**TELKOMNIKA**, Vol. 15, No. 3, September 2017, pp. 1040 ∼ 1047
ISSN: 1693-6930, accredited A by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/telkomnika.v15.i3.6513

# Application Profiling and Mapping on NoC-based MPSoC Emulation Platform on Reconfigurable Logic

**Jia Wei Tang[*], Yuan Wen Hau[**], Nasir Shaikh-Husin[*], and Muhammad Nadzir Marsono[*]**
[*]Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Johor, Malaysia
[**]IJN-UTM Cardiovascular Engineering Centre, Faculty of Biosciences and Medical Engineering, Universiti Teknologi Malaysia, Johor, Malaysia
[*]Corresponding author, e-mail: jwtang2@live.utm.my

***Abstract***

*In network-on-chip (NoC) based multi-processor system-on-chip (MPSoC) development, application profiling is one of the most crucial step during design time to search and explore optimal mapping. Conventional mapping exploration methodologies analyse application-specific graphs by estimating its run-time behaviour using analytical or simulation models. However, the former does not replicate the actual application run-time performance while the latter requires significant amount of time for exploration. To map applications on a specific MPSoC platform, the application behaviour on cycle-accurate emulated platform should be considered for obtaining better mapping quality. This paper proposes an application mapping methodology that utilizes a MPSoC prototyped in Field-Programmable Gate Array (FPGA). Applications are implemented on homogeneous MPSoC cores and their costs are analysed and profiled on the platform in term of execution time, intra-core communication and inter-core communication delays. These metrics are utilized in analytical evaluation of the application mapping. The proposed analytical-based mapping is demonstrated against the exhaustive brute force method. Results show that the proposed method is able to produce quality mappings compared to the ground truth solutions but in shorter evaluation time.*

***Keywords:*** *Embedded system, application mapping, multi-processor system-on-chips (MPSoCs), throughput constraint.*

## 1. Introduction

Multi-Processor System-on-Chips (MPSoCs) is commonly used in modern embedded systems such as smartphones and tablets as well as in other diverse fields such as multimedia and networking applications to perform time-constrained tasks. Mapping of time critical applications is a crucial step in design time to provide the highest possible system performance. The problem of mapping applications onto MPSoC is one of the biggest challenge embedded system programmers face nowadays [1–3]. An application is commonly specified as a data-flow graph consisting of tasks that communicate between them, on the other hand, an MPSoC platform is described by the number of cores, the type of cores (in heterogeneous platform), and their communication topology or infrastructure. The design space exploration process takes the application graph as input and assigns all tasks to processing cores on the desired MPSoC platform to find the best performance mapping.

Methods for evaluating each mapping performance in design space can be categorized into three types [4]: estimation from analytical model, measurement from simulation model, and measurement from implementation prototype. Analytical estimation is the fastest evaluation method but with limited accuracy as it does not sufficiently replicate the system behaviour. Simulation-based measurement exists between two extremes, i.e. slow with high accuracy and fast but less accurate. On the contrary, prototype-based evaluation provides the highest accuracy but requires long development time.

*Received February 27, 2017; Revised June 28, 2017; Accepted July 21, 2017*

In most cases, a certain application is desired to be executed on a specific MPSoC platform. As the task execution and communication delays vary for different MPSoC platforms, mapping that solely depends on the costs for executing application specification behaviour may not be sufficient to produce high quality mapping when considering the MPSoC physical interconnect fabric. The behaviour on the platform must be considered and profiled thoroughly to aid application mapping exploration process. Hence, an application profiled in a prototype-based platform can improve profiling precision over the analytical-based mapping while having fast evaluation speed.

This paper presents a application mapping methodology that utilizes field-programmable gate array (FPGA) emulation as the evaluation platform to optimize the application mapping for the highest throughput. Applications are analysed and profiled on the emulation platform to estimate their implemented costs in term of execution, intra-core communication and inter-core communication delays. The profiled application costs are used in improving the accuracy of analytical-based mapping. The proposed methodology is demonstrated by mapping several test applications on a 3$\times$3 meshed network-on-chip (NoC) based MPSoC prototyped in FPGA.

The rest of this paper is organized as follows. Section 2. presents the related works in application mapping. Section 3. discusses the proposed throughput aware application mapping methodology including the execution trace analysis, platform parameters extraction and mapping exploration steps. Section 4. presents the result of the proposed mapping approach on different test cases and Section 5. concludes this paper.

