Enhancing Performance of MPSoCs through Distributed Resource Management

Marcelo Mandelli, Guilherme M. Castilhos, Fernando G. Moraes
PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
{marcelo.mandelli, guilherme.castilhos}@acad.pucrs.br, fernando.moraes@pucrs.br

Abstract—**The constant growth in the number of cores in SoCs implies an important issue: scalability. NoC-based MPSoCs offer scalability at the hardware level. However, the management of the MPSoC resources requires also scalable methods, to effectively extract the computational power offered by dozens of processors. State-of-the-art proposals adopt different approaches to tackle such problem, using the MPSoC clustering as the most common approach. The present work proposes a distributed mapping approach, using a clustering method, having as main goal to evaluate its pros and cons. Evaluation is carried-out using cycle accurate simulation, in large MPSoCs (up to 144 processors). Results show an important reduction in the total execution time of the applications running in the MPSoC, even if some processors are reserved for resources management.**

I. Introduction and Related Work

MPSoC design is a clear trend in the semiconductor industry. As an example, Intel launched the Xeon Phi family [1], with up to 50 cores, with the goal to provide high performance for parallelizable applications. It is expected that the number of cores continue to increase, reaching a thousand of cores within a decade [2]. Thus, the constant growth in the number of cores implies an important issue: scalability. Despite the scalability offered by NoCs and distributed processing, the MPSoC resources must be managed to deliver the expected performance. Management tasks include access to input/output devices, task mapping, task migration, monitoring, DVFS. One processing element (PE) responsible for resource management may become a bottleneck, since this PE will serve all other PEs of the system, increasing its computation load and creating a communication hot-spot region. An alternative to ensure scalability is to decentralize or distribute the management functions of the system.

Al Faruque [3], Anagnostopoulos [4] and Cui [5] divide the system in regions, named *clusters*. In Al Faruque et al. [3] the clusters are controlled by an agent (manager) responsible for the task mapping within a cluster. The overall system is controlled by many synchronized global agents, responsible to store information related to all clusters, decide in which cluster a given application will be mapped and to re-organize the clusters (re-clustering) at runtime. Cui et al. [5] claim that the work proposed by Al Faruque leads to a high communication traffic to collect resource information to make decisions. To solve this issue, they propose a cluster based scheme for task mapping, aiming to reduce the communication traffic between a global agent and the cluster agents, removing some of local clusters information from the global agent and changing the cluster re-organization scheme.

Anagnostopoulos et al. [4] propose a divide and conquer method to perform distributed runtime mapping onto homogeneous and heterogeneous many-core platforms. When a new application is required to execute in the system, the proposed scheme divides the network in regions, using as criterion the application size. The region that best fit the application is chosen. A regional controller is configured to execute the mapping algorithm within the chosen region.

Kobe et al. [6] also propose task mapping using agents. However, agents do not have a predefined region to control as in the previous works. The agent selects PEs to map a given application. An agent is assigned at runtime to a random PE when a new application is required. Then, the agent searches for PEs to map the tasks, starting with the closest to farthest ones. The disadvantage of the presented method is the possibility of a communication bottleneck, since the agent may become distant from the application it controls.

Weichslgartner et al. [7] propose a decentralized mapping approach aiming to reduce the NoC congestion. The proposed mapping approach considers only a local view of adjacent nodes on the NoC to execute the mapping, where each task contains a list of its succeeding tasks to be mapped. The method is limited to applications with a tree topology. The root (initial task) is mapped first, and then each task executes the mapping heuristic to map its successor tasks.

Shabbir et al. [8] propose a decentralized management technique to meet performance requirements of applications. Through the evaluation, the Authors claim that the techniques provide scalability, but the proposed work is limited to experiments with a small number of processors (6 PEs).

