Investigating Runtime Task Mapping for NoC-based Multiprocessor SoCs

Ewerson Carvalho, Ney Calazans, Fernando Moraes
Faculdade de Informática - Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS - Porto Alegre, Brazil
{ewerson.carvalho, ney.calazans, fernando.moraes}@pucrs.br

Abstract—**Multiprocessor Systems on Chip (MPSoCs) are a trend in VLSI design, since they minimize the design crisis configured by the gap between the silicon technology and the actual SoC design capacity. An important issue in MPSoCs is task mapping. Applications running in MPSoCs execute a varying number of tasks simultaneously, where each task may be started at some distinct moment, according to applications requests. Thus, task mapping should be executed at runtime. This work investigates the performance of dynamic task mapping heuristics in NoC-based MPSoCs, targeting NoC congestion minimization. Tasks are mapped on demand, according to the NoC channels load. Results using congestion-aware mapping heuristics compared to a straightforwardly defined heuristic achieve better results. In average, it is possible to reach up to 31% smaller channel load, up to 22% smaller packet latency, and up to 88% less. (Abstract)**

Keywords-task mapping; NoC; MPSoC (key words)

I. Introduction

The evolution of deep-submicron technology dramatically increases the density of ICs, enabling the development of SoCs. SoC design relies on the massive reuse of IP cores, reducing design effort and time-to-market. Multiprocessors Systems-on-Chip or simply MPSoCs are SoCs with multiple processing elements (PEs). A communication infrastructure interconnects PEs. As traditional communication schemes (such as buses and point-to-point links) will not be able to scale in future architectures [1], the use of Networks-on-Chip (NoCs) seems mandatory.

A homogeneous MPSoC contains a set of identical PEs, typically programmable processors. While homogeneous MPSoCs simplify task migration, heterogeneous MPSoCs tend to support a wider variety of applications because they integrate distinct PEs. Recently, Sony, Toshiba and IBM proposed a heterogeneous MPSoC composed of 9 RISC processors [2]. Intel Research [3] and Tilera [4] developed homogeneous NoC-based MPSoCs, with 80 and 64 PEs respectively. These architectures reveal the growing complexity in MPSoC development trends.

Applications running in MPSoCs (e.g. multimedia and network), may present a dynamic task workload. This implies a varying number of tasks running at any given moment, with their number possibly exceeding the available resources. Thus, it is necessary to control task operation and system resources usage, including the dynamic management of task load. *Task mapping* is an important issue, which consists in finding a placement for a set of tasks, aiming to fulfill some specific requirement (*e.g.* energy consumption saving, congestion reduction). Mapping decisions may drastically influence on the overall system performance.

The *objective* of this work is to investigate the performance of six different mapping algorithms for NoC-based MPSoCs. The main objective is to minimize congestion inside the network (*i.e. congestion-aware*) through the optimization of its channels usage. The evaluated performance figures include total execution time, NoC channels load, NoC congestion level, and packet latency.

The paper is organized as follow. Section II discuses related works in task mapping. Section III presents the target architecture, and the mapping problem formulation. Section IV explains the proposed algorithms for dynamic task mapping. Section V explores the experimental setup and presents the obtained results. Finally, Section VI displays some conclusions.

II. Related Works

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. This approach is not appropriate for dynamic workloads scenarios, due to their complexity and execution time. Work in [5] presents a two-step genetic mapping algorithm that aims to optimize application execution time. In [6], Wu et al. also investigate the use of a genetic mapping algorithm. Results show 51% less energy consumption using DVS techniques in combination with the mapping. Murali et al. [7] explore mappings for more than one application during the NoC design process. Taboo Search algorithm is employed to explore the large search space. Work in [8] investigates task mapping in NoCs, aiming to guarantee packet latency. For this purpose, both task mapping and packet routing are defined at design time. Work in [9] presents a branch-and-bound algorithm to map a given set of IPs into a NoC with bandwidth reservation. Work in [10] maps tasks using simulated annealing and taboo search to reduce the total energy consumption. Work in [11] investigates mapping of applications on MPSoCs, and propose a parameter selection scheme for the Simulated Annealing algorithm.