## 2. Related works

Task mapping technique can be classified into design time mapping, run-time mapping and hybrid mapping. In design time mapping [5–9], the mapping process is performed off-line to facilitate better decision in using system resources. Design time mapping is suitable for static workload scenarios having a predefined task graph and static platform (i.e., both applications and platform do not change in time). Run-time mapping [10–13] on the contrary, is able to handle dynamism of run-time workload such as insertion or removing new applications into or from the system, as well as the dynamic platform behaviour due to run-time faults. However, run-time task mapping is performed using limited on-line processing resource while making a quick decision without sufficient mapping exploration, thus may produce low-quality mapping.

In addition, hybrid mapping has also been proposed in many different works [14–19] by combining design time and run-time techniques. Hybrid mapping explores best mapping of tasks during design time for different scenarios (constraints conditions) while chooses the most suitable mapping at run-time. However, these conventional approaches in application mapping assume a specific well-defined program behaviour, often specified in task graph and does not consider platform dependent parameters, such as the changes in communication delay and communication volume during dynamic mapping exploration.

Most mapping methodologies in literatures e.g. [5–13] assumed a specific program execution while the communication cost was considered constant across platform. As practical behaviour of applications may differ compared to its specified task graph, there are different methods in literatures extracting application run-time metrics prior to the mapping exploration. Trace analysis technique was used in [20] to find the execution behavior for each mapping and examine their differences. Singh et al. [21] used the application behaviour from trace analysis to optimally map multiple applications on the same MPSoC. To our best knowledge, there is no work that profiles and maps application based on execution, intra-core and inter-core communication on an emulation platform.

## 3. Proposed Application Mapping Methodology

The flow of the proposed application mapping approach is depicted in Figure 1. Unlike conventional mapping methodology which solely depends on the costs specified in application task graph, the proposed approach considers the application practical behaviour to better estimate the mapping performance. The proposed methodology for application mapping are as follows:

1. **Application Profiling** – The predefined application are executed on the targeted platform to measure its practical implementation costs. Application are profiled in term of execution and communication costs in time delay (cycles).
2. **Application Mapping** – Application mapping design space exploration is performed to optimize overall throughput. In order to evaluate each mapping performance, execution traces are analysed. An analytical model is formulated based on the profiled application costs are used to estimate the mapping performance during the design space exploration.

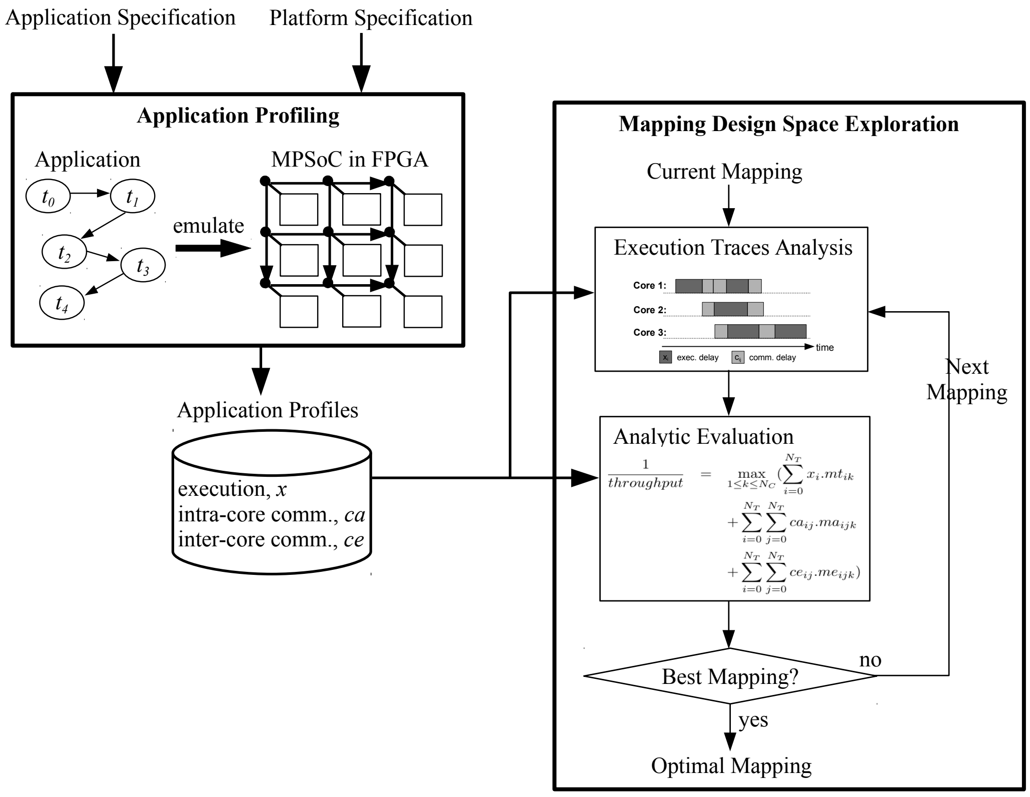


Figure 1. Proposed application mapping using application profiled in FPGA-emulated MPSoC.

