Run-Time Computation and Communication Aware Mapping Heuristic for NoC-based Heterogeneous MPSoC Platforms

Samarth Kaushik[a], Amit Kumar Singh[a], Wu Jigang[b], Thambipillai Srikanthan[a]
[a] Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore
[b] School of Computer Science and Software, Tianjin Polytechnic University, Tianjin, China
[a] *{samarth2, amit0011, astsrikan}@ntu.edu.sg*, [b]*asjgwu@gmail.com*

Abstract

The rapid increase in the complexity of real-life applications has led to the perpetual demand of refined architectural designs. Multiprocessor systems-on-chip (MPSoC) emerges as one of the possible solution for satiating such enormous computational needs. These MPSoCs are employed with Network-On-Chip (NoC) interconnect for power efficient and scalable inter-communication required between processors. Mapping parallelized tasks of applications onto these MPSoCs is the next gigantic problem, which can be done either at design-time or at run-time. However, design-time strategies may sometimes provide a more optimal mapping but they are restricted to predefined set of applications and seem incapable of run-time resource management. On the contrary, run-time mapping techniques overcome this limitation by determining the state of the platform and incorporating resource management before mapping. This paper describes a heuristic for run-time mapping of parallelized tasks of an application considering efficient computation, communication and resource utilization as the main parameters for optimization.

Keywords: **Heterogeneous architectures, Network-on-Chip (NoC), MPSoC design, mapping heuristics.**

1. Introduction

The era of employing single general purpose processors to handle entire computational complexity of an application has come to an end. Earlier, it was feasible for applications having smaller scope and less complexity but presently, due to huge advancement in every sphere of technology, the complexity of applications has surpassed the capabilities of single processor. However, advancement in Nanotechnology paved a way to fulfill this overwhelming need for computational complexity by employing concurrently working multiple processors on a single chip referred as MPSoC [1]. MPSoCs further arise need for efficient inter-communication between various processing elements (PEs), thereby, making previous communication architectures i.e. shared buses, dedicated point-to-point connections inadequate. However, Network-on-Chip (NoC) architecture is presently the most promising candidate that interconnects these PEs through a configurable mesh of on-chip connections in a power efficient and highly scalable manner [2]. The data communication among interconnected PEs is done through multiple point-to-point data links controlled by switches. MPSoCs may either consist of identical PEs referred as homogeneous or combination of different types of PEs referred as heterogeneous. NoCs can be used to realize these MPSoCs by interconnecting several types of PEs i.e. General Purpose Processors (GPPs), Digital Signal Processors (DSPs), intellectual property cores (IPs), FPGA fabric tiles, Instruction Set Processors (ISPs) and specialized memories on a single chip in order to accomplish high computation performance and energy efficiency [4]. The reported literature provides several MPSoC architecture models [18, 19]. In [18], eight floating-point units and one manager processor have been proposed for heterogeneous MPSoC by combined efforts of IBM, Toshiba and Sony.

Due to high complexity, multimedia and networking applications become the target applications for MPSoCs. These applications can be represented as a conglomeration of several computational or communication intensive tasks that are executing in parallel. Mapping these tasks onto MPSoCs in order to obtain high performance and enhanced resource utilization poses to be the next gigantic problem. Mapping can be carried out either at design-time or at run-time. Design-time mapping techniques are more dominant in the reported literature [3, 19, 20], where the decision for placement of tasks onto the platform is pre-calculated. Since, the mapping is performed without considering the current state of the platform, therefore, these techniques are not suited for dynamic workloads. Run-time mapping techniques overcome this limitation of adapting to dynamic workloads and run-time resource management by determining the state of the platform before mapping. The primary interest for most of the existing run-time mapping techniques is to reduce communication overhead and increase energy efficiency. In [16], authors propose a communication aware run-time heuristic which maps the tasks on the fly depending upon the communication requests and the load in the NoC links. Singh et al. [6, 7] propose packing strategies that try to map the communicating tasks of an application in close vicinity to each other in order to decrease the communication latencies. These mapping heuristics follow a localized approach and does not consider computation load variance as a parameter for optimization while mapping. In this paper, we present a run-time mapping heuristic for heterogeneous MPSoCs, where each PE can support multiple tasks. The MPSoC platform is similar to that described in [7], which consists of two types of PEs (Instruction Set Processors and Reconfigurable Areas). The proposed heuristic shows significant performance improvements, higher energy and resource utilization along with lower computation load variance over state-of-the-art heuristics reported in literature.