Four approaches for distributed resource management were identified:

i. Clusters, each one controlled by a manager. The clusters may be responsible for more than one application and they may be re-organized, modifying its size at runtime through a *re-clustering* method [3][5].

ii. The system is divided in clusters, based on the applications' size. Instead of the previous approach, a cluster contains only one application [4].

iii. An application manager is used as in the second approach, but there are no predefined clusters, allowing spreading applications in non-contiguous PEs [6][8].

iv. A global function of the system is implemented in all nodes, such as the task mapping heuristic [7].

The Author Fernando G. Moraes acknowledge the support of CNPq, CAPES, FAPERGS, projects 301599/2009-2, 708/11 and 10/0814-9, respectively. The Authors also acknowledge the National Science and Technology Institute on Embedded Critical Systems (INCT- SEC) for the support to this research.

978-1-4673-1260-8/12/$31.00 ©2012 IEEE

The main limitation observed in the state-of-the-art is the lack of accurate results, due the abstract modeling used in these works. This paper has two main goals: (*i*) deploy distributed resource management architecture for NoC-based MPSoCs, using RTL cycle accurate modeling (SystemC); (*ii*) compare the performance of the centralized versus distributed approaches, using as cost function the execution time.

II. Distributed Management Architecture

The present work adopts the first distributed approach mentioned before. The distributed management architecture divides the MPSoC in *n* equally sized clusters, defined at design time. At execution time, if a given application does not fit in the cluster, the cluster may request additional resources to adjacent clusters, a process named *re-clustering*. Such approach is preferable to the other ones since:

i. The number of PEs dedicated to management functions is limited to number of clusters. An approach with one manager per application may imply large overhead, since the number of applications that will execute in the system is unknown at execution time.

ii. The clustering approach reduces the number of hops among tasks belonging to the same application, reducing the overall traffic in the NoC (if the application fits in the cluster).

iii. It is not necessary to create/destroy agents each time a new application enters/leaves the system, enhancing in this way the overall system performance.

Figure 1 shows a 6x6 MPSoC instance containing four 3x3 clusters. The MPSoC contains three types of PEs: the Global Master PE (GMP); Local Master PEs (LMP); Slave PEs (SP). The LMP is responsible to control the cluster, executing functions such as task mapping, task migration, monitoring, deadlines verification, and communication with other LMPs and GMP. The GMP contains all functions of the LMP, and functions related to the overall system management. Examples of such functions include choose in which cluster a given application will be mapped, control the available resources in each cluster, receive debugging and control messages from LMPs, access the application repository (external memory containing all tasks object codes), receive new applications requests from the external interface. SPs are responsible for task execution.

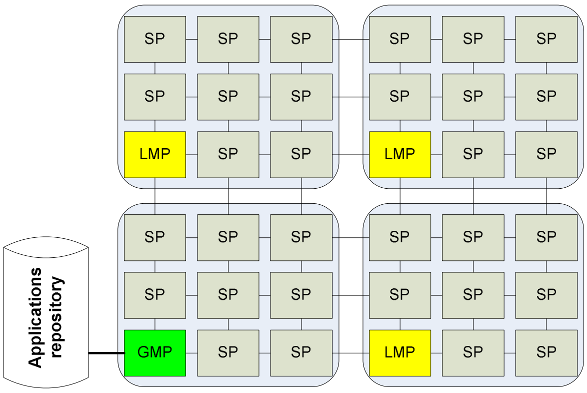

Figure 1 - Clustering approach for distributed resource management.

When the system starts, the GMP is also responsible for cluster initialization, informing LMPs which region they will manage. Thus, when a LMP knows the region it will control, it informs all SPs in this region that it will be their manager. This cluster and SPs initialization mechanism provides better system adaptability, allowing the modification of the cluster size at runtime.

The GMP is the only PE with access to the external devices (e.g the application repository). In Figure 1, four PEs are reserved for management functions, representing 11.1% of PEs not executing user applications. Using a 4x4 cluster in a 16x16 MPSoC, this overhead becomes 6.25%, which is an acceptable cost, considering the obtained benefits, as demonstrated in the Results Section.