## 3.1. Application and Platform Model

Task graph of an application can be characterized by a directed graph (DG), $G = (T, E)$, where $T = \{t_1, t_2, ..., t_n\}$ is the set of tasks in the application and $E = \{e_1, e_2, ..., e_n\}$ is the set of directed edges representing dependencies of tasks as shown in Figure 2. Each task, $t_i$ contains an $x_i$ denoting the worst-case execution time for the task and is assumed to remain fixed during the execution. Each edge, $e_k$ includes $v_{ij}$ which represents the communication volume between tasks $t_i$ and $t_j$.

As designing the entire NoC-based MPSoC platform is out of scope of this work, the employed platform is generated using ProNoC tool [22]. The overview of the NoC-based MPSoC platform used in this work is depicted in Figure 3. Figure 3a illustrates the architecture for each core (tile) in the MPSoC platform. Distributed memory architecture is implemented, where each core consists of a processor, a local memory, and a network interface (NI). Mapped tasks in each core are stored and executed from the local memory, while cores inter-communicate with other cores via network interfaces. All cores are homogeneous and connected to each other through the NoC interconnect fabric.

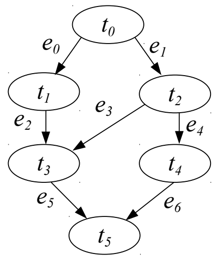
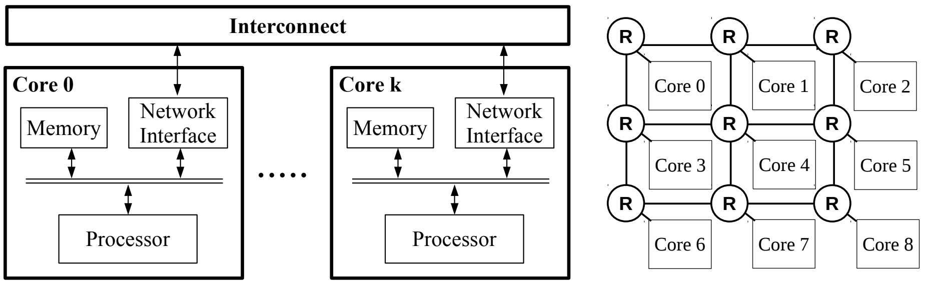


Figure 2. Example of an application model containing 6 tasks.



(a) Each core consists of a processor, local memory, and network interface in MPSoC platform model

(b) 3×3 mesh based NoC interconnect.

Figure 3. MPSoC platform model for profiling, mapping and emulating application in FPGA.

Mapping of an application is the process of assigning its tasks into the available cores on MPSoC platform. Cooperative (non pre-emptive) multi-tasking is employed in each core, where one core can execute more than one task. Communicating tasks use message passing interface (MPI) protocol to send and receive data and control packets. Hence, communication between tasks that are mapped to the same core also require message copying delay that is considered as intra-core communication. On the other hand, inter-core communication is defined as the communication between tasks that are mapped in different cores.

## 3.2. Application Profiling

Application profiling is the process to extract practical task execution and communication delays of an application when emulated in FPGA. The aim of performing application profiling is to measure the effect of task execution and communication on the overall system throughput. Execution time, $x_i$ of each task $t_i$ is defined as the time required to complete a task after it receives complete data (or token) during one execution period. Task execution time of an application can be easily obtained by capturing the time difference between the starting and ending of a task's execution. As the execution time on any homogeneous core is equal regardless of the mapping, the maximum execution delays of all tasks can be recorded by mapping all tasks into one core.