Dynamic mapping, on the other hand defines each task place at runtime. Smit et al. [12] present an iterative hierarchical approach to map an application to a NoC-based SoC at runtime. The application is modeled as a set of communicating tasks. The optimization targets energy consumption saving. For this purpose, the mapping algorithm tries to place each task near to its communicating entities. In [13], Ngouanga et al. propose the use of a Force Directed mapping algorithm, which aims at approximating communicating tasks location. Mehran et al. [14] present a mapping algorithm that searches a placement following a Spiral path, tending to place communication tasks near to each other, as in [12]. This method is executed for a single application without cost function evaluation. Al Faruque et al. [15] suggest a distributed agent-based mapping approach, which is recommended for larger MPSoCs, as 32x64 systems.

III. Problem Formulation

In the proposed heterogeneous NoC-based MPSoC model, each PE supports one task executing at a time. Tasks may be executed either in hardware or in software. Software tasks execute in *Instruction Set Processors*, while hardware tasks execute in *embedded reconfigurable logic*. MPSoCs that employ reconfigurable logic may load hardware tasks on the system at runtime. This technology allows flexibility to hardware in a level similar to the software programmability. Among the available MPSoC resources, one processor, named *Manager Processor* or simply MP, is responsible for controlling task operation and system resources usage, including dynamic management of task loads. One important assumption of the model is that task mapping is fired by an attempt to communicate with a not yet mapped task.

Definition 1 – A task is a triple T = (Tid, Tex, Tty), where Tid $\in \mathbb{N}$ is the task identifier; Tex $\in \mathbb{N}$ is its execution time, and Tty $\in$ {software, hardware, initial} is the task type.

Definition 2 – A communication Cms between tasks m and s is a 4-tuple Cms=(Vms, Rms, Vsm, Rsm), where Vms $\in \mathbb{N}$ is the data volume sent from m to s, according to the injection rate Rms. Vsm and Rsm have equivalent meaning for data sent from s to m. V values express transmitted flits, while Rs are percentage usage of available bandwidth.

Definition 3 – An application graph is a directed graph AP=<ST, SC>, where ST is the vertex set, representing the set of application tasks, and SC is the edge set, representing tasks communications.

Definition 4 – A processing element is a 5-tuple PE=(PEid, PEad, PEti, PEuse, ST), where PEid $\in \mathbb{N}$ is the PE identifier and PEad $\in \mathbb{N}$ is the PE address, used to receive packets. PEti $\in$ {software, hardware, initial} is the PE type and PEuse $\in$ {occupied, free} represents its usage status. Finally, ST is the set of PE mapped tasks.

Definition 5 – A communication channel is a pair C = (Cw, Cuse), where Cw $\in \mathbb{N}$ represents the channel width in bits, including both data and control wires, and Cuse is the data transmission available bandwidth usage, expressed as a percentage of available bandwidth.

Definition 6 – An MPSoC graph is a graph GMPSoC = <PE, L>, where the vertex set PE is the set of processing elements of the MPSoC, and L is the edge set, that represents on chip MPSoC communication channels.

Definition 7 (**Mapping)** – Let T = {t_1, t_2, ..., t_n} be a set of n tasks, and PE = {pe_1, pe_2, ..., pe_m} be the set of m processing elements of a GMPSoC = <PE, L>. A mapping is an injective function f: T$\rightarrow$PE, which associates tasks to processing elements.

The task mapping problem is very similar to the Quadratic Assignment, a well-known NP-hard problem. The use of heuristics to solve the mapping problem at runtime is mandatory for even moderately sized NoCs.

IV. Dynamic Task Mapping Algorithms

A *clustering* heuristic is used to define the mapping for each application initial task (see Definition 1). The NoC is divided into regions or *clusters*. Each cluster may support only one initial task at some specific moment. Thus, resource overlapping between different applications may happen, but its incidence is reduced. As clusters boundaries are virtual, an application may use parts of different clusters if necessary.