Because the application mapping is the first action executed by the MPSoC manager when a given application enters into the system, the next section details the distributed mapping approach.

III. Distributed Mapping

This Section presents the proposed distributed mapping process. Section A details the protocol related to cluster selection when a new application enters into the system, detailing the interaction GMP→LMP. Section B details the intra-clusters mapping process, detailing the interaction LMP→SP. Section C compares qualitatively the distributed versus centralized mapping process.

A. Insertion of New Applications

Applications are modeled as task graphs. It is supposed that at least one task does not have dependences on other tasks, being the *initial(s) task(s)*. Applications are stored in the *application repository*. According to user requests, a new application may be requested to execute in the system. In Figure 2 this action corresponds to the arrow “new application”, entering in the GMP.

The GMP executes the “cluster selection” heuristic to choose the cluster that will receive the new application. The heuristic chooses the cluster that best fit the application in terms of available resources. The number of resources of a given cluster is defined according to Equation 1.

$$cluster\ resources = (PE\text{-}1) * pages \qquad (1)$$

where: *PE* is the number of processing elements of the cluster (one PE is dedicated to the LMP), and *pages* is the number of tasks a given PE can execute simultaneously.

Once a given cluster is selected, the GMP reads from the application repository the *application description*, transmitting it to the selected LMP. The application description contains:

- application identifier, $\mathrm{ID}_{\mathrm{App}}$, along with the task list;
- identification of each application task (2).

$$task_i = \{initial_i,\ ID_i,\ ADD_i,\ SIZE_i,\ \{(t_1,c_{t1});(t_2,c_{t2});(t_n,c_{tn})\}\} \qquad (2)$$

where: $initial_i$, defines if the application is initial or not; ID_i, task identifier; ADD_i, address of the task object code in the application repository; $SIZE_i$, size of the task object code $\{(t_1,c_{t1});(t_2,c_{t2});(t_n,c_{tn})\}$, list of tasks that communicate with $task_i$ and its respective communication volume.

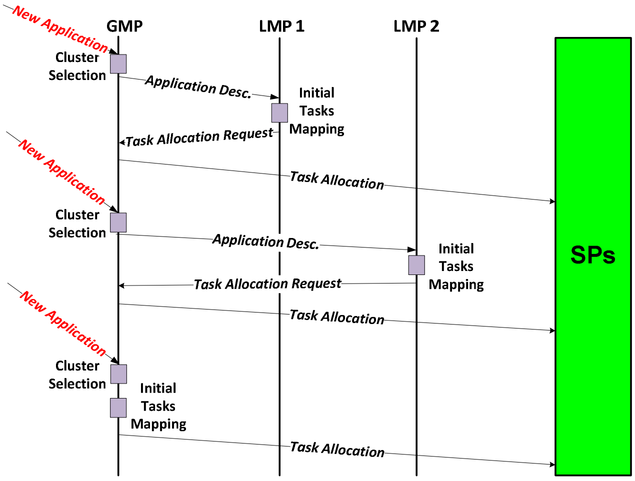

Figure 2 - Protocol to insert new applications into the system.

The selected LMP stores the *application description*, starting the task mapping process. Two mapping situations may arrive: mapping of the initial tasks and mapping of the remaining tasks. The initial task mapping searches the SP with the highest number of available resources around it, increasing the probability to map the remaining application tasks close to each other. This action reduces the distance among tasks, and therefore the communication energy. The mapping of the remaining tasks is explained next.

After selecting the SP that will receive the initial task, the LMP sends a packet to the GMP with the service *task allocation request* containing $\{PE_{position}, ID_i, ADD_i, SIZE_i,\}$. The GMP programs its DMA module to read $SIZE_i$ words of the application repository from ADD_i, transmitting these words to $PE_{position}$. The SP will schedule the new task at the end of the *task allocation* packet reception. In addition, the LMP keeps a data structure, named *task table*, with the address of all mapped tasks.