On the contrary, communication time, $c_{ij}$ is divided into two types represented by a two-tuple $< ca, ce >$, with $ca$ and $ce$ denote the intra-core and inter-core delays, respectively. As application as specified in a task graph contains only communication volume in each connecting task, profiling each connecting core has to be done to measure the practical communication time. In order to profile each intra-core communication delay, the two connecting tasks are isolated and executed in the same core. The intra-core communication delay is measured as the throughput difference between executing both tasks independently without communication and with communication. On the contrary, both tasks are mapped in neighbouring cores to profile inter-core communication delay. After profiling, the application graph is re-constructed with profiled specification as illustrated in Figure 4. Although the actual delays of the applications may differ for different mappings (maybe due to the multi-task overhead or communication infrastructure delay such as contention), the profiled costs can be used as the basis to estimate the application run-time behaviour.

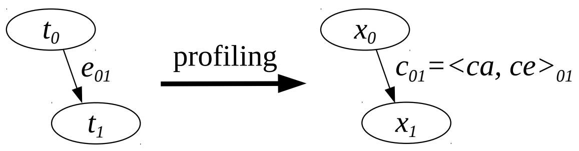


Figure 4. Application graph re-construction after profiling execution and communication time on emulated FPGA.

### 3.3. Mapping Design Space Exploration

After application profiling, analytical-based mapping utilizes the profiled costs to optimize the application throughput. Execution traces of each mapping is analysed to find the estimation of application run-time behaviour. Each mapping is evaluated using analytical expression to calculate the throughput using the profiled costs. Mapping exploration process is iterative until the best mapping is obtained.

#### 3.3.1. Execution Traces Analysis

Execution traces of a mapped applications are analysed to obtain the application throughput. Task parallelism and temporal parallelism are considered in the execution traces analysis. The former is the ability to allow parallel execution of different tasks while the latter allows parallel execution of different temporal data in different cores at the same time.

For instance, Figure 5 depicts the execution traces of an application having 6 tasks, as shown in Figure 2, mapped onto a 4-core system. All 4 cores are communicating with each other while executing their assigned tasks in parallel as soon as their data are available. Task parallelism occurs for task $t_1$ and $t_2$, taking $x_1$ and $x_2$ delays to be executed in core 1 and core 2, respectively. Data parallelism is demonstrated through the execution of different temporal data (e.g data 1 and data 2) at different time instances. As core 4 requires the longest time to compute one result, the period (inverse of throughput) is determined by the combination of execution delay and communication delay.

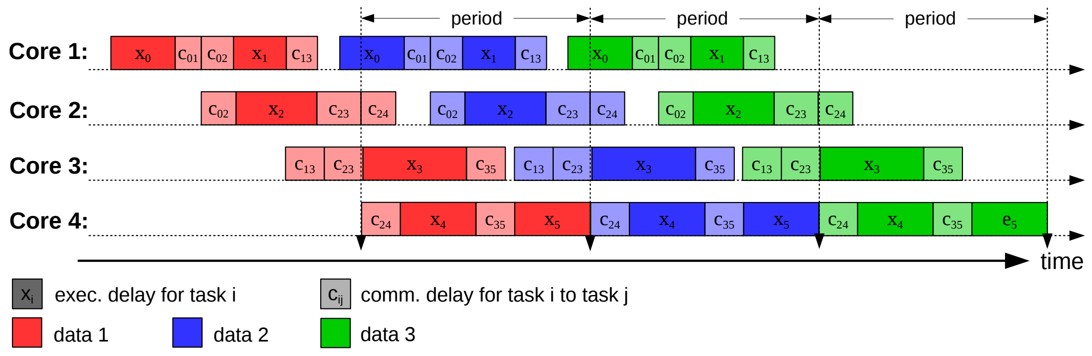

period
period
period
Core 1:
Core 2:
Core 3:
Core 4:
time
$x_i$ exec. delay for task i
$c_{ij}$ comm. delay for task i to task j
data 1
data 2
data 3

Figure 5. Execution traces of a 6-tasks applications running on 4 cores system.

Application throughput is reciprocal of one period (i.e, the average time needed to complete one iteration of the application). Each core utilizes the time period to execute its assigned task as well as to communicate with other cores. Since the tasks in each application can be executed in pipeline and task-parallel in different cores, the overall throughput is determined by the slowest core (i.e, the core that requires the longest time to complete executions and communications of all its mapped tasks).