The rest of the paper is organized as follows. Section 2 provides an overview of related work. Section 3 presents the problem definition and target architecture. Section 4 describes

978-0-7695-4575-2/11 $26.00 © 2011 IEEE
DOI 10.1109/PAAP.2011.32

results are presented in Section 5. In Section 6, we conclude the paper and provide future directions.

2. Related Work

Task mapping is one of the crucial activity for obtaining high performance, low energy consumption and efficient resource utilization. Numerous design-time techniques have been reported in literature, for example, [8], [9], [10] but all of these cannot be applied for run-time mapping of multiple tasks on MPSoC platform.

Mehran et al. [5] present a run-time heuristic in which the communicating tasks are mapped close to each other by finding placement in a spiral path. This Dynamic Spiral Mapping (DSM) employs PEs arranged in 2-D mesh topology for run-time mapping of application tasks.

Briao et al. [11] present a combination of bin-packing algorithms along with linear clustering algorithms for dynamic task allocation of soft real-time applications. The energy efficiency is improved by providing a low-power mode for idle processors and reducing the operating voltage when processor's maximum performance is not required.

Schranzhofer et al. [12] presents a multiple-step strategy that employs polynomial-time algorithm which involves finding initial solutions followed by power constrained task remapping. At first, linear programming relaxation generates initial solutions for the power-aware scenario, and then the quality of the solution is improved by task remapping.

Carvalho et al. [13], presents a detailed performance analysis of five run-time mapping heuristics, First Free (FF), Nearest Neighbor (NN), Minimum Maximum Channel Load (MMC), Minimum Average Channel Load (MAC) and Path load (PL).

Nollet et al. [14] present a run-time task assignment heuristic for mapping various tasks on an MPSoC containing FPGA fabric tiles. The FPGA fabric tile enables the support for configuration hierarchy that improves the quality and success rate of task assignment. Spatial task assignment leads to efficient usage of platform resource.

Holzenspies et al. [15] propose a run-time heuristic involving four different steps that leads to optimized spatial mapping of inherently parallel streaming applications on MPSoC. The strategy involves different steps including design-time modeling, run-time feedback analysis and spatial mapping.

Singh et al. [7] describe a run-time packing heuristic that tries to reduce the communication overhead between the connected tasks by mapping them on a single PE. If a single PE cannot accommodate all connected tasks, then the heuristic attempts to map these tasks onto PEs at minimum hop distance with each other. Increasing energy efficiency and reducing the NoC average channel load forms its basis. However, the heuristic does not fully utilize the capabilities of reconfigurable areas (RAs) present in the MPSoC by sparsely executing the computation intensive tasks on them. Additionally, the heuristic results in high computation load variance among different PEs after application mapping.

3. Problem Definition

A task graph representation of an application as described in [15] has been adapted. An application has been broken down into cluster of communicating individual units, known as tasks, which execute concurrently to generate output. A directed graph $ATG = (T, E)$ represents the task graph, where T is a set of application tasks and E is the set of all edges in the application, connecting the tasks and representing their communication. Every task is considered to be executable on both software and hardware resources, having different execution time for each resource. The difference in execution times determines the suitability of a task to be executed on hardware or software resource, i.e., a hardware or software task. A task $t_i \in T$ is represented as $(t_{id}, t_{swcomp}, t_{hwcomp})$, where t_{id} is the task identifier, t_{swcomp} is the task software computation load in cycles (when the task is executed on general purpose processor) and t_{hwcomp} is the task hardware computation load in cycles (when the task is executed on reconfigurable area). An edge $e_i \in E$ represents the connection between two communicating tasks and its weight is denoted as t_{comm} as shown in Figure 1, which represents the data volume for a single token to be sent between connected tasks in terms of number of cycles taken for transfer, when full channel bandwidth is available. The communicating tasks forms a master-slave pair, where the master task in the task set T executes till last token is sent to its slave task. A slave task will start its execution once it has received a complete token from its master task. For example, t_1 is master and t_2 & t_3 are slaves in Figure 1.

