Comparative Analysis of Dynamic Task Mapping Heuristics in Heterogeneous NoC-based MPSoCs

Leandro Möller[1], Leandro Soares Indrusiak[2], Luciano Ost[3], Fernando Moraes[4], Manfred Glesner[1]

[1] Darmstadt University of Technology - Institute of Microelectronic Systems - Darmstadt, Germany
[2] Department of Computer Science - University of York - York, United Kingdom
[3] LIRMM – University of Montpellier II, France
[4] Faculty of Informatics - Catholic University of Rio Grande do Sul - Porto Alegre, Brazil

Abstract — **Dynamic mapping heuristics can cope with dynamic application scenarios by allocating tasks to cores of an MPSoC during runtime. In this paper, we compare eight heuristics in terms of the response time of application tasks - that is, the time between the issuing of a task and the time when it completes executing and communicating. By taking into account the task execution, communication and waiting times, we could better evaluate the quality of the different heuristics and show that there is room for improvement when it comes to heterogeneous platforms under high utilization.**

Keywords — multiprocessor systems-on-chip, networks-on-chip, embedded systems, dynamic task mapping.

I. INTRODUCTION

Applications running on Multiprocessor Systems-on-Chip (MPSoCs) may vary dynamically at execution time, according to user (e.g. load of new applications) and/or performance (e.g. change the frequency operation for optimizing battery lifetime) requirements, which leads in both time-changing processor workload and communication patterns [1][3][4]. Thus, offline-mapping techniques can be sub-optimal or inadequate in many scenarios. In this context, dynamic task mapping techniques have been used to achieve the required runtime adaptability demanded by such multiprocessing systems [6][7]. Such dynamic task mapping techniques are evaluated in both homogeneous and heterogeneous platform architectures.

More than avoiding congestion and placing communicating tasks near to each other, heterogeneous MPSoCs need to care about the affinity of tasks with the IP cores available on the platform. This is only true when the same task is developed for different IP cores and trading efficiency against utilization of these cores is left for the system to balance. The result is then a computing system that can analyze its own resources and allow the use of them in a more optimized manner. Therefore, smart implementations of dynamic mapping algorithms are vital for the MPSoC to execute applications with good performance figures and using as few resources as possible.

Our contribution is to evaluate quantitatively and comparatively dynamic task mapping using the affinity of tasks to the IP cores available on the heterogeneous MPSoC. Some of the presented algorithms are multi-objective, considering not only the affinity of the task and the congestion of the network, but also the utilization of the IP cores, the position on the network and the amount of communication among tasks.

This work is divided as follows. Section II presents the state of the art on dynamic task mapping for MPSoCs. The joint validation model composed by platform, application and task mapper used on this work is presented on section III. Section IV presents the dynamic mapping algorithms compared on this work. Section V presents the case studies and obtained results. Section VI concludes this work.

II. STATE OF THE ART

Examples of dynamic task mapping techniques explored in homogeneous architectures are [4][6][9]. In turn, dynamic task mapping on heterogeneous MPSoC platforms are investigated in [1][2][3][5][8][10][11]. Due to the distinct nature of processing elements (PEs) that can be integrated in such platforms, the mapping process is more complex when compared to the homogeneous case because additional constraints (e.g. the affinity of the task to a PE) must be considered at run-time. In this context, Carvalho et al. [1] proposed and evaluated the performance of six mono-task mapping heuristics considering different application workloads. Some of these heuristics were extended to consider multi-tasks mapping onto the same PE, while minimizing the commutation overhead in the same NoC-based platform [2]. Singh et al. [2] also proposed new heuristics that consider the power consumption as the product of number of bits to be transferred and distance between source-destination pair.

Faruque et al. [3] present a distributed agent-based mapping scheme. The proposed scheme divides the system into virtual clusters. A cluster agent (CA) is responsible for all mapping operations within a cluster. Global agents (GAs) store information about all the clusters of the NoC and use a negotiating policy with CAs in order to define to which cluster an application will be mapped. Another distributed approach is proposed in [4], which explores different implementations of a decentralized self-embedding algorithm, aiming to minimize network contention and latency while providing fault-tolerance support for NoC-based systems.

