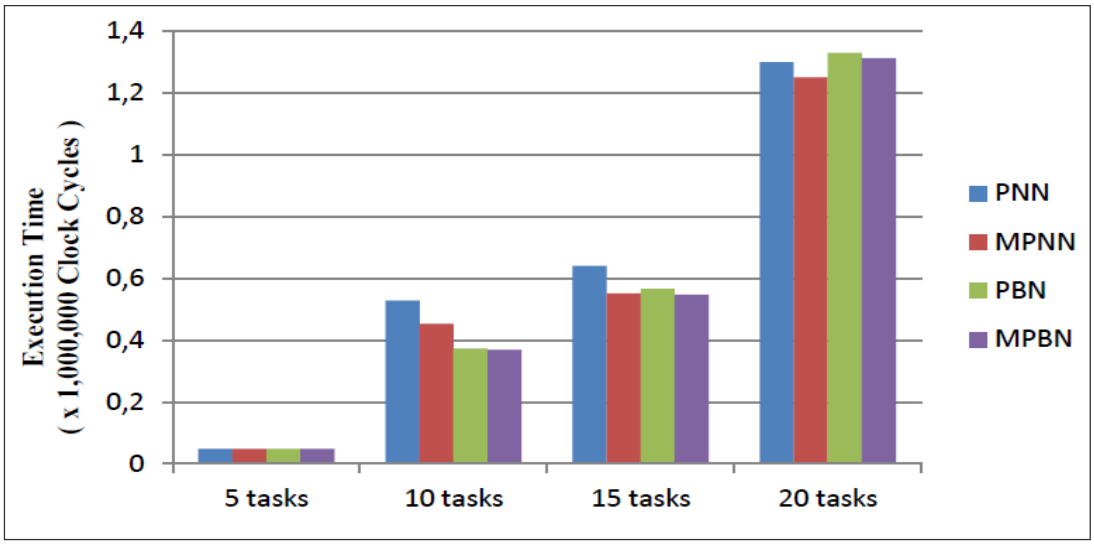

Figure 8. Execution Time of 10 applications for four applications sets (Scenario 3), where each application contains 5, 10, 15 and 20 tasks

approaches reduce the execution time with respect to existing approaches even when existing routing approach XY is employed. The reduction is more in scenario 2 than the scenarios 1 as MPEG-4 considered in scenario 2 contains higher number of tasks (13). The proposed approach will provide better results when the applications containing more number of tasks are considered.

5.2.2 Performance evaluation for large size applications

We have evaluated the performance of our approach for large size applications considered in Scenario 3. Figure 8 shows the execution times for the four application sets considered in Scenario 3. The reduction in execution time 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.

Figure 9 shows energy consumption for four application sets considered in Scenario 3. The energy has been computed in the similar way as in [3]. 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.

5.2.3 Complexity comparison: Number of searches

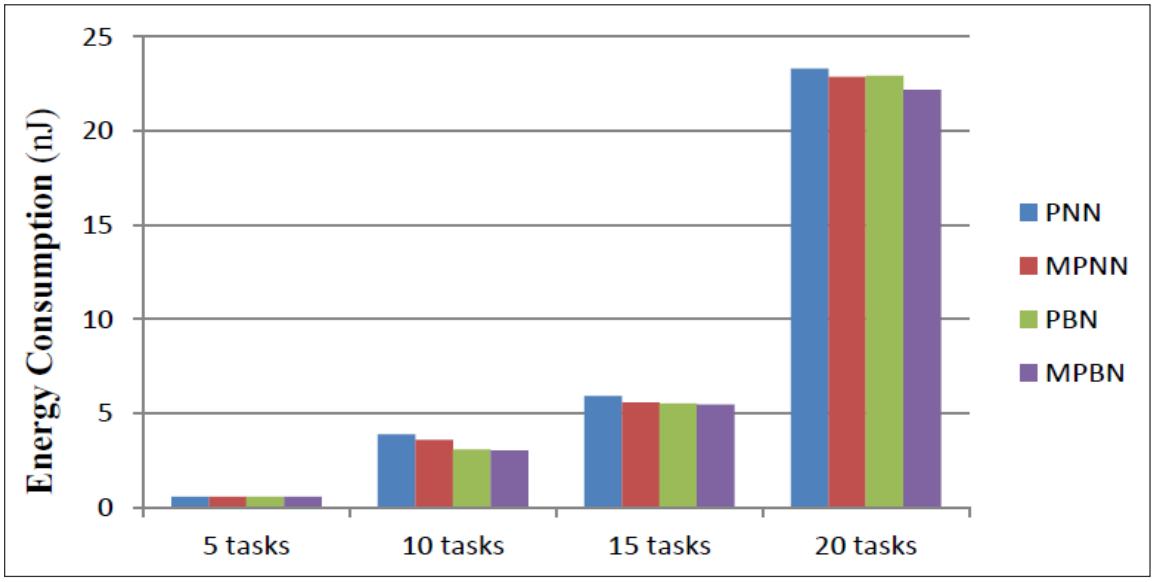

Figure 9. Energy Consumption of 10 applications for four applications sets (Scenario 3), where each application contains 5, 10, 15 and 20 tasks

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 Figure 4.

Table 1 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 1. 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.

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

Table 1. Number of searches for all the tasks in different application sets for Scenario 1 when employing existing and Manhattan strategies

6. Conclusion and Future Directions

This paper presents a 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. Experiments have shown that the proposed Manhattan-based mapping strategies 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 Plan to propose a Dynamic mapping of communicating tasks of applications to reduce the communications cost. 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] 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., 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 and al. (2009). Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs, in *SoC*.

[8] Smit and 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, in *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*.

Author Biographies

Benyamina Abbou 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 et 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.

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.

Dr Amit Kumar Singh received 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. Since February 2012, he has been working with the Department of Electrical and Computer Engineering, National University of Singapore (NUS) as a post doctoral researcher. 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 20 papers in leading related international journals/conferences.

Dr 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 (Masters degree) in 1993 and a PhD of computer science in 1996 from the École Normale Supérieure 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 the 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.