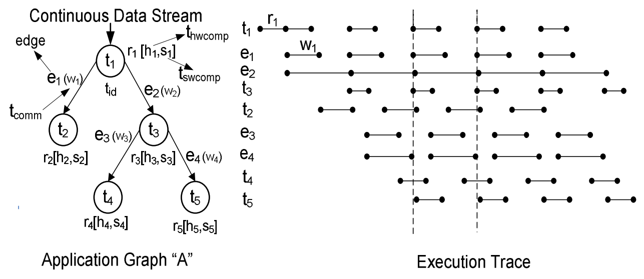

Figure 1: Execution Trace of Application Graph A

A directed graph $AG = (P, C)$ represents NoC-based heterogeneous MPSoC architecture, where P is the set of PEs identified by its identifier p_{id} and $c_{i,j} \in C$ represents the physical communication channels for interconnecting the PEs. A tile $p_i \in P$ consists of a network interface, a heterogeneous processing element, local memory and a cache.

The proposed MPSoC architecture is an extension of the architecture used in Carvalho [16]. In [16], each processing node is capable of supporting only a single task. In the proposed architecture, multiple tasks can be supported by each processing node that has fixed area & memory. Software tasks execute in instruction set processors (ISPs) and hardware tasks execute in reconfigurable areas (RAs). Inclusion of RAs in the platform provides the hardware programmability, similar to that of general purpose processors. The platform supports varying channel bandwidth but for evaluation purpose, communicating tasks have been allocated full available

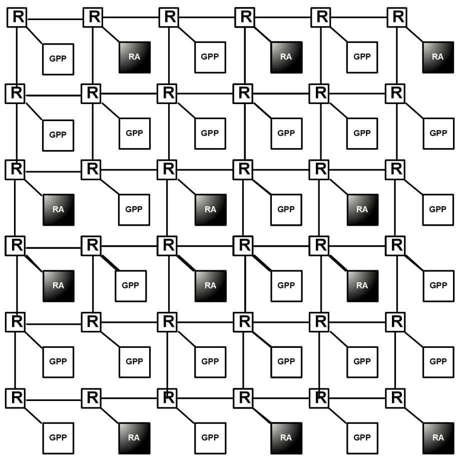

Figure 2: Our 6×6 NoC based MPSoC Architecture.

channel bandwidth. The PEs are connected in 6×6 mesh topology by a NoC as shown in Figure 2, where GPP represents general purpose processor, RA represents reconfigurable area and R represents router. However, we can consider mesh of different sizes. One of the PE is used as Manager Processor that performs task mapping, resource control, configuration and management. Task mapping is represented by function *mpg:* $t_i \in T \rightarrow p_i \in P$, that maps a task of the application to a PE in the MPSoC architecture.

4. Proposed Mapping Strategy

The proposed mapping strategy first performs the processing of the given application followed by mapping on the MPSoC platform. The processing algorithm facilitates the execution of computation intensive tasks on RAs, thereby, improving performance, energy efficiency and reduction in communication overhead. It emphasizes on the excessive utilization of RAs to their maximum capability. The proposed mapping algorithm maps the processed application on the MPSoC platform.

4.1 Algorithm

The proposed algorithm takes application task graph as an input and tries to efficiently optimize the graph before actual mapping is done on the MPSoC. The technique requires that every task can be executed on every PE (hardware or software) available in the MPSoC. Initially, the algorithm assumes that all the tasks of an application should be executed on hardware resources, i.e., RAs and hence it starts optimizing the graph considering hardware computation load of each task. The scheme starts by finding the most communication intensive edges in the graph and then tries to combine the associated tasks on the same PE. The merge is performed only when the area restriction of the corresponding PE is satisfied, i.e., the PE must have enough area to configure the logic for accommodating both tasks. This merging of tasks facilitates in the reduction of communication overhead which arises due to the transfer of data among connected tasks. This data transfer over the NoC channel not only enhances the execution time but also increases the energy consumption. Hence, removing communication bottlenecks leads to improved performance. The effectiveness of the algorithm can be observed by analyzing the execution trace of the application graph as shown in Figure 1, where r_i represents the trace for hardware computation time of each task