III. JOINT VALIDATION MODEL

Määttä et al. present in [12] the joint validation of an application mapped onto different platform models based on NoCs. This validation model enforces the use of a well-defined API among the main layers of the MPSoC: application, mapper and platform. An **application** is modeled by any number of concurrent tasks that communicate by explicitly exchanging

978-1-4673-2896-8/12/$31.00

messages. The communication dependencies between tasks are modeled using directed graphs with tasks represented by the nodes and messages by the edges. Each task is characterized by its computation time, and each inter-task communication message is characterized by its source and destination tasks, and its data volume. Many different models of computation can be used to further describe the concurrent behavior of tasks and messages. In this paper, we reuse the model described in [12], where messages are sent by a task only when they finish execution, and tasks can only be triggered by a timer or a predefined combination of messages (which may have to arrive in a predefined sequence). The **mapper** is responsible to map tasks to PEs on the platform. In turn, a **platform** is composed of PEs interconnected by a NoC. When tasks are triggered, the PEs onto which they are mapped are made busy for the task's execution time (or it enters a scheduling queue in case the PE was already busy). Once a task has finished its execution, the PE sends the respective messages to the NoC, which simulates its transmission towards the core where the destination task is mapped. The latency of the task execution on the PEs and the flit-by-flit message transmission over the NoC is then back-annotated to the application model, allowing for an accurate estimation of the response time of each task of the system, taking into account the contention for the PEs as well as NoC links and routers.

In this work, we extend all three layers described above: application, platform and mapper. One extension is that the PEs of the platform are now multi-tasking, using an Earliest Deadline First scheduling algorithm. Another extension that involved all three layers was to support the heterogeneity of the MPSoC. This involves setting the type of each PE (e.g. CPU1, DSP1, CPU2, …) and setting for each task of the application which are the PEs that can execute it (i.e. the system contains the object code of the task for a certain PE). We also extended the characterization of application tasks by modeling the interplay between computation time and affinity. C**omputation time** denotes how long does it take for a task to execute all its functionality. Sometimes the computation time may depend on the inputs of the task. In such cases, it is common in real time systems to define it as the worst case computation time to guarantee that the task will always be able to execute without missing the deadline. **Affinity** is measured in percentage and it is a multiplicative factor to increase or reduce the computation time depending on which PE the task is mapped. Every task must have its computation time defined in relation to the PE over which the task has greater affinity. So, the computation time of a task m mapped on a PE k (CT_{mk}) can be calculated by

$$CT_{mk} = \frac{CT_m}{Af_{mk}} \quad (1)$$

where CT_m is the computation time of the task m when mapped on a PE which has 100% affinity and Af_{mk} is the affinity of the task m to the PE k.

IV. Dynamic Mapping Algorithms

Carvalho et al. [1] compare six dynamic mapping algorithms. The **First Free (FF)** simply selects the next compatible IP core to map a given task, thus walking sequentially through all IP cores before considering an IP core again. **Nearest Neighbor (NN)** considers first the IP cores located near to the requesting task, and it maps the target task on the first compatible IP core found. **Minimum Maximum Channel load (MMC)** considers all possible mappings for a given task and chooses the one that increases the least the peak load of a channel of the NoC. **Minimum Average Channel load (MAC)** considers all possible mappings for a given task and chooses the one that increases the least the average load of the channels of the NoC. **Path Load (PL)** considers all possible mappings for a given task and chooses the one that increases the least the sum of the load of the channels between the requesting task and the target task. **Best Neighbor (BN)** considers first the IP cores located near to the requesting task, and if there is more than one candidate mapping at the same hop distance from the requesting task, the best alternative is selected according the PL algorithm.