Instead of mapping all tasks of an application at once as proposed in [5] [9] [10] [15], the approach proposed here is based on greedy algorithms, which map only one task at a time. Mapping all tasks at once might provide better solutions for task mapping, since it considers all global information about resource utilization and the whole application graph [16]. However, in a dynamic workload scenario it is not possible to define when each task is needed. In addition, mapping the complete application at once may underutilize the system. Even if a set of resources is available, if their number and/or type do not match those required by the whole application, the latter cannot be mapped.

A. *Reference Mapping Methods – FF and NN*

Two reference mapping methods are employed here: *First Free* (FF) and *Nearest Neighbor* (NN). *First Free* is a method that starts at resource R_{00}, walking the network column by column, and 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. However, even worse solutions would be expected if random mapping were used as reference. Similar to FF, the *Nearest Neighbor* 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.

B. *Minimum Maximum Channel Load – MMCL*

The MMCL 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 usage peaks, reducing the occurrence of hotspots. For every mapping *k*, the communication rates *Rms* and *Rsm* are added to the channels employed for each master-slave communication. The selected mapping is that resulting in the smaller maximum channel load. In the present work, NoC links are modeled by four matrices: East channels (EC_{ij}); West channels (WC_{ij}); North channels (NC_{ij}); and South channels (SC_{ij}). Assuming an *x* by *y* NoC, for each channel matrix a maximum rate *MR* is obtained with Equations 1 to 4, where $R_{EC(i,j)}$, $R_{WC(i,j)}$, $R_{NC(i,j)}$ and $R_{SC(i,j)}$, denote the rate of each channel matrix element.

$$\mathrm{MR_{EC}(k)} = \max\left(\mathrm{R_{EC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x-1;\ 0 \le j < y \tag{1}$$

$$\mathrm{MR_{WC}(k)} = \max\left(\mathrm{R_{WC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x-1;\ 0 \le j < y \tag{2}$$

$$\mathrm{MR_{NC}(k)} = \max\left(\mathrm{R_{NC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x;\ 0 \le j < y-1 \tag{3}$$

$$\mathrm{MR_{SC}(k)} = \max\left(\mathrm{R_{SC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x;\ 0 \le j < y-1 \tag{4}$$

Equation 5 computes the maximum cost for each *k* of the *kP feasible mappings*, named $C_{MMCL}(k)$. Finally, the selected mapping is the one with minimum cost, as given by Equation 6.

$$\mathrm{C_{MMCL}(k)} = \max\left(\mathrm{MR_{EC}(k),\ MR_{WC}(k),\ MR_{NC}(k),\ MR_{SC}(k)}\right) \tag{5}$$

$$\mathrm{Selected_{MMCL}(k)} = \min\left(\mathrm{C_{MMCL}(k)}\right)\ \forall\ k,\ 0 \le k < kP \tag{6}$$

C. *Minimum Average Channel Load – MACL*

The MACL aims at reducing the average NoC channels usage. This heuristic is similar to the MMCL, replacing the *max* function with the *avg* (average) function. While the MMCL heuristic tries to minimize the channel peak usage, the MACL heuristic tries to homogenously distribute the communication load in the NoC. The selected mapping is the one resulting in the lower average channel occupancy. Equations 7 to 10 compute the average load of each NoC channel, for a given mapping *k*.

$$\mathrm{AR_{EC}(k)} = \mathrm{avg}\left(\mathrm{R_{EC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x-1;\ 0 \le j < y \tag{7}$$

$$\mathrm{AR_{WC}(k)} = \mathrm{avg}\left(\mathrm{R_{WC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x-1;\ 0 \le j < y \tag{8}$$

$$\mathrm{AR_{NC}(k)} = \mathrm{avg}\left(\mathrm{R_{NC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x;\ 0 \le j < y-1 \tag{9}$$

$$\mathrm{AR_{SC}(k)} = \mathrm{avg}\left(\mathrm{R_{SC(i,j)}(k)}\right)\ \forall\ (i,j), 0 \le i < x;\ 0 \le j < y-1 \tag{10}$$