Algorithm 1 : Processing

```
Input: ATG(T,E)
Output: Optimized ATG(T,E)
(1) do{
(2)   Find task $t_i \in T$ having maximum hardware computation
      load ($max\_p_{hw\_load}$) from ATG (T,E).
(3)   Find edge $e_i \in E$ having maximum communication load
      ($max\_c_{load}$) from ATG (T,E).
(4)   if ($max\_p_{hw\_load} < max\_c_{load}$) then
(5)       Find hardware computation and communication
          load of connecting tasks $t_p$ & $t_q$ of the edge $e_i \in E$,
          i.e., $p_{hw\_load}(t_p)$ and $p_{hw\_load}(t_q)$
(6)       if ($p_{hw\_load}(t_p) + p_{hw\_load}(t_q) <= max\_c_{load}$) then
(7)           Merge $t_p$ and $t_q$ to a single node if their area
              requirements are satisfied on a single
              PE location and update ATG(T,E).
(8)       else
(9)           break; //goto next phase of optimization.
(10)      end if
(11)  end if
(12) while($max\_c_{load} > max\_p_{hw\_load}$)
(13) Find $t_i$ and $t_j$ with Min ($p_{hw\_load}(t_i) + p_{hw\_load}(t_j)) < max\_p_{load}$)
     from updated ATG(T,E).
(14) if ($t_i$ & $t_j$ are communicating tasks) then
(15)     Merge $t_i$ and $t_j$ to a single node if the area requirements
         are satisfied on a single PE and update ATG(T,E).
(16) else
(17)     Find $e_i$ & $e_j \in E$ having minimum communication load.
(18)     if ($c_{load}(e_i) + c_{load}(e_j) < max\_p_{hw\_load}$) then
(19)         Merge $t_i$ and $t_j$ to a single node if their area
             requirements are satisfied on a single PE and update
             ATG(T,E)
(20)     else goto (13) to find next minimum combined load.
(21) repeat steps (13) to (18) till condition at (13) is satisfied.
(22) Find task $t_i \in T$ in the updated graph.
(23) if(sum of $p_{sw\_load}$ of individual merged node) <
     $max\_p_{hw\_load}$)
(24)     Update ATG(T,E) with $t_i$ as a software resource having
         computation load = sum of $p_{sw\_load}$ of individual
         merged node.
(25) repeat steps (22) to (24) till condition at (23) is satisfied
```

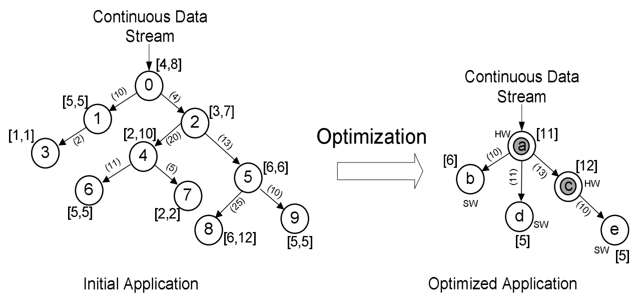

Figure 3: Optimized Application obtained after processing

$t_i \in T_i$ and w_i represents the trace for communication time of each edge $e_i \in E_i$. The figure clearly delineates the inherent parallelism that can be tapped in order to gain considerable performance improvements. For example, if tasks t_1 and t_3 are merged on a single PE, then this will remove the bottleneck e_2, thereby shrinking the execution trace and leading to lower overall execution time. The resulted optimized application graph consists of hardware computation bottleneck as the graph has been obtained considering the hardware computation time of each task. The limited number of hardware resources on the platform restricts the possibility of mapping this optimized graph onto MPSoC. Therefore, depending upon the hardware computation bottleneck, we confine the usage of the hardware resources. The tasks of the optimized graph are analyzed and considered to be executed on software resources if the total software computation time of a task does not exceed the hardware computation bottleneck obtained earlier. The resultant graph is a combination of hardware and software tasks that need to be executed on the platform. For example, Figure 3 represents the resultant graph after applying the algorithm on the given application graph. The values in brackets [] and () denotes computation load [t_{hwcomp}, t_{swcomp}] and communication load in cycles, respectively. This graph contains set of tasks at each node, like, tasks (0, 2, 4, 7) on a, (1, 3) on b, (5, 8) on c, (6) on d and (9) on e.

4.2 Mapping Algorithm

The resultant graph is then mapped on the MPSoC using the technique described in Algorithm 2. The initial node is mapped on the appropriate PE (hardware or software), as obtained from previous algorithm. The other nodes are requested to be mapped when a communication to them is required. In Figure 3, when the initial node (a) is mapped, it requests its communicating slave nodes (b, c & d) and their mapping is found on the nearest possible PE. After mapping nodes b, c and d, their slave nodes e is requested and its mapping is found.The data transfer between the nodes start when they are mapped.