Two dynamic mapping algorithms were developed in the frame of this work. **Minimum Data Exchange (MDE)** considers all possible mappings for a given task and computes for each of them the total amount of data that must be sent/received by the already mapped tasks. The PE with less communication load receives the target task. If more than one PE returns the same communication load (very likely to happen in the beginning of the execution of the system), the PE with minimum hops distance to the requesting task is selected. If again there is more than one candidate PE, the first candidate of an array of final candidates is selected. The **Cost Based (CB)** dynamic mapping algorithm considers all possible mappings for a given task and chooses the one with minimum cost according to the following equation

$$Cost = \frac{U_k \times H_{st} \times L_{st}}{Af_{tk}} \quad (2)$$

where U_k is the current utilization of the PE under consideration for mapping k, H_{st} is the number of hops between the source task s and the target task t (considering t mapped on k), L_{st} is the load between s and t measured by the amount of bytes exchanged by them, and Af_{tk} is the affinity of the target task t to k. The utilization of a PE can be calculated by

$$U_k = \sum_{m=0}^{q} \frac{CT_{mk}}{P_m} \quad (3)$$

where CT_{mk} is the computation time of task m when mapped to the PE under consideration k, P_m is the period of task m and q is the number of tasks mapped onto k.

Table 1 presents the metrics used by the cost functions of the dynamic mapping algorithms introduced on this work. FF is for sure the fastest algorithm, since it requires only to find the next compatible PE for a task. NN is also fast and tries to put the communicating tasks near to each other. BN comes next in terms of speed since it searches for possible PEs according to NN and only uses PL when more than one candidate PE is found. All the other algorithms consider all PEs for making a mapping decision, therefore, they become slower with the increase of the number of PEs. On the other hand, other algorithms can consider the channels of the NoC and the communication load of the tasks for preventing congestions. The computation load of the tasks mapped on a PE is also an

important metric for avoiding the overload of the PE, and is considered by the CB algorithm.

All dynamic mapping algorithms, except the FF, require a requesting task to perform their mapping decision accurately. As the tasks that start the application do not dispose of a requesting task and such a decision can affect all subsequent decisions of the dynamic mapper, two initial mapping algorithms were developed to deal with this situation. One is the FF that was already presented, and the other is the **Cluster (CL)** initial mapping algorithm. This algorithm divides the PEs in clusters and maps each initial task to a different cluster. The size of the cluster depends on the amount of initial tasks and PEs the system contains. The goal of this algorithm is to separate the initial tasks, allowing tasks mapped later to be near to their corresponding initial tasks.

Table 1. Metrics used by the cost functions of different dynamic mapping algorithms used on this work.

	Network position	Channel load	Comm. load	Task affinity	PE utilization
FF	✓				
NN	✓				
MMC	✓	✓			
MAC	✓	✓			
PL	✓	✓			
BN	✓	✓			
MDE	✓		✓		
CB	✓		✓	✓	✓

V. CASE STUDY

In order to evaluate the different dynamic mapping algorithms, one synthetic application was developed and executed over a 4x4 and a 5x5 heterogeneous platforms based on the HERMES NoC [12]. These heterogeneous platforms are configured with 2 DSPs, one on the upper left corner and another on the lower right corner. All the other PEs are GPPs, and both platforms reserve the PE on the lower left corner for the mapper. The application is composed by 30 tasks, where 12 use the initial mapper and 18 are dynamically mapped. These 30 tasks are used and reused by a total of 15 task graphs which describe different functionalities of the application. The computation time of the tasks range from 1,000 to 70,000 simulation cycles and the period of the tasks range from 200,000 to 500,000 simulation cycles. Each simulation cycle corresponds to one clock cycle of the real platform. The message sizes communicating the tasks range from 1,000 to 50,000 bytes. Six tasks are compatible to the 2 DSPs available on the platform, where these tasks have an affinity of 100% to them, while they have an affinity of 20% to the GPPs. Each platform-mapper combination was simulated for 10,000,000 simulation cycles and every time a task is mapped to a certain PE it remains there until the end of the simulation.

Figure 1 presents boxplots with the response times of 5 task graphs of the application mapped to the 4x4 platform (graphs A-E) and the 5x5 platform (graphs F-J). The numbers 1 and 2 used on the label of the plots indicate respectively the use of the CL and FF initial mappers. The plots aligned on the same row indicate the task graph. Each plot presents on the X-axis the dynamic mapping algorithm used by the application. The Y-axis refers to the response time of a complete task graph (i.e. the time between triggering the execution of the first task and the time when the last task finishes executing and communicating). Boxplots show minimum, lower quartile, median, upper quartile and maximum response times of all executions of each task graph, measured on simulation cycles.