Equation 11 computes the average cost for each mapping *k*, named $C_{MACL}(k)$. The selected mapping is the one with minimum average cost, as given by Equation 12.

$$\mathrm{C_{MACL}(k)} = \mathrm{avg}\left(\mathrm{AR_{EC}(k),\ AvgR_{WC}(k),\ AR_{NC}(k),\ AR_{SC}(k)}\right) \tag{11}$$

$$\mathrm{Selected_{MACL}(k)} = \min\left(\mathrm{C_{MACL}(k)}\right)\ \forall\ k, 0 \le k < kP \tag{12}$$

D. *Path Load – PL*

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

All inter-task communication occurs by means of a communication path, formed by path segments. A *path segment* groups a set of network channels, defined as a function of the task placement and of the routing algorithm. A path segment can be empty, if the master-slave pair is mapped to the same column or row of PEs. Given two tasks, *A* and *B*, *CP(A,B)* represents a *communication path* between these tasks. It is composed by four segments, where $PS_{EC}(A,B)$ is the *path segment* taken from East channels matrix, and $PS_{WC}(A,B)$, $PS_{SC}(A,B)$ and $PS_{NC}(A,B)$ has similar meaning.

The Path Load heuristic computes the cost of each mapping *k*, according to its respective path segments, using Equations 13 to 16. For this purpose, it adds the rates of the new tasks to the current rates in each channel in *CP*.

$$\mathrm{SR_{EC}(k)} = \sum\left(\mathrm{R_{EC(i,j)}(k)}\right)\forall(i,j) \in \mathrm{PS_{EC}}, 0 \le i < x-1; 0 \le j < y \tag{13}$$

$$\mathrm{SR_{WC}(k)} = \sum\left(\mathrm{R_{WC(i,j)}(k)}\right)\forall(i,j) \in \mathrm{PS_{WC}}, 0 \le i < x-1; 0 \le j < y \tag{14}$$

$$\mathrm{SR_{NC}(k)} = \sum\left(\mathrm{R_{NC(i,j)}(k)}\right)\forall(i,j) \in \mathrm{PS_{NC}}, 0 \le i < x; 0 \le j < y-1 \tag{15}$$

$$\mathrm{SR_{SC}(k)} = \sum\left(\mathrm{R_{SC(i,j)}(k)}\right)\forall(i,j) \in \mathrm{PS_{SC}}, 0 \le i < x;\ 0 \le j < y-1 \tag{16}$$

The cost C_{PL} is obtained from Equation 17, for all feasible maps. The selected mapping is the one with minimum C_{PL} cost (Equation 18).

$$\mathrm{C_{PL}(k)} = \mathrm{SR_{EC}(k) + SR_{WC}(k) + SR_{NC}(k) + SR_{SC}(k)} \tag{17}$$

$$\mathrm{Selected_{PL}(k)} = \min\left(\mathrm{C_{PL}(k)}\right)\ \forall\ k,\ 0 \le k < kP \tag{18}$$

E. *Best Neighbor – BN*

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

V. EXPERIMENTS AND RESULTS

The simulation environment employs the RTL VHDL Hermes NoC [17], a 2D-mesh topology with 16-bit flit width. Other parameters include: wormhole packet switching, input buffers, and deterministic XY routing algorithm.

PEs are modeled using *SystemC*, with two different RTL *SystemC-Cthreads,* one for the MP and the second one for the remaining PEs. The *MPthread* is responsible for the MPSoC management, task scheduling, and task mapping. In addition, this thread contains scheduling queues, channel occupation matrices and PE occupation matrix, to hold system usage status. This status is updated at runtime by monitors attached to all NoC ports, which measure the real channel occupation. MP takes mapping decisions according to matrices. The *TASKthread* implements the tasks behavior, which is customized by means of a configuration file that describes the communication rates and the execution time for each task.