Consider in Figure 2 the third application insertion. This situation illustrates a scenario where the selected cluster is the one with the GMP. In this case, the GMP also executes the task mapping procedure.

B. Task Mapping Process

An application starts after the initial task mapping. When an initial task *task_1* starts, it executes some functions, and then it communicates with a given task (*send* in Figure 3). The operating system of the SP hosting *task_1* verifies if the target task (*task_2*) is present in its local task table. If it is present, the message is transmitted to the target task. If it is not, a packet with a *task request* service is transmitted to the LMP responsible for *task_1* ('1' in Figure 4).

```
Message msg1;
int main(){
   msg1.length = 128;
   ...
   send(&msg1,task_2);  //communicate with task_2
   ...
   exit();
}
```

Figure 3 – Example of an initial task description, with a send command.

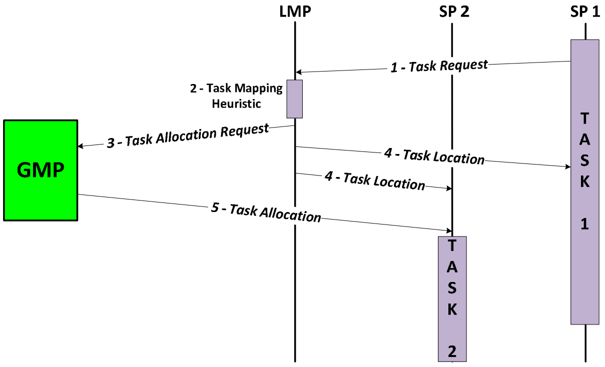

Figure 4 – Protocol to map one task.

Receiving the *task request*, the LMP executes the PREMAP-DN multi-task mapping heuristic [9] to select the SP to receive *task_2* ('2' in Figure 4). This mapping heuristic minimizes the energy consumed in the NoC by approximating the tasks with higher communication volume. The overall system performance is improved by distributing the mapping heuristic in several LMPs, since it is a time-consuming process. Suppose the task mapping heuristic selected SP 2 to receive *task_2*. Thus, the LMP sends a packet to the GMP with the service *task allocation request* ('3' in Figure 4). The LMP also sends to SP 1 the location of *task_2* and to SP 2 the location of *task_1* ('4' in Figure 4). These locations are stored in the local task tables. The last event in the task mapping process is the transmission of the task code by the GMP to the SP 2 ('5' in Figure 4).

C. Centralized versus Distributed Task Mapping

In the centralized mapping, the GMP maps all tasks. All incoming task mapping requests are serialized (Figure 5(a)), reducing the system performance, and increasing the NoC traffic in the GMP region. Using the distributed task mapping (Figure 5(b)), the mapping computation is distributed in several LMPs, reducing the communication load generated by mapping requests.

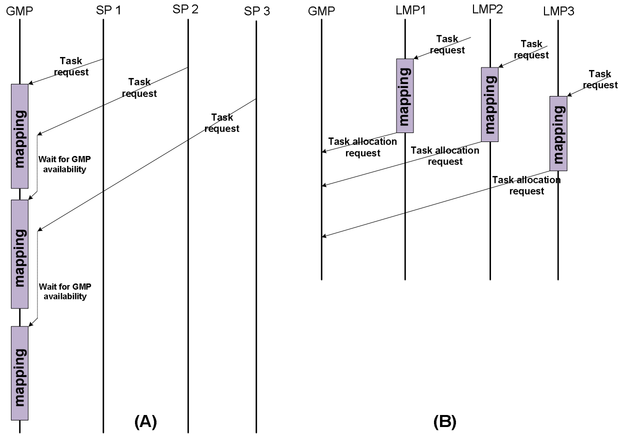

Figure 5 – Centralized (a) versus distributed (b) mapping.

