

# Evaluation of Static and Dynamic Task Mapping Algorithms in NoC-Based MPSoCs

Ewerson Carvalho, César Marcon, Ney Calazans and Fernando Moraes  
Pontificia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS)  
Av. Ipiranga, 6681 – P32 – 90619-900 – Porto Alegre – RS – Brasil  
{ewerson.carvalho, cesar.marcon, ney.calazans, fernando.moraes}@puers.br

**Abstract** – Task mapping is an important issue in MPSoC design. Most recent mapping algorithms perform them at design time, an approach known as static mapping. Nonetheless, applications running in MPSoCs may execute a varying number of simultaneous tasks. In some cases, applications may be defined only after system design, enforcing a scenario that requires the use of dynamic task mapping. Static mappings have as main advantage the global view of the system, while dynamic mappings normally provide a local view, which considers only the neighborhood of the mapping task. This work aims to evaluate the pros and cons of static and dynamic mapping solutions. Due to the global system view, it is expected that static mapping algorithms achieve superior performance (*w.r.t.* latency, congestion, energy consumption). As dynamic scenarios are a trend in present MPSoC designs, the cost of dynamic mapping algorithms must be known, and directions to improve the quality of such algorithms should be provided without increasing execution time. This quantitative comparison between static and dynamic mapping algorithms is the main contribution of this work.

## I. INTRODUCTION

Applications running in MPSoCs (*e.g.* multimedia and networking) may present dynamic task workload, implying a variable number of tasks running at any given moment, which may exceed the available resources. As a result, it is necessary to control task operation and system resources use, including the dynamic management of task load.

*Task mapping* consists in finding a placement for application tasks, to fulfill a set of requirements (*e.g.* energy consumption saving and congestion reduction). Mapping decisions may drastically influence system performance. Concerning the *moment* in which it is defined, task mapping can be classified as *static* or *dynamic*. In the first case, tasks placement is defined at design time. Several works propose static mapping techniques, including *e.g.* references [1] to [7]. These techniques are not appropriate for dynamic workload scenarios, due to their algorithmic complexity and consequent execution time. Dynamic task mapping, on the other hand, is capable to define each task placement at runtime. Related references are [8] to [13].

This work investigates the performance of static and dynamic task mapping algorithms in NoC-based MPSoCs. The main objective here is to compare both mapping alternatives and enumerate advantages and drawbacks of each one. The investigation employs two static mapping algorithms proposed in several approaches available in the literature, and two dynamic algorithms originally proposed by some of the Authors in [13].

The remaining of this paper is organized as follows. Section II presents the state of the art in static mapping algorithms. The state of the art and some techniques of dynamic task mapping algorithms are supplied in Section III. Section IV presents the experimental setup and results. Finally, Section V provides a set of conclusions.

## II. STATIC MAPPING TECHNIQUES

Static approaches are suitable only for specific platforms, not allowing the insertion of new applications into the system at run-time. As these approaches are performed during the design time, the algorithms may use a more thorough amount of information about the system to take decisions. This includes the topology of the application graph, the volume/rate of the communication between tasks, the total system use, etc. In this way, the quality of mappings with a global view may be superior *w.r.t.* approaches with only a local view, as the dynamic mappings presented later.

Lei and Kumar [1] present a two-step genetic mapping algorithm that aims to optimize application execution time. Graphs represent applications and the target architecture is a NoC. In [2], Wu et al. also investigate the use of a genetic mapping algorithm. Results show 51% less energy consumption by considering DVS techniques in combination with the mapping algorithm. Murali et al. [3] explore mappings for more than one application during the NoC design process. The Authors employ the Taboo Search algorithm to explore the large solution search space. Manolache et al. [4] investigate task mapping into NoCs, aiming at guaranteeing packet latency. For this purpose, both task mapping algorithm (*i.e.* Taboo Search) and packet routing are defined at design time. In [5], Hu and Marculescu

present a branch-and-bound algorithm to map a given set of IPs onto a NoC with bandwidth reservation. Experimental results show 51.7% communication energy savings compared to an *ad hoc* implementation. Orsila et al. [6] investigate the mapping of applications on MPSoCs. For automated mapping, these Authors propose a new parameter selection scheme for the Simulated Annealing algorithm. The Authors suggest that this may render mapping faster and less memory expensive.