On the first glance to any of the graphs, it is possible to see that the worse response time of a task graph can be more than the double of the best response time, indicating that the mapping algorithm really influences the timing of the application. One expectation was that the dynamic mapping algorithm CB would always present better timing results, since it considers both communication and computation of the application. However, CB provided the best timing results on only 40% of the cases presented on

Figure 1. While it is enough to know that the CB was the dynamic mapping algorithm that performed better on most of the times, it cannot be forgotten that the CB is also the one that costs more in terms of communication overhead (i.e. information about communication load, task affinity and PE utilization, which are information that need to be transmitted through the network from all PEs to the mapper). This costs in terms of time and network usage for transferring this mapping information is currently not considered on this work.

Another expectation was that the initial mapping algorithm CL would always present better timing results, since it creates clusters to keep the initial tasks of different task graphs apart from each other, thus giving space for the subsequent tasks mapped with the dynamic mapping algorithm to be grouped together with their corresponding initial task. While this is true and visible when compare graph G1 with G2 and graph H1 with H2, the CL was only better than FF on 42.5% of the cases on the 4x4 platform and 82.5% of the cases on the 5x5 platform.

VI. CONCLUSIONS

Eight dynamic mapping algorithms for heterogeneous MPSoCs were compared on two platform configurations. These dynamic mapping algorithms consider different cost functions like the position of the task on the platform, the congestions of the network, the amount of data transmitted by the tasks, the affinity of the tasks to the different types of PEs available on the platform and the utilization of the PEs. Hence, the timing characteristics of computation and communication of the application and platform were taken into account on the presented case studies.

From the results obtained we could see that in most of the cases the variance w.r.t. the timing to execute the application task graphs is enormous. This is mainly due to the fact that the baseline HERMES NoC is employed, which does not contain any mechanism that can enable Quality-of-Service (QoS) guarantees. The results also showed that the initial mapper CL was better than the FF on 42.5% of the times over the 4x4 platform and 82.5% of the times over the 5x5 platform. The dynamic mapping algorithm CB provided the best timing results on 40% of the cases. The main future work is to consider NoC architectures that provide QoS and to consider in a lightweight manner the costs of the mapper in terms of computation time and communication through the network to deliver the information required for the mapper. Another future

work is to allow the definition of deadlines for each message and keep control if the deadlines are missed during simulation. Further comparisons regarding the dynamic mapping algorithms will then be performed to extract a more solid evaluation.

REFERENCES

[1] E. Carvalho, N. Calazans, and F. Moraes, "Dynamic Task Mapping for MPSoCs," *IEEE Design & Test of Computers*, vol. 27(5), 2010.

[2] Singh, A. K., at al. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. Journal of Systems Architecture, 56(7), 2010.

[3] Faruque, M.A.; et al. ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication. In: DAC'08, 2008.

[4] Weichslgartner, A., et al. "Dynamic Decentralized Mapping of Tree-Structured Applications on NoC Architectures". In: NoCS'11, 2011.

[5] Hölzenspies, P.K.F.; et al. Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSoC). In: DATE'08, 2008.

[6] Wildermann, S.; et al. Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures. In: FPT'09, 2009.

[7] Molnos, A.; et al. Composable, energy-managed, real-time MPSOC platform. In: OPTIM'10, 2010.

[8] Smit, L.T.; et al. Run-time mapping of applications to a heterogeneous SoC. In: SoC'05, 2005.

[9] Chou, C-L. and Marculescu, R. "Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29(1), 2010.

[10] Ferrandi, F.; et al. "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29(6), 2010.

[11] Huang, L., et al. "Customer-Aware Task Allocation and Scheduling for Multi-Mode MPSoCs". In: Design Automation Conference, 2011.

[12] Määttä, S.; et al. "Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms". Int. J. of Embedded and Real-Time Communication Systems (IJERTCS), vol. 1(1).