It is important to mention two limitations of the distributed approach. Even if the mapping is distributed, the access to the external world (task repository) is not. The impact of this issue is minimized by transmitting the task data in burst, using a DMA approach. The second limitation is related to the present implementation, which does not include the re-clustering procedure. Therefore, all applications must fit in the clusters. The re-clustering procedure is an ongoing work.

IV. RESULTS

A cycle accurate system model of HeMPS was employed [10]. Each cluster contains 8 SPs (3x3 clusters), and each SP can execute up to 2 simultaneous tasks. Therefore, each cluster may execute 16 tasks (see Equation 1). Three benchmarks were evaluated: *MPEG*, executes a partial MPEG decoder; *multispec* image analysis [11], which evaluates the similarity between 2 images using different frequencies; and, a synthetic application (*synthetic*).

Table I presents nine evaluated scenarios (A, B, …, I). The second column of the Table I contains the MPSoCs size, the number of clusters, and the number of SPs for the distributed and centralized management approaches. Note that the centralized approach has more SPs than the distributed one, since it does not use LMPs. The third column presents the number of tasks for each benchmark, while the forth column shows the number of application instances (App_{CL}) that fit in the cluster. The fifth column contains the total number of tasks that must be mapped onto the MPSoC platform. The last two columns present the system usage for the distributed and centralized approaches, i.e., the percentage of used system resources (number of tasks / (number of SPs * 2)).

All applications instances are inserted in the system at the same time (1 ms) to maximize the use of SPs. Table II presents the total execution time reduction, adopting the centralized mapping as reference. Such results are a clear demonstration of the poor scalability related to centralized management of the MPSoC resources. Scenario A does not reduce the total execution time using the distributed mapping due to the smaller system utilization. As the centralized approach has more free resources, some SPs may receive one task, instead two, reducing the application execution time, since there is no time-sharing between tasks. Note that this behavior also occurs in scenarios D and G, where the synthetic benchmark presents smaller gains than scenarios E/F and H/I, respectively. The mapping process in the distributed approach has a smaller search space. For example, in the 12x12 MPSoC the centralized mapping has to evaluate the status of 143 SPs, while in the distributed mapping the search space is always the same, proportional to the cluster size. Therefore, the execution of the mapping heuristic is faster in the distributed version.

TABLE I. CHARACTERISTICS OF THE EVALUATED SCENARIOS.

	MPSoC Size	Benchmark - Nb. of Tasks	App_{CL}	Total number of tasks	SU_{dist}	SU_{centr}
A	**6x6** - 4 clusters	Syntetic - 6	2	48	75%	69%
B	- 32 SPs (distributed)	MPEG - 5	3	60	94%	86%
C	- 35 SPs (centralized)	Multispec -14	1	56	88%	80%
D	**9x9** - 9 clusters	Syntetic - 6	2	108	75%	68%
E	- 72 SPs (distributed)	MPEG - 5	3	135	94%	84%
F	- 80 SPs (centralized)	Multispec -14	1	126	88%	79%
G	**12x12** - 16 clusters	Syntetic - 6	2	192	75%	67%
H	- 128 SPs (distributed)	MPEG - 5	3	240	94%	84%
I	- 143 SPs (centralized)	Multispec -14	1	224	88%	78%

TABLE II. TOTAL EXECUTION TIME REDUCTION, ADOPTING THE CENTRALIZED MAPPING AS REFERENCE.

Scenario	MPSoC Size	Benchmark	Execution time reduction (w.r.t centralized mapping)
A	6x6	Synthetic	**-15% (increase of time)**
B		MPEG	**28%**
C		Multispect	**34%**
D	9x9	Synthetic	**50%**
E		MPEG	**63%**
F		Multispect	**54%**
G	12x12	Synthetic	**79%**
H		MPEG	**86%**
I		Multispect	**85%**

Figure 6 presents the clock cycles when each application instance starts and finishes