Four simulation scenarios are evaluated:

a. *Pipe*: 20 identical *pipeline*-like applications (typical dataflow applications presenting this behavior), each one with 10 tasks, and injection rate varying from 5 to 30% of the available channel bandwidth;

b. *Tree*: 20 identical *tree*-like applications (typical parallel benchmarks have this profile), each one with 10 tasks, and injection rate varying from 5 to 20%;

c. *Generic*: 20 different *generic* applications generated using the TGFF, with 5 to 10 tasks, and injection rates randomly chosen from 5 to 30%.

d. *True Apps:* 8 different applications including: 4 applications where the graph and rates are based on real applications (MPEG-4, MWD, VOPD and Romberg integration), and 4 graphs generated using TGFF.

An 8x8 *heterogeneous* MPSoC is adopted for the experiments of scenarios **a**, **b** and **c**. One node (the router with its PE) is used for the MP, 16 nodes are hardware resources and 47 nodes are software resources. The PEs placement and the PEs reserved for initial tasks are defined according to each PE type, aiming to uniformly spread resources over the system area. The set of experiments arbitrarily assume a maximum of 15 applications simultaneously running.

Scenario **d** employs a 9x9 *homogeneous* MPSoC where all PEs are processors. Thus, applications are modeled as a set of software tasks only. This scenario aims to evaluate the performance of the mapping heuristics when graphs based on real applications are employed.

A. NoC Channels Usage

The *average channel load* represents the NoC use. All algorithms reduce the average channel load in comparison to the FF method. According to Figure 1, for the three first simulated scenarios (**a**, **b** and **c**), two congestion-aware mapping heuristics, MMCL and MACL, reduce in average 14% the channel load (compared to FF), a result inferior to the NN algorithm (29.95%), which does not consider the traffic during mapping, but explores the proximity of communicating tasks. The BN heuristic has gains similar to NN (29.74%), while PL shows the best gain (31.36%).

The *channel load standard deviation* measures the traffic distribution inside the network. Lower values correspond to a homogeneous traffic distribution, while higher values suggest some channels with higher loads, and others are not used at all. For the three first simulated scenarios, the BN heuristic reduces the channel load standard deviation (20% less), with values similar to the Nearest Neighbor algorithm. Again, PL presents the best results with a gain of 22% with regard to the FF reference mapping.

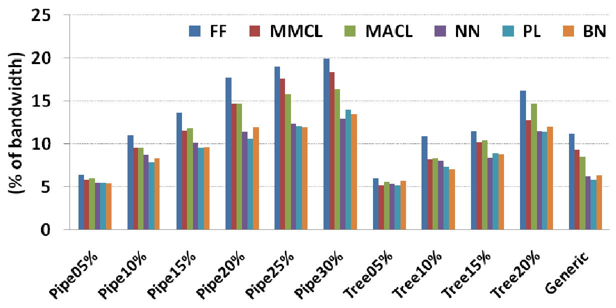

Figure 1. Average channel load (*in % of available bandwidth*) for the first three simulated scenarios.

B. Packet Latency

The *average packet latency* is a function of the distance between the source and the target PEs and the congestion in the communication path. Table I presents the average packet latency for the three first simulated scenarios. The last line of this Table shows for all algorithms its percentage gain with regard to the FF reference mapping. In most cases, PL heuristic obtains the best results. Additionally, PL, BN, and NN algorithms present quite close packet latencies values, resulting in similar gains (≈14%) compared to FF. However, Path Load again presents the best results (15%). The average gains for MACL and MMCL heuristics (respectively 8% and 6%) are less significant.

TABLE I. AVERAGE PACKET LATENCY (*IN CLOCK CYCLES*) FOR THE THREE FIRST SIMULATED SCENARIOS (A, B AND C).