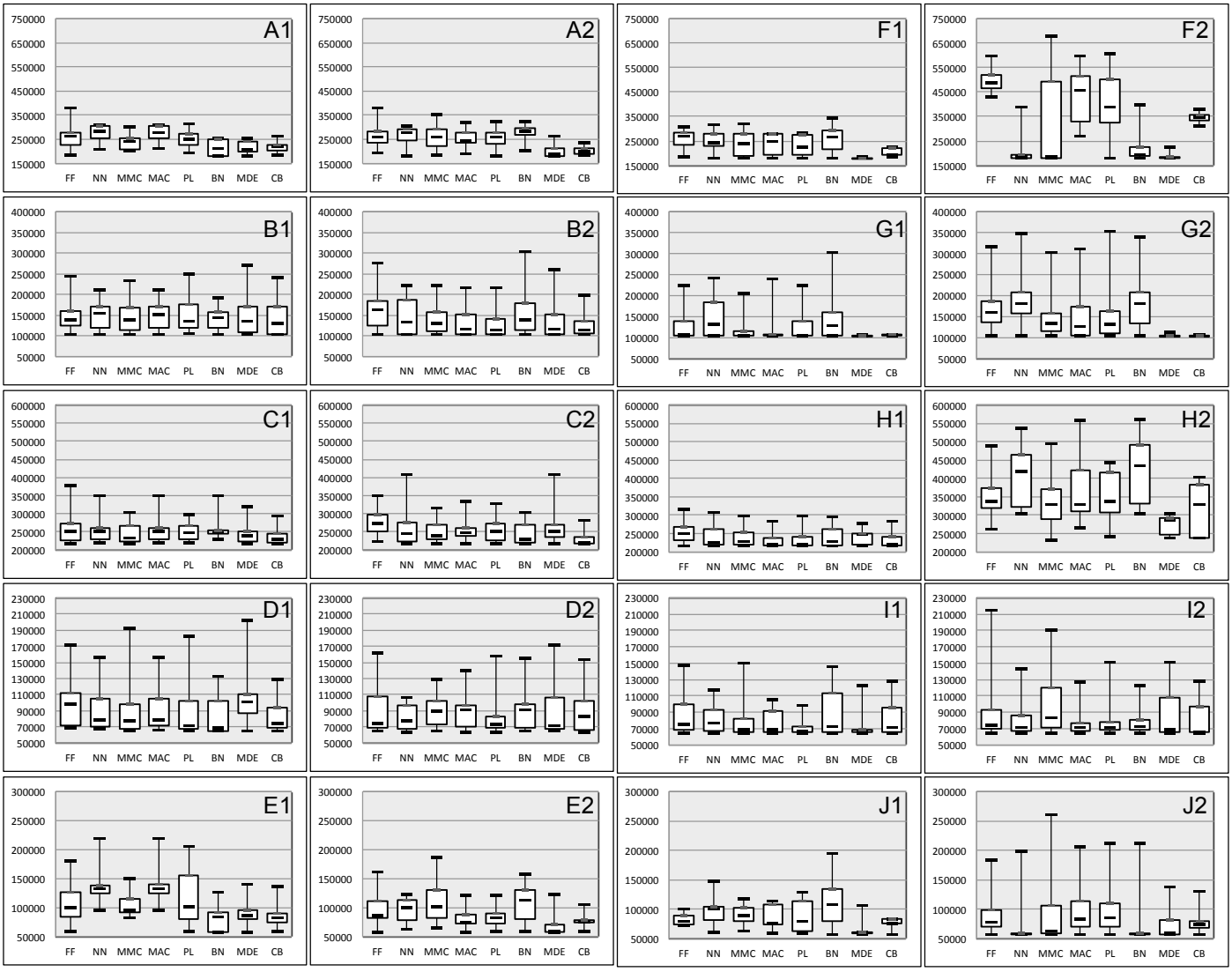

Figure 1. Response time of 5 application task graphs for the 4x4 platform (A-E) and the 5x5 platform (F-J). Graphs labeled with 1 use the CL initial mapper and graphs labeled with 2 use the FF initial mapper. The X-axis presents the dynamic mapping algorithm and the Y-axis presents the response times of all executed instances of each task graph in cycles (minimum, lower quartile, median, upper quartile and maximum response times).