Algorithm 2: Mapping

```
Input: Optimized ATG(T,E) , AG(P,C)
Output: mpg
(1) Map initial node at appropriate PE.
(2) Request communicating slave nodes.
(3) Requested nodes are mapped at minimum hop distance
     w.r.t. to their master node
(4) Repeat steps 2 to 3 till all tasks are mapped.
```

5. Experiments and Results

The experimental setup used is similar to the simulation platform described in [16]. Experiments are performed using Model-Sim co-simulation (System-C for applications and RTL-VHDL for NoC). *System-C* has been used for modelling the processing elements with the help of two threads. One thread for the Controller Processor named *CONthread* and one for the rest of the PEs named *TSKthread.* The *CONthread* manages the task configuration, task scheduling, task placement and resource management. On the other hand, the *TSKthread* describes implementation behavior of PEs in terms of computation time and communication latency for each task as presented in a configuration file.

The evaluated scenarios include multiple random, pipeline & tree like application having (i) 5 tasks, (ii) 10 tasks and (iii) 15 tasks. A 6×6 NoC-based MPSoC is considered, as modeled in Figure 2. Initial task acts as the manager of application and the PE reserved for initial task is pre-defined. We have varied the number of times an application has been executed.

5.1 Total Execution Time

The total execution time comprises of the time employed in configuration, optimization, mapping and communication overhead. The communication time dominates the total execution time and is greatly reduced by our proposed algorithm resulting in overall execution time reduction.

The comparison between total execution time when Communication-aware Nearest Neighbor (CNN) heuristic proposed in [7] and our proposed algorithm are employed is shown in Figure 4 (a). It clearly shows that our algorithm performs better as the complexity of the application increases.

5.2 Energy Consumption

The energy model as described in [7] has been employed in our simulation. Total energy consumption is the sum of communication and computation energy, i.e., energy consumed in data transfer from source PE to destination PE and the energy consumed in processing of the data at destination PE, respectively.

$$E_{total} = E_{comp} + E_{comm} \qquad (1)$$

Figure 4(b) shows that the energy consumption from the mapping obtained by CNN is higher that our proposed algorithm. Our algorithm efficiently targets the communication intensive edges and attempts to remove such overheads, thereby resulting in higher energy efficiency.

5.3 Resource Optimization

The percentage decrease in the number of PEs (hardware or software) utilized by an application on the MPSoC measures the resource optimization. Our proposed algorithm tries to efficiently decrease the number of hardware and software

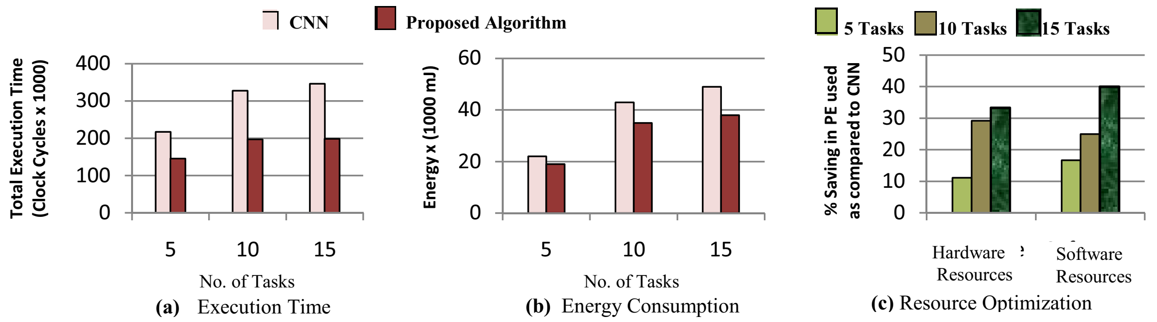

Figure 4: Overall Execution Time, Energy Consumption and Resource Optimization for CNN and Proposed Heuristics for the evaluated scenarios.

resources used by the application and employs a fair distribution of computation load among several PEs. It is observed that Figure 4(c) shows savings in resource usage by our approach when compared to CNN for different evaluated scenarios. Applications with 15 tasks show an improvement of 33.33% and 40.00% over the CNN, for hardware and software resources, respectively. Table 1 show the distribution of computation load among several PEs. It indicates that hardware resources in CNN are lightly loaded along with highly uneven distribution of load among PEs. Our algorithm results in efficient utilization of hardware and software resources when compared to CNN.