The static mapping algorithms used as reference in this work are based on [7], where the Authors investigate how to map modules into a NoC targeting low energy consumption. They compare several algorithms using the communication-weighted model CWM, where applications are characterized by inter-task communication volume. The employed heuristics are **Simulated Annealing (SA)** and **Taboo Search (TS)**.

### III. DYNAMIC MAPPING TECHNIQUES

In opposition to static mapping, in dynamic scenarios the time taken to map each task is relevant, since it influences the overall application execution time. To reduce mapping overhead, greedy algorithms may be used, since these trade search space exploration quality by fast results.

Even if the mapping execution time is an important cost function, the literature in general does not consider the overhead to map tasks [9] [11]. These works assume that the system execution time gains with this new mapping compensate the time spent to map each task. Based on this assumption, many authors employ the same strategies used for static scenarios in dynamic scenarios. For example, SA algorithms are employed for static mapping in [3] and [6] and for dynamic mapping in [9].

Smit et al. [8] present an iterative hierarchical approach to map an application to a NoC-based SoC at runtime, where a set of communicating tasks models the application. Aiming at energy consumption savings, the mapping algorithm tries to place each task near to its communicating entities. Similarly, in [9], besides the SA algorithm, Ngouanga et al. use a Force Directed mapping algorithm, which aims to approximate communicating tasks. Chou and Marculescu [10] include user behavior information in the task mapping process. This behavior information is employed to define tasks periodicity and communication rates. Mehran et al. [11] present a mapping algorithm that searches a placement following a Spiral path, tending to place communication tasks near to each other, as in [8]. This method executes for a single application without cost function evaluation. Al Faruque et al. [12] suggest a distributed agent-based mapping approach, recommended for larger MPSoCs, as the one presented in their paper, a 32x64 system.

The dynamic mapping algorithms used in this paper were originally proposed in [13]. The main cost function of these algorithms is to reduce the NoC congestion, being thus called *congestion aware* algorithms. MPSoCs presented in [13] are heterogeneous, composed by distinct Processing

Elements (PEs) as embedded reconfigurable logic and processors. In the present work, MPSoCs are homogeneously modeled, to focus on the comparison between static and dynamic approaches.

Each application is modeled having an initial task. The mapping of initial application tasks employs a *Clustering* strategy. The MPSoC is partitioned in regions named *clusters*, where only one initial task can be mapped at some specific time instant. As each application has a unique initial task, the clustering strategy reduces the probability of tasks belonging to different applications to share the same NoC region. Consequently, the sharing of NoC channels by communications of distinct applications is also reduced, which is one of the applied heuristics to fulfill the congestion reduction requirement.

The employed heuristics are:

**Path Load (PL)** – It considers only the channels used by the mapping task (*communication path*). The PL algorithm computes the cost of each mapping according to the communication path. Then, it adds the new task rates to the current rates in each channel of the communication path. The cost of all feasible maps is then computed, and the selected mapping is the first one with minimum cost.

**Best Neighbor (BN)** – The BN algorithm reuses the PL cost computation scheme changing the search method. In BN, the search method follows a spiral path, testing the neighbors of the task requesting a new task from distance 1 to NoC limits. The best neighbor is selected according to the communication path occupation. This algorithm is faster than PL, since it is not necessary to verify all possible maps. However, the BN algorithm does not evaluate the number of hops, so it may find false small cost maps.

### IV. EXPERIMENTS AND RESULTS

The simulation environment employs the Hermes NoC [14], a 2D-mesh topology with 16-bit flit width, described in RTL VHDL. Other parameters include wormhole packet switching, input buffers, and deterministic XY routing algorithm. Processing Elements are modeled using two different RTL *SystemC-Cthreads*, one for the Manager Processor and the second one for the remaining PEs.

The set of graphs employed in the experiments is composed by 4 synthetic application graphs (obtained with the TGFF tool, containing from 7 to 9 tasks), and 4 real applications graphs: MPEG-4 decoder, with 13 tasks; Video Object Plane Decoder (VOPD), with 13 tasks; Romberg integration method (RBERG), with 10 tasks; and Multi-Window Display (MWD), with 12 tasks. The MPEG-4 graph contains, as relevant characteristic, one task connected to 8 other tasks. The VOPD application presents smaller inter-task dependency when compared to MPEG-4. The RBERG graph presents an important inter-task dependency, where most tasks communicate with 4 tasks.

Two simulation scenarios are evaluated:

- **Single Application Mapping** – This scenario employs a 5x4 homogeneous MPSoC. The goal here is to compare mapping algorithms according to different performance figures, for a single application. Thus, a static mapping can explore all MPSoC resources without the interference of other applications.
- **Multiple Application Mapping** – This scenario employs a 9x9 homogeneous MPSoC. This scenario aims at comparing mapping algorithms when a set of applications are simultaneously mapped.

### A. Single Application Mapping

Figure 1 presents the task mapping of the real applications, for each algorithm. The *number of hops* required to execute all bidirectional communication between all pair of connected tasks is also indicated, for all mappings.

The SA algorithm obtains the smaller number of hops for all applications, which denotes a smaller number of used channels to accomplish all communications. BN and PL, which have a local view of the application, achieve results near to SA and superior to TS. The exception is the MPEG-4 application, where task 1 has 8 neighbors. For BN and PL this task was mapped near task 0 (its *caller*), and the remaining tasks connected to this task were mapped far from it, due to the lack of available resources. Algorithms considering the whole application graph (SA e TS) are able to map MPEG-4 task 1 in a central position.

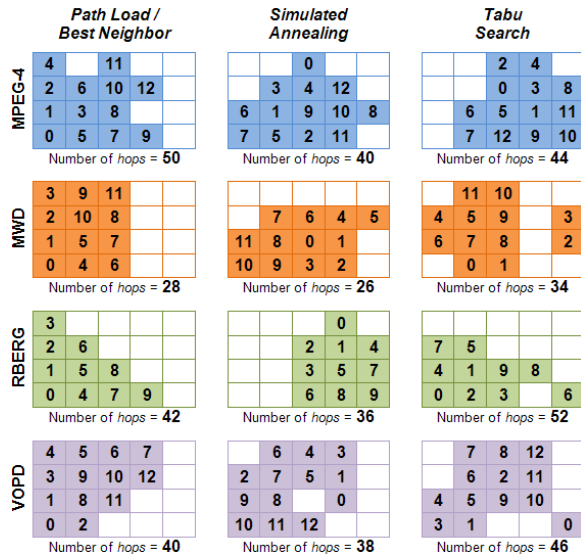


Figure 1. Mapping results. Each square represents one PE. Numbers inside squares identify the task allocated inside it.

Table I presents the results obtained in this first scenario, normalized to the SA algorithm. The first two parameters are related to the *channel occupation*, which represents NoC usage. Observe that the cost of the dynamic algorithms is low, approximately 6% for the average NoC occupation. The average overhead of *packet transmission latency* is even

lower, 1%. The average total *number of hops* increases 14% for the dynamic approaches.

TABLE I. SUMMARY OF RESULTS FOR THE MPSoC 5x4 SCENARIO. VALUES ARE NORMALIZED W.R.T. THE SA ALGORITHM RESULTS.

Performance figures	Dynamic		Static	
	PL	BN	SA	TS
Channel Occupation (average)	1.07	1.06	1.00	0.96
Channel Occupation (deviation)	0.97	0.92	1.00	0.94
Packet Latency (average)	1.01	1.01	1.00	1.01
Packet Latency (deviation)	0.99	0.99	1.00	1.00
Number of Hops	1.14	1.14	1.00	1.26
Total Execution Time	1.04	1.03	1.00	1.01
Communication Energy	1.38	1.38	1.00	1.11

Concerning the total execution time, Table I shows that the values obtained with BN/PL mapping algorithms are in average only 4% worse if compared to SA. It is important to point out that it is considered the time to map each task when dynamic algorithms are executed. Since static mapping algorithms are performed at design time, there is no mapping overhead during the application execution. In addition, it is important to note that this overhead is obtained for scenarios where applications execute, in average, only 310,909 clock cycles. Increasing the communication volume and/or task execution time, the task mapping overhead is minimized. For example, when the communication volume is increased by 10 times in [15], this overhead is canceled.

The last line of Table I presents the average energy consumption, estimated according to [7]. Dynamic mapping algorithms are penalized in this performance figure, due to the additional network hops involved in the communication. MPEG-4 is the application that most penalizes dynamic mapping algorithms. If this application is not considered in the average, the energy consumption penalty becomes smaller (17%). This comparison reveals that dynamic mapping has smaller costs compared to static mapping, when application graphs do not have strongly connected tasks. Situations like that in the MPEG-4 benchmark are not commonplace in real applications.

### B. Multiple Application Mapping

This experiment maps 8 concurrent applications into the MPSoC. The mapping quality may be estimated as a function of the area fragmentation. Figure 2 presents the application area distribution for the BN mapping. BN and PL generate applications that are more isolated. The smaller fragmentation observed in the dynamic mapping algorithm comes from the clustering strategy, which increases the size of contiguous blocks.

Table II presents the results for this second scenario, normalized as a function of the TS algorithm (for larger benchmarks TS outperforms SA). The overall overhead induced by dynamic mapping in such a complex scenario is small, which may be observed in columns PL and BN, respectively: 7% and 13% in average channel occupation,

7% and 10% in average packet latency, 4% and 3% in total execution time, and 18% and 13% in the communication energy consumption.

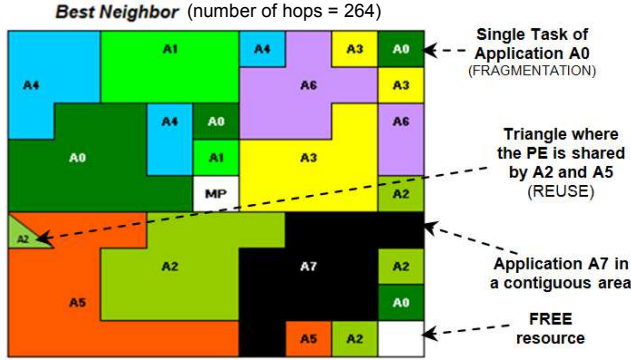


Figure 2. Application distribution with BN algorithm in a 9x9 MPSoC. Applications are represented by  $A_i$ . MP is the Manager.

TABLE II. SUMMARY OF RESULTS FOR THE MPSoC 9x9 SCENARIO. VALUES ARE NORMALIZED W.R.T THE TS ALGORITHM.

Performance figures	Dynamic		Static	
	PL	BN	SA	TS
Channel Occupation (average)	1.07	1.13	1.13	1.00
Channel Occupation (deviation)	1.18	1.27	1.36	1.00
Packet Latency (average)	1.07	1.10	1.16	1.00
Packet Latency (deviation)	1.03	1.11	1.05	1.00
Number of Hops	0.93	0.92	1.59	1.00
Total Execution Time	1.04	1.03	1.02	1.00
Energy Consumption	1.18	1.13	1.35	1.00

## V. CONCLUSION

This work addressed an important issue for the practical utilization of MPSoCs: task mapping. The evaluation of mapping algorithms is found in the literature, but a fair comparison between static and dynamic approaches was not yet available. This is the main contribution of this paper.

Experiments with a complex scenario (8 simultaneous applications) showed that the dynamic mapping overhead, when compared to static mapping was in average 10% in channel occupation, 8.5% in latency, 3.5% in total execution time and 15.5% in the communication energy consumption. This is an acceptable overhead, considering the advantages offered by dynamic mapping: (i) smaller systems may be used, since only tasks being executed are required to be mapped into the system; (ii) the number of tasks/applications may be superior to available system resources; (iii) the inclusion of new applications after system design extends the MPSoC lifetime.

As stated in the text, the main weakness of dynamic mapping is the partial view of the application graph, since the task being mapped considers only the communication

with its caller task. On the other hand, static algorithms consider all tasks and resources together, enabling to explore better mappings using complex algorithms. Some directions can be explored to improve the performance of dynamic mapping algorithms. Examples are: (i) when a given task starts its execution, a manager processor could start the mapping of all its slave tasks, or reserve resource for them; (ii) adoption of some task migration strategy when the communication cost becomes too high for a given task; (iii) to increase the view of the dynamic heuristic for 2 or more neighbor levels.

## VI. ACKNOWLEDGMENTS

This research was supported partially by CNPq (Brazilian Research Agency), projects 300774/2006-0, 471134/2007-4, 141225/2005-0, 308924/2008-8 and 309255/2008-2.

## REFERENCES

- [1] Lei, T.; Kumar, S. *Algorithms and Tools for Networks on Chip based System Design*. In: SBCCI, 2003. pp. 163-168.
- [2] Wu, D., Bashir, M.; Petru, E. *Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems*. In: DATE, 2003. pp. 253-360.
- [3] Murali, S.; et al. *A methodology for mapping multiple use-cases onto networks on chips*. In: DATE, 2006. pp. 118-123.
- [4] Manolache, S.; et al. *Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC*. In: DAC, 2005. pp. 266-269.
- [5] Hu, J.; Marculescu, R. *Energy- and Performance-Aware Mapping for Regular NoC Architectures*. IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, v. 24. 2005. pp. 551-562.
- [6] Orsila, H.; et al. *Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips*. Journal of Systems Architecture, v. 53-11, 2007. pp. 795-815.
- [7] Marcon, C.; et al. *Comparison of NoC Mapping Algorithms Targeting Low Energy Consumption*. IET Computers & Digital Techniques, v. 2-6. Nov. 2008. pp. 471-482.
- [8] Smit, L.T.; et al. *Runtime mapping of applications to a heterogeneous SoC*. In: International Symposium on SoC, 2005. pp.78-81.
- [9] Ngouanga, A.; et al. *A contextual resources use: a proof of concept through the APACHES platform*. In: DDECS, 2006. pp.42-47.
- [10] Chou, C-L.; Marculescu, R. *User-Aware Dynamic Task Allocation in Networks-on-Chip*. In: DATE, 2008. pp.1232-1237.
- [11] Mehran, A.; et al. *DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip*. IEICE Electronics Express, v. 5-13, p. 464-471, 2008. pp. 464-471.
- [12] Al Faruque, M. A.; et al. *ADAM: Runtime Agent-based Distributed Application Mapping for on-chip Communication*. In: DAC, 2008. pp. 760-765.
- [13] Carvalho, E.; Moraes, F. *Congestion-aware Task Mapping in Heterogeneous MPSoCs*. In: SoC, 2008. pp. 1-4.
- [14] Moraes, F. et al. *Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*. Integration, the VLSI Journal, Vol. 38-1, 2004.
- [15] Carvalho, E.; Calazans, N.; Moraes, F. *Investigating Runtime Task Mapping for NoC-based Multiprocessor*. In: VLSI-SoC, 2009.