Scenarios	Reference Mapping		Congestion-aware Heuristics			
	FF	**NN**	**MMCL**	**MACL**	**PL**	**BN**
Pipe 05%	164	**141**	152	156	142	147
Pipe 10%	269	239	255	258	239	**238**
Pipe 15%	371	334	350	356	**328**	330
Pipe 20%	527	431	479	478	**425**	450
Pipe 25%	656	542	645	579	**531**	**531**
Pipe 30%	815	665	781	711	673	**664**
Tree 05%	156	**143**	147	153	146	146
Tree 10%	271	247	255	246	**233**	242
Tree 15%	370	**327**	349	352	329	330
Tree 20%	514	433	447	493	**430**	434
Generic	343	286	319	313	**285**	291
% Gain w.r.t. FF	**-**	***14.98%***	***6.15%***	***8.07%***	***15.59%***	***14.66%***

The *packet latency standard deviation* is an important metric for systems with QoS constraints. Packets originated from a given source must keep a regular interval between them to respect the injection rate. Variation in the latency incurs in jitter, which can lead to packet loss in applications with strict temporal deadlines like real-time applications. In scenario **c**, composed by graphs generated using the TGFF, all congestion-aware heuristics improve the latency standard deviation. However, MMCL and MACL present 10% and 16% gains respectively, with regard to FF, while BN presents a 14% gain. NN and PL allow reducing in around 21% the standard deviation in packet latency.

C. *Network Congestion Level*

This paper employs two metrics for evaluating congestion. In The first metric, named *wasted time* (w.t.), monitors are employed to count the number of clock cycles a packet is stalled in router buffers. Figure 2 shows the total *wasted time* for the three first simulated scenarios. It is noticeable that the growth in injection rate causes an increase in the wasted time, due to increasing NoC congestion. For smaller injection rates, FF and MACL, are the algorithms presenting the worst results. As the injection rate increases, to *e.g.* 30%, the BN heuristic induces smaller NoC congestion, followed by PL and NN respectively. NN and BN algorithms reduce the time wasted in congested areas by approximately 83%. Algorithms MACL and MMCL achieve smaller but still significant improvements, respectively 52% and 47%. Again, PL presents the best results, with 87% of reduction in time wasted in congested areas.

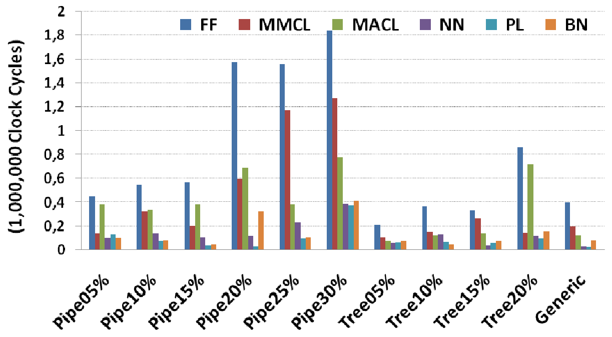

Figure 2. Total wasted time (in millions *of clock cycles*) for the three first simulated scenarios (a, b and c).

The second metric, named *saturated channels*, counts the number of channels that achieve their maximum allowed rate (Figure 3). NN, BN and PL algorithms reduce the number of saturated channels by 69%, 70%, and 76% respectively, when compared to the *First Free* mapping. MACL and MMCL obtain smaller improvements, 38% and 45%, respectively.

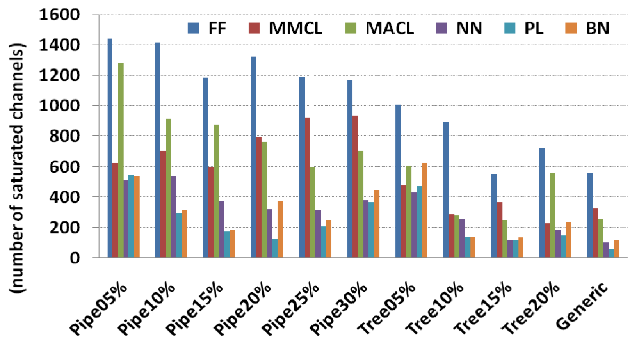

Figure 3. Total number of saturated channels for the three first simulated scenarios.