		Computation Load (in cylcles) on different Hardware/Software PEs								St. Deviation
		HW 1	HW 2	HW 3	SW 1	SW 2	SW 3	SW 4	SW 5	
10 Tasks	CNN	3	2	6	14	5	2	11	NA	4.67007
	Proposed	11	12	NA	6	5	5	NA	NA	3.42053
15 Tasks	CNN	1	5	1	15	10	2	3	3	4.98569
	Proposed	12	12	NA	4	5	8	NA	NA	3.76829

Table 1: Computation Load Distribution for Applications with 15 and 20 Tasks.

6. Conclusions

This paper details our proposed algorithm for run-time mapping of applications onto 6×6 NoC-based Heterogeneous MPSoC. The proposed heuristic facilitates the execution of computation intensive tasks on RAs providing significant improvement in total execution time, resource optimization and energy consumption when compared to state-of-the-art run-time mapping heuristic. The improvements are clearly enunciated in the experiments and results section. Our future scope includes evaluation of real-time benchmarks on the MPSoC platform and to incorporate task migration when a performance bottleneck is detected in order to improve the performance.

7. References

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

[2] J. Henkel et al., On-chip networks: a scalable, communication-centric embedded system design paradigm, in: Proceedings of VLSI Design, 2004, p.845

[3] M . Branca et al., Evolutionary algorithms for the mapping of pipelined applications onto heterogeneous embedded systems, in: Proceedings of theGen. and Evolutionary Comp., 2009, pp. 1435–1442.

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

[5] A.Mehran, A. Khademzadeh, S. Saeidi, DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip, IEICE Electronics Express 5 (13) (2008) 464–471.

[6] A.K.Singh et al., "Efficient Heuristics for Minimizing Communication Overhead in NoC-based Heterogeneous MPSoC Platforms". 2009 IEEE International Symposium on Rapid System Prototyping, pp.55-60.

[7] A. K. Singh et al., "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," Journal of Systems Architecture,vol. 56, no. 7, 2010, pp. 242-255.

[8] D. Wu, B. M. Al-Hashimi, P. Eles, Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems, in: DATE 2003, pp. 90–95.

[9] S. Murali, M. Coenen, A. Radulescu, K. Goossens, G. De Micheli, A methodology for mapping multiple use-cases onto networks on chips, in: DATE '2006, pp. 118–123.

[10] C. Marcon et al., Time and energy efficient mapping of embedded applications onto noc, in: Proceedings of ASP-DAC, 2005, pp. 33–38.

[11] E. W. Briao, D. Barcelos, F. R. Wagner, Dynamic task allocation strategies in mpsoc for soft real-time applications, in: DATE, 2008, pp. 1386–1389.

[12] A. Schranzhofer et al., Power-aware mapping of probabilistic applications onto heterogeneous mpsoc platforms, in: Proceedings of RTAS 2009, pp. 151–160.

[13] Carvalho, E.; et al. Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs. RSP, 2007.

[14] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, H. Corporaal, Run-time management of a mpsoc containing fpga fabric tiles, IEEE Trans. Very Large Scale Integr. Syst. 16 (1) (2008) 24–33.

[15] P. K. F. H¨olzenspies, J. L. Hurink, J. Kuper, G. J. M. Smit, Run-time spatial mapping of streaming applications to a heterogeneous multiprocessor system-on-chip (mpsoc), in: DATE 2008, pp. 212–217.

[16] Carvalho, E.; Moraes, F. Congestion-aware task mapping in heterogeneous MPSoCs. System-on-Chip (SoC), 2008, pp. 1–4.

[17] S. Bell et al., Tile64tm processor: a 64-core soc with mesh interconnect, in: ISSCC, 2008, pp. 88–90.

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

[19] L.-Y. Lin et al., Communication-driven task binding for multiprocessor with latency insensitive network-on-chip, in: Proceedings of ASP-DAC, 2005, pp. 39–44.

[20] L. Thiele et al., Mapping applications to tiled multiprocessor embedded systems, in: Proceedings of ACSD,2007, pp. 29–40.