### 3.3.2. Analytical Evaluation

Given a graph representation of an application with $N_t$ tasks, mapping exploration in this work is the process of assigning all tasks into available $N_c$ cores to maximize the system throughput. Based on the execution trace analysis, throughput of a mapped application is the reciprocal of period. The period is defined as the maximum total required time (among all cores) for execution and communication, and can be mathematically formulated in Equation 1. The total required time in each core is calculated by the summation of the profiled execution delays of all individual assigned tasks as well as all the profiled intra-core and inter-core communication delays associated to it.

$$\begin{aligned}\frac{1}{throughput} &= period \\ &= \max_{1\le k\le N_C}(X_k + C_k) \\ &= \max_{1\le k\le N_C}(\sum_{i=0}^{N_T} x_i.mt_{ik} + \sum_{i=0}^{N_T}\sum_{j=0}^{N_T} ca_{ij}.ma_{ijk} + \sum_{i=0}^{N_T}\sum_{j=0}^{N_T} ce_{ij}.me_{ijk}) \qquad (1)\end{aligned}$$

where $i$ and $j$ are task identifiers while $k$ is the core identifier. $X_k$ denotes the total execution delays of mapped tasks in core $k$, which is the summation of the profiled execution delay $x_i$ if task $t_i$ is mapped to core $k$ ($mt_{ik} = 1$). $C_k$ denotes the total communication delays associated to core $k$, which is the summation of all profiled intra-core communication delay $ca_{ij}$ if intra-core communication of task $t_i$ and $t_j$ exist in core $k$ ($ma_{ijk} = 1$), and all profiled inter-core communication delay $ce_{ij}$ if inter-core communication of task $t_i$ and $t_j$ is associated to core $k$ ($me_{ijk} = 1$).

## 4. Results and Discussion

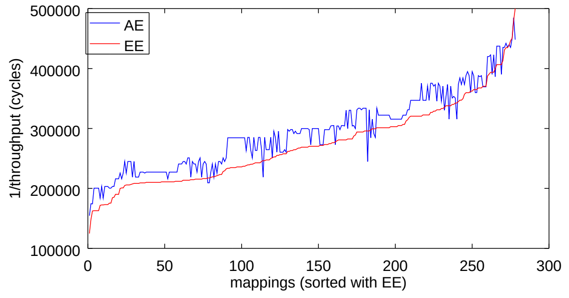
The proposed analytical-based mapping is demonstrated by employing the exhaustive brute force method to map several random application graphs and find the mapping that produces the best application throughput. The best mapping obtained by the proposed analytical-based exhaustive (AE) are compared with emulation-based exhaustive (EE) solutions (i.e., ground truth) which is obtained by executing all mappings on a FPGA emulation system.

Figure 6a shows the throughput for all possible mapping for the proposed analytical-based application mapping and emulation-based mapping. The proposed analytical-based mapping produces results with similar trend with the ground truth solution. Figure 6b depicts the comparison between emulation-based throughput and analytical-based solution. The proposed analytical-based mapping throughput is almost similar with emulation-based throughput as the points scatter nearby the straight line of gradient one.

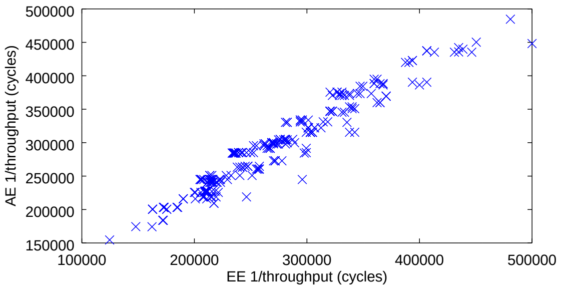
Table 1 illustrates the comparison between proposed analytical-based and emulation-based exhaustive (ground truth) for application of different random graphs. The best throughput (BT) is obtained by emulating the best mapping produced by both methods on FPGA. The evaluation time (ET) is the time taken to perform the exhaustive evaluation on all possible mappings while profiling time is the time taken to profile the tasks based on the proposed method. Results illustrate that the evaluation time for both exhaustive mapping increases exponentially with the number of possible mappings. The proposed analytical-based method is able to evaluates all mappings in much shorter time despite requiring extra profiling time. However, the profiling time does not increase with number of mappings as it corresponds to the number of profiles required, that is the total number of tasks and edges in application graphs.

## 5. Conclusion

Efficient mapping algorithm produces high quality solution in fast evaluation time. This paper presents a application mapping methodology that utilizes FPGA emulation as an evaluation platform to optimize the application throughput. Applications are analysed and profiled in the



(a) Throughput of all possible mapping for the proposed analytical-based exhaustive (AE) and emulation-based exhaustive (EE) mapping (sorted based on EE).



(b) Emulation-based versus analytical-based exhaustive throughput.

Figure 6. Throughput comparison between emulation-based exhaustive (EE) and proposed analytical-based exhaustive (AE) mapping in design space exploration of a 7-task application.