D. *Overall Execution Time*

The last evaluation concerns the *execution time overhead*. The goal of this parameter is to measure the impact on implementing the mapping heuristics on the overall execution time. The complexity of MMCL and MACL induces a large penalty for these heuristics (13% and 25% respectively). PL and BN heuristics present execution time overhead of 8.8% and 2.5%, respectively). This constitutes an acceptable cost, since these heuristics reduce channel load and congestion as well. A variation in parameters of the performed experiments shows that an increase in communication volume of 10 times can cancel this overhead. In this case, BN results in an execution time 0.8% less compared to FF, while the PL algorithm execution in 1.13% less time than FF. In addition, the evaluated scenarios are dominated by communication. Longer task execution times also tend to reduce the overhead.

Table II summarizes the results, normalized to the FF reference mapping. This Table considers the average results obtained from scenarios **a**, **b** and **c**. PL heuristic achieves the best results in most performance figures, as 31% smaller channel usage, 22% smaller packet latency, and up to 88% smaller NoC congestion. NN heuristic executes faster, due to its smaller algorithmic complexity. Note that with increased transmission volume, the mapping execution time does not affect the total execution time.

TABLE II. SUMMARY OF RESULTS FOR ALL MAPPING LGORITHMS NORMALIZED TO FIRST FREE RESULTS. IT CONSIDERS THE AVERAGE FOR THE THREE FIRST SIMULATED SCENARIOS (A, B AND C).

Metric	Reference Map.		Congestion-aware Heuristics			
	FF	**NN**	**MMCL**	**MACL**	**PL**	**BN**
Mapping complexity (x is the NoC width)	$O(x^2)$	$O(x^2)$	$O(x^4)$	$O(x^4)$	$O(x^3)$	$O(x^2)$
Channel Load (avg)	*1.00*	0.70	0.86	0.85	**0.69**	0.70
Channel Load (s.d.)	*1.00*	0.80	0.88	0.90	**0.78**	0.80
Packet Latency (avg)	*1.00*	0.85	0.94	0.92	**0.84**	0.85
Packet Latency (s.d.)	*1.00*	**0.66**	1.33	0.89	0.98	1.19
Congestion (s.c.)	*1.00*	0.31	0.55	0.62	**0.23**	0.29
Congestion (w.t.)	*1.00*	0.17	0.53	0.47	**0.12**	0.17
Exec. Time (Vol)	*1.00*	**1.00**	1.25	1.14	1.09	1.03
Exec. Time (10xVol)	*1.00*	**0.98**	1.00	1.00	0.99	0.99

E. *Scenario D Evaluation*

Scenario **d** is composed by 4 graphs obtained from real applications, and 4 synthetic graphs. The first application is the MPEG-4 decoder used for compression of audio and video digital data. It is composed by 13 tasks. The second application is the *Video Object Plane Decoder* or simply VOPD, also composed by 13 tasks. The VOPD application presents less inter-task dependency when compared to MPEG-4. The *Romberg's integration method* is the third application graph used in experiments. It contains 10 tasks. This graph presents a higher inter-task dependency, where most tasks communicate with 4 tasks. The fourth graph is based on *Multi-Window Display* (MWD) application, composed by 12 tasks. Synthetic graphs, generated by TGFF, contain 7 to 9 tasks.

Table III summarizes the obtained results. In this Table, PL and BN results are normalized to the FF reference mapping. As in previous experiments, the PL mapping displays the best results. It allows to improve the average channel load by almost 11% compared to FF. The average packet latency too, presents a considerable gain, 21% less compared to FF. NoC congestion still presents the most significant results since PL obtains 63% less saturated channels, and 67% less wasted time during NoC congestion periods.

TABLE III. SUMMARY OF RESULTS OF PL AND BN MAPPING ALGORITHMS NORMALIZED TO FIRST FREE RESULTS.

