Congestion-aware Task Mapping in Heterogeneous MPSoCs

Ewerson Carvalho and Fernando Moraes
Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS)
Av. Ipiranga, 6681 – P32 – 90619-900 – Porto Alegre – RS – Brasil
{ecarvalho, moraes}@inf.pucrs.br

***Abstract*—Multiprocessors Systems-on-Chip (MPSoCs) are a trend in VLSI design, since they minimize the design crisis represented by the gap between the silicon technology and the actual SoC design capacity. MPSoCs may employ NoCs to integrate several programmable processors, specialized memories, and other specific IPs in a scalable way. Besides communication infrastructure, another important issue in MPSoCs is task mapping. Dynamic task mapping is needed, since the number of tasks running in the MPSoC may exceed the available resources. Most works in literature present static mapping solutions, not appropriate for this scenario. This paper investigates the performance of mapping algorithms in NoC-based heterogeneous MPSoCs, targeting NoC congestion minimization. The use of the proposed congestion-aware heuristics reduces the NoC channel load, congestion, and packet latency.**

I. INTRODUCTION

The evolution of deep-submicron technology allows increasing circuits density significantly, giving support to the development of Multiprocessors System-on-Chip. MPSoC merges System-on-Chip (SoC) concept and the multiprocessing approach. It can integrate several programmable processors, specialized memories and IP-cores. Concerning its communication infrastructure, generally the use of NoC is mandatory in MPSoC design, since traditional communication infrastructures (e.g. busses) provide low scalability and small parallelism.

In general, industrial [1] [2] and academic [3] [4] devices employs homogeneous MPSoCs, composed by a set of identical processing elements. Homogeneous MPSoCs favors task migration and load balancing. Meanwhile, heterogeneous MPSoC, composed of distinct processor elements, tends to support a wide variety of applications. Tasks can be loaded on-demand at runtime and executed as software, at different processing elements, or as hardware at embedded reconfigurable logic.

Applications running in heterogeneous MPSoCs (e.g. multimedia, networking) may have their tasks dynamically loaded. Therefore, the number of tasks simultaneously running may exceed the available MPSoC resources. Thus, it is necessary to control task management and resource occupation. This management includes *task mapping*, a NP-hard problem that consists on finding a placement for a required task in the system, according to some specific criteria, as energy, channel load, and system fragmentation. Task mapping decisions reflect directly on the overall system performance. Most works in literature propose static mapping solutions [5] [6] [7], which are not appropriate for dynamic workloads scenarios, while few works focus on dynamic approaches [8] [9].

This work investigates the performance of a set of mapping algorithms for NoC-based MPSoCs, with dynamic workload. The main cost function is to optimize the NoC channel occupation. The mapping algorithms evaluation includes overall execution time, channel load, NoC congestion and packet latency.

The paper is organized as follows. Section II presents related works on task mapping. Section III discusses the target MPSoC architecture. Section IV introduces the task mapping algorithms. Section V discusses the experimental setup and results. Section VI presents conclusions and directions for future work.

II. RELATED WORKS

Academic works often propose homogeneous NoC-based MPSoCs [3] [4]. On the industrial side, IBM, Sony and Toshiba proposed a heterogeneous MPSoC composed of one manager processor and 8 floating-point units [10]. Intel [1] and Tilera [2] present homogeneous NoC-based MPSoCs, composed by 80 and 64 identical processors respectively.

Most works presenting static mapping algorithms employ NoC-based homogeneous MPSoCs [5] [6] [11]. Different algorithms are used: genetic approaches [7] [12], Tabu Search [11] [13], and Simulated Annealing [5] [6] [14]. Hu and Marculescu [5] and Marcon et al. [6] explore energy-aware mapping algorithms, with the purpose of reducing the overall power consumption by decreasing the consumed energy on communication. Results achieved in such works effectively improve the overall system performance, but this static mapping defines task placement at design time, for a system that will execute a fixed set of applications, with a well-known computation and communication behaviour. Therefore, static mapping approaches are not adequate for systems that support applications with dynamic workload.

Dynamic mapping approaches also target NoC-based homogeneous MPSoCs [8] [9]. Ngouanga et al. [8] present a mapping solution based on attraction forces between com-

municating tasks, which tend to place them near to each other. A drawback of this work is that the Authors do not consider the overhead of mapping on the execution time. Wronski et al. [9] evaluate heuristics for the bin-packing problem as Best Fit and Worst Fit, mapping all tasks belonging to a given application simultaneously. Differently from [9], the strategy proposed here maps only the initial task of each application. The advantage of dynamic mapping is to improve system use, since only resources executing tasks are effectively mapped. On the other hand, dynamic mapping can increase execution time, if a required resource is not available for mapping.

III. MPSoC Target Organization

A heterogeneous MPSoC is a set of different PEs interacting through a communication network [15]. PEs may support either hardware or software tasks execution. Software tasks execute in Instruction Set Processors (ISPs), while hardware tasks execute in reconfigurable logic (RL) or dedicated IPs. Figure 1 illustrates the proposed heterogeneous MPSoC organization. Among the available resources, one processor, named Manager Processor (MP), is responsible for resource control, task scheduling, task binding, task mapping, task migration, and configuration control. The MP starts the initial task of each application. New tasks are loaded into the MPSoC from the task memory when a communication to them is required, if they are not already mapped.

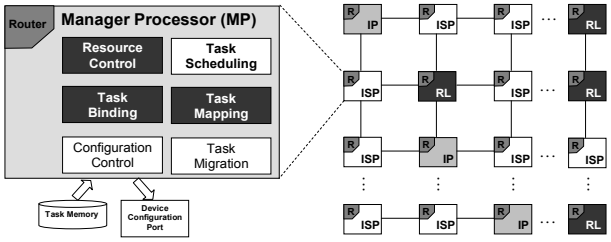

Figure 1. MPSoC target organization.

Actually, the present work emphasises resource control, task binding, and task mapping. Task scheduling is based on a queue strategy. There are three queues, one for each task type (i.e. hardware, software or initial). A task enters on a queue if there are no free resources, and it waits until this condition changes. The configuration control process is simulated based on configuration overhead results [16].

Resources are represented by R_{ij}, where i and j refer to the resource column and row respectively. The information stored into the resource matrix regards its type (hardware or software) and its status (free or used). Four matrices define the available channels: East channels (EC_{ij}); West channels (WC_{ij}); North channels (NC_{ij}); and South channels (SC_{ij}). Each element of the channel matrices contains the percentage of channel bandwidth use.

Inter-task communications use messages transmitted through the network. Four message types compose the adopted communication protocol: REQUEST, RELEASE, NOTIFY and GENERAL. A REQUEST message to the MP contains the identification of a new task to be inserted into the system and the respective communication volumes and rates. The RELEASE message notifies the MP that a processing PE has finished its current task, being possible to reuse the PE for a new task. The NOTIFY message is sent by the MP, after task mapping, to both tasks involved in the communication, containing task addresses to make possible a correct packet transmission. Inter-task communications employ the GENERAL messages

Usually, application models employ tasks graphs, which define tasks dependency [5] [6], volume [5] [6] of communication between tasks, and estimated computation time [17]. Here, an acyclic directed graph models an application, where vertices represent software or hardware tasks and edges define its dependencies. Volumes and rates from/to the node define communication values between tasks. Nodes connection defines a master-slave pair. Each graph edge has four parameters that represent the *volume* and *rate* of data sent from master to slave, and vice-versa. A source task of directed edges (master) needs to request the configuration of the target task (slave) before starting its communication.

IV. Dynamic Mapping Heuristics

The method used to define the mapping of each application initial task has a significant impact in the performance of the dynamic mapping. The method adopted in the present work divides the NoC into regions, named clusters. Each initial task is mapped to a unique cluster, reducing the probability of tasks belonging to different applications to share the same NoC region.

Two reference mapping methods are employed here: *First Free* (FF) and *Nearest Neighbor* (NN). FF is a method that starts at resource R_{00}, walking the network column by column, bottom to top. FF selects the first free resource according to binding definitions, without taking into account other metrics. FF may generate the worst results when compared to the other heuristics presented here. The NN method does not take into account other metrics except the proximity of an available resource able to execute the required task. NN starts searching for a free node able to execute the task near the requesting task. The search tests all n-hop neighbors, n varying between 1 and the NoC limits, stopping when a first resource able to execute the task is found.

A. Congestion-aware Mapping Heuristics

The *Minimum Maximum Channel Load* (MM) congestion-aware mapping heuristic evaluates all possible mappings for each new task inserted into the system. The goal of this heuristic is to globally minimize the channel occupancy peaks, reducing the occurrence of hotspots.

The *Minimum Average Channel Load* (MA) aims reducing the average occupancy of the NoC channels. This heuristic is similar to the MMC, replacing the *maximum* with the *average* function. While the MMC heuristic tries to minimize the channel occupancy peak, the MAC heuristic tries to homogenously distribute the communication load in the

NoC. The selected mapping is that resulting in the lower average channel occupancy.

The MMC and MAC heuristics consider all NoC channels while mapping a new task. Since this evaluation can take long, the *Path Load* algorithm are proposed to considers only the channels used by the task being mapped (its *communication path*). However, all possible mapping possibilities are still evaluated.

The *Best Neighbor* (BN) heuristic combines NN search strategy and the PL computation approach. The search space of BN is similar to NN, i. e., circular searches from the source node. This avoids computing all feasible mapping solutions, as in the PL heuristic, reducing the execution time for the heuristic. BN selects the best neighbor, according to PL equations, instead of the first free neighbor.

More details concerning such mapping algorithms are presented in [18]. In [18] TLM modeling was employed, while in the present work it is adopted a RTL level modeling, ensuring accurate results.

V. EXPERIMENTS AND RESULTS

This Section presents the experimental setup and results. All results are obtained from *ModelSim* co-simulation (RTL-VHDL for the NoC and *System-C* for the applications). The number of applications varies from 1 to 20, each one containing from 3 to 10 tasks. The processing time of each task is fixed to 25 microseconds. Rates are fixed from 5 to 30% of the available channels bandwidth, using a Pareto On-Off distribution. Each task transmits 200 to 500 packets, with size varying from 100 to 450 16-bit flits.

An 8x8 2D-mesh topology, described in VHDL [15], is responsible to transfer data between tasks. The maximum communication rate between routers is 50% of the available bandwidth, since the handshake control flow is used (two clock cycles to transmit one flit). The 64 processing elements (PEs) are distributed as follows: one node is used for the MP, 16 nodes are hardware resources and 47 nodes are software resources.

Three scenarios are evaluated: (*i*) 20 identical pipeline-like applications; (*ii*) 20 identical tree-like applications; and (*iii*) 20 different generic applications generated using TGFF [19]. Table I summarizes the results, comparing the relative gain of each mapping algorithms to the reference FF method.

The **average channel load** (2nd line of Table I) represents the NoC usage, primary cost function of MM, MA, PL and BN heuristics. As expected, all algorithms reduce the *average channel load* when compared to FF. Two congestion aware mapping algorithms, MM and MA, reduces 20% the channel load in average, result inferior to obtained with NN heuristic, where no traffic status is considered during the mapping. The BN heuristic have similar gains to NN. The average channel load is reduced to 38.04% when adopting PL, demonstrating its efficiency.

The **channel load standard deviation** (3th line of Table I) measures the traffic distribution inside the network. Lower values correspond to homogeneous traffic distribution, while higher values suggest some channels with higher loads, and others not used at all. This performance figure follows the average channel load. The PL and BN heuristics reduces the channel load standard deviation, with values similar to the NN heuristic.

TABLE I. ALGORITHM PERCENT GAIN IN RELATION TO FF METHOD.

Metrics	Mapping Algorithms (% *Gain w. r. t. FF*)				
	NN	MM	MA	PL	BN
Avg. Channel Load	34.49%	21.59%	21.56%	**38.04%**	35.81%
Channel Load s.d.	24.67%	12.71%	12.70%	**25.63%**	25.44%
Avg. Packet Latency	**13.90%**	5.38%	6.85%	13.79%	13.79%
Packet Latency s.d.	43.84%	5.87%	27.23%	46.42%	**48.42%**
Congestion n.c.	56.81%	32.85%	25.88%	**63.11%**	61.51%
Congestion w.t.	77.35%	41.07%	44.72%	**78.34%**	75.46%
Execution Time	**1.69%**	-15.63%	-8.47%	-4.06%	0.51%

The **average packet latency** (4th line of Table I) is proportional to the distance between source and target PEs, as well the congestion in the communication path. All heuristics leads to positive gains, with similar results between the simple NN and the congestion-aware algorithms PL and BN.

The **latency standard deviation** (5th line of Table I) is an important metric for systems with QoS constraints. Packets being transmitted from a given source must keep a regular interval between then to respect the injection rate. Variation in the latency incurs in jitter, which may lead to packet loss for applications with strict temporal deadline (e.g. real-time). The PL and BN congestion aware heuristics reduces the latency standard deviation to 46.42% and 48.42% respectively. The gain obtained with NN is 43.84%, smaller when compared to the other congestion aware heuristics.

These results may suggest smaller gains with the congestion aware heuristics when compared to NN algorithm. Note that the NoC is not heavily used. Only 9 simultaneous applications may run simultaneously, and each application may have up to 10 tasks, Therefore, the maximum number of simultaneous tasks (90) is very close the number of PEs (64), favoring the locality explored by the NN mapping heuristic. Even in such situation, the congestion aware heuristics surpass the NN heuristic.

The evaluation of **congestion** uses two metrics (6th and 7th lines of Table I). The first one, *n.c.* – number of saturated channels, counts the number of channels with an occupancy equal to the maximum allowed (50%). The second one, *w.t.* – wasted time, measures the number of clock cycles packets stalled in routers buffers. Reductions are observed with PL and BN heurists when compared to NN. PL is the most effective heuristic to reduce congestion, in both metrics.

Finally, the **execution time overhead** must be considered (8th line of Table I). The cost to perform the congestion aware heuristics, using FF as reference, is 4% for PL and 0.5% to the BN. This is an acceptable cost, since the reduction on channel load and congestion implies in energy saving. In addition, longer execution times applications may compensate this mapping cost

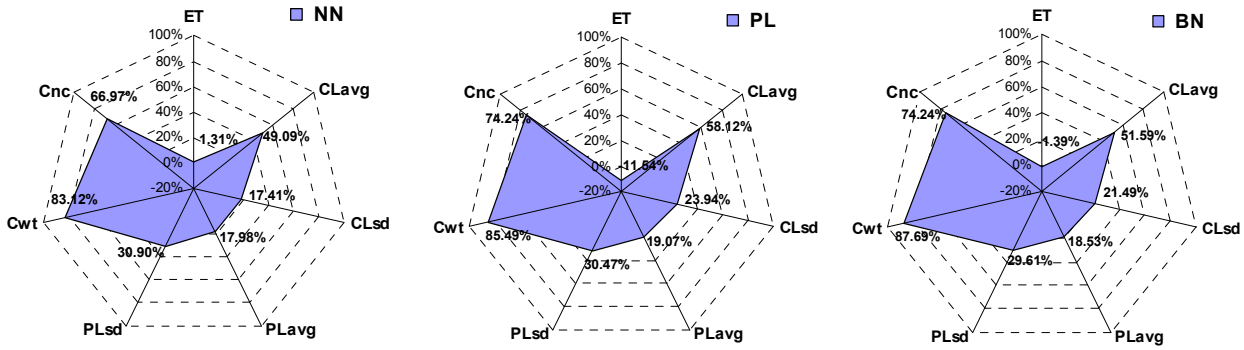

Figure 2. Results for NN, PL and BN mapping algorithms in comparison to FF, concening: the average channel load (*CLavg*) and standard deviation (*CLsd*); the total system execution time (*ET*); average packet latency average (*PLavg*) and deviation (*PLsd*); the accumulated number of NoC congested channels (*Cnc*) and respective wasted time (*Cwt*). This results are for the third scenario (i.e. 20 different generic applications generated using TGFF).

Figure 2 summarizes the improvements of **NN** (not congestion aware), **PL** (congestion aware) and **BN** (congestion aware) algorithms when compared to FF. It takes into account the third scenario (TGFF), where a more realistic scenario is modeled, since different application graphs co-exist simultaneously. The gains obtained for each performance figure can be inferred based on figures surfaces. From one side, PL and BN improve channel load, latency and congestion when compared to FF and NN. By the other hand, such congestion aware heuristics increase the total execution time.

VI. CONCLUSION

Some commercial MPSoCs were recently presented, exposing the necessity for research in MPSoC area, including mechanisms for the dynamic control of the system functionalities. Communication infrastructure and task mapping are two important research topics. Concerning communication schemas, NoC seems to be a trend. Several works on task mapping were found in literature, most of them targeting static mapping, not suitable for systems with dynamic task load.

Congestion-aware dynamic mapping heuristics are proposed and evaluated here. According to obtained results, PL heuristic is the best solution when compared to the other algorithms. MA and MC are not effective since these algorithms take local decisions: when a new mapping does not reduce the average link load (or maximum load), the algorithm selects the first mapping option available. NN and BN have similar improvements, however with gains inferior to PL. The heuristics execution time is an important parameter, which increases the total execution time. While NN and FF execute from 15 to 20 clock cycles, MMC and MAC present the worse performances (1000 and 1600 clock cycles). PL and BN execute in approximately 500 and 100 clock cycles, respectively. As mentioned before, in application with larger execution time, the mapping execution time overhead will be negligible.

Currently, two related topics are being investigated by the Authors: benchmarks evaluation with higher NoC loads and energy consumption measurements. Other works deemed as essential are the evaluation of heuristics complexity and the modeling of the use of multitasking processors.

ACKNOWLEDGMENTS

This research was supported partially by CNPq (Brazilian research agency), projects 141225/2005-0 and 300774/2006-0.

REFERENCES

[1] Vangal, S.; et al. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In: ISSCC.2007.
[2] Tilera Corporation. TILE64™ Processor. Product Brief. 2007.
[3] Lin, L.; et al. Communication-driven task binding for multiprocessor with latency insensitive network-on-chip. In: ASPDAC, 2005.
[4] Saint-Jean, N.; et al. HS-Scale: a Hardware-Software Scalable MPSOC Architecture for embedded Systems. In: ISVLSI. 2007.
[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, Vol 24-4. 2005.
[6] Marcon, C.; et al. Evaluation of Algorithms for Low Energy Mapping onto NoCs. In: ISCAS. 2007.
[7] Lei, T.; and Kumar, S. Algorithms and Tools for Networks on Chip based System Design. In: SBCCI. 2003.
[8] Ngouanga, A.; et al. A contextual resources use: a proof of concept through the APACHES platform. In: DDECS. 2006.
[9] Wronski, F.; et al. Evaluating Energy-aware Task Allocation Strategies for MPSoCS. In: DIPES. 2006.
[10] Kistler, M.; et al. Cell Multiprocessor Communication Network: Built for Speed. IEEE Micro, Vol 26-3. 2006.
[11] Murali, S.; et al. A methodology for mapping multiple use-cases onto networks on chips. In: DATE. 2006.
[12] Wu, D.; et al. Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems. In: DATE. 2003.
[13] Manolache, S.; et al. Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC. In: DAC. 2005.
[14] Orsila, H.; et al. Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips. Journal of Systems Architecture, Vol 53-11. 2007.
[15] Moraes, F. et al. Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. Integration, the VLSI Journal, Vol 38-1. 2004.
[16] Möller, L.; et al. A NoC-based Infrastructure to Enable Dynamic Self Reconfigurable Systems. In: ReCoSoC. 2007.
[17] Marcon, C.; et al. Exploring NoC mapping strategies: an energy and timing aware technique. In: DATE. 2005.
[18] Carvalho, E.; et al. Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs. In: RSP. 2007.
[19] Dick, R.P.; et al. TGFF: task graphs for free. In: CODES/CASHE, 1998.