Table 1. Comparison in throughput and evaluation time between proposed analytical-based (AE) and emulation-based exhaustive (EE) for application of different random graphs

| Applications | No. of Tasks | EE | | Proposed AE | | |
|---|---|---|---|---|---|---|
| | | BT | ET | BT | PT | ET |
| random1(r7) | 5 | 87 | 607.61 s | 87 | 252.21 s | 0.03 s |
| random2(r8) | 6 | 401 | 2912.33 s | 401 | 296.10 s | 0.15 s |
| random4(r2) | 7 | 106 | 15745.43 s | 106 | 309.34 s | 0.96 s |
| random5(r4) | 7 | 875 | 16143.52 s | 875 | 276.82 s | 0.91 s |

BT: Best Throughput ET: Evaluation Time PT: Profiling Time

emulated environment to obtain their practical implemented costs in term of execution, intra-core communication and inter-core communication delays. Application mapping utilizes the profiled costs find optimal mapping through execution trace analysis and analytical evaluation. Results show that the proposed analytical-based exhaustive based on the profiling can produce similar best throughput compared to the ground truth solutions but in shorter evaluation time.

**Acknowledgement**

This work is supported in part by the Collaborative Research in Engineering, Science & Technology (CREST) grant P17C114 (UTM vote no. 4B176) and Universiti Teknologi Malaysia Matching grant (UTM vote no. 00M75).

## References

[1] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, “Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3–21, 2009.

[2] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, “Mapping on multi/many-core systems: survey of current and emerging trends,” in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 1.

[3] P. K. Sahu and S. Chattopadhyay, “A survey on application mapping strategies for network-on-chip design,” *Journal of Systems Architecture*, vol. 59, no. 1, pp. 60–76, 2013.

[4] R. Piscitelli and A. D. Pimentel, “Design space pruning through hybrid analysis in system-level design space exploration,” in *Design, Automation & Test in Europe Conference & Exhi-*

*bition (DATE), 2012*. IEEE, 2012, pp. 781–786.

[5] J. Hu and R. Marculescu, "Energy-and performance-aware mapping for regular noc architectures," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 24, no. 4, pp. 551–562, 2005.

[6] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 407–420, 2006.

[7] S. Tosun, "Cluster-based application mapping method for network-on-chip," *Advances in Engineering Software*, vol. 42, no. 10, pp. 868–874, 2011.

[8] P. K. Sahu, T. Shah, K. Manna, and S. Chattopadhyay, "Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 2, pp. 300–312, 2014.

[9] Y. Z. Tei, Y. W. Hau, N. Shaikh-Husin, and M. N. Marsono, "Network partitioning domain knowledge multiobjective application mapping for large-scale network-on-chip," *Applied Computational Intelligence and Soft Computing*, vol. 2014, p. 9, 2014.

[10] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for mpsocs," *IEEE Design & Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.

[11] A. K. 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, no. 7, pp. 242–255, 2010.

[12] M. Fattah, A.-M. Rahmani, T. C. Xu, A. Kanduri, P. Liljeberg, J. Plosila, and H. Tenhunen, "Mixed-criticality run-time task mapping for noc-based many-core systems," in *2014 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2014, pp. 458–465.

[13] T. D. Ngo, K. J. Martin, and J.-P. Diguet, "Move based algorithm for runtime mapping of dataflow actors on heterogeneous mpsocs," *Journal of Signal Processing Systems*, pp. 1–18, 2015.

[14] S. Stuijk, M. Geilen, and T. Basten, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*. IEEE, 2010, pp. 548–555.

[15] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2012, pp. 71–80.

[16] C. Lee, S. Kim, and S. Ha, "Efficient run-time resource management of a manycore accelerator for stream-based applications," in *2013 IEEE 11th symposium on Embedded systems for real-time multimedia (ESTIMedia)*. IEEE, 2013, pp. 51–60.

[17] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for mpsocs," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 131.

[18] A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1, p. 9, 2013.

[19] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous mpsocs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, p. 14, 2015.

[20] A. Goens and J. Castrillon, "Analysis of process traces for mapping dynamic kpn applications to mpsocs," in *IFIP International Embedded Systems Symposium (IESS)*, 2015.

[21] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Resource and throughput aware execution trace analysis for efficient run-time mapping on mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 72–85, 2016.

[22] ProNoC. (2016) Prototype NoC based MPSoC. Opencore. [Online]. Available: http://opencores.org/project,an-fpga-implementation -of-low-latency- noc-based-mpsoc