Metric	**FF**	**PL**	**BN**
Channel Load (avg)	*1.00*	**0.89**	0.92
Channel Load (s.d.)	*1.00*	**0.88**	0.98
Packet Latency (avg)	*1.00*	**0.79**	0.81
Packet Latency (s.d.)	***1.00***	1.80	1.93
Congestion (s.c.)	*1.00*	**0.37**	0.40
Congestion (w.t.)	*1.00*	**0.33**	0.43
Exec. Time	*1.00*	0.96	**0.95**

Compared to scenarios **a**, **b** and **c**, the reduction in channel load, packet latency and congestion level parameters observed in scenario **d** are smaller, but still significant. It is important to observe the total execution time, which is reduced using PL and BN heuristics. Such results, using complex applications and larger NoCs, highlights the advantages of using congestion-aware mapping heuristics.

VI. CONCLUSION

Some MPSoCs have recently been proposed in industry containing several tens of processors. To effectively use such complex MPSoCs it is necessary to include the management of the processing elements, in a distributed or centralized way. Such management includes, *e.g.* frequency and voltage control, processors and network load monitoring, task/application mapping. This research work contributes in last item. The dynamic task mapping heuristics proposed and evaluated enables to include new applications in the MPSoC, not only aggregating new functionalities to it after design, but also increasing its life time.

The present work investigated the performance of six different mapping algorithms for NoC-based MPSoCs. According to the obtained results, the PL heuristic is the best solution compared to all others. In summary, when compared to ad hoc mapping, the use of congestion-aware algorithms allows 31% smaller channel load, up to 22% smaller packet latency, and up to 88% reduction in congestion. Additionally, these heuristics presents an acceptable cost for the mapping overhead. MMCL and MACL are not effective, due to local decisions taken by the algorithms: when a new mapping does not reduce the average link load (or maximum load), the algorithm selects the first available mapping option. NN and BN display some improvements, but with gains inferior to PL.

Currently, three related research topics are under investigation: benchmarks evaluation with higher NoC loads, energy consumption measurements, and dynamic versus static mapping comparison [16].

ACKNOWLEDGMENTS

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

REFERENCES

[1] Henkel, J.; et al. On-chip networks: a scalable, communication-centric embedded system design paradigm. In: VLSID, 2004. pp. 845-851.

[2] Kistler, M.; et al. Cell Multiprocessor Communication Network: Built for Speed. IEEE Micro V26-3, 2006. pp. 10-26.

[3] Vangal, S.; et al. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In: ISSCC, 2007. pp. 5-7.

[4] Tilera Corp. TILE64™ Processor. Product Brief Description, 2007. pp. 1-2.

[5] Lei, T.; Kumar, S. Algorithms and Tools for Networks on Chip based System Design. In: SBCCI, 2003. pp. 163-168.

[6] Wu, D.; et al. Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems. In: DATE, 2003. pp. 253-360.

[7] Murali, S.; et al. A methodology for mapping multiple use-cases onto networks on chips. In: DATE, 2006. pp. 118-123.

[8] Manolache, S.; et al. Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC. In: DAC, 2005. pp. 266-269.

[9] 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. pp. 551-562.

[10] Marcon, C.; et al. Evaluation of Algorithms for Low Energy Mapping onto NoCs. In: ISCAS, 2007. pp. 389-392.

[11] Orsila, H.; et al. Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips. Journal of Systems Architecture V53-11, 2007. pp. 795-815.

[12] Smit, L.T.; et al. Runtime mapping of applications to a heterogeneous SoC. In: SoC, 2005. pp. 78-81.

[13] Ngouanga, A.; et al. A contextual resources use: a proof of concept through the APACHES platform. In: DDECS, 2006. pp. 42-47.

[14] Mehran, A.; et al. DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip. IEICE Electronics Express V5-13, 2008. pp. 464-471.

[15] Al Faruque, M.A.; et al. ADAM: Runtime Agent-based Distributed Application Mapping for on-chip Communication. In: DAC, 2008. pp. 760-765.

[16] Carvalho, E.; et al. Evaluation of Static and Dynamic Task Mapping Algorithms in NoC-Based MPSoCs. In: SoC, 2009. To appear.

[17] Moraes, F. et al. Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. Integration, the VLSI Journal, Vol 38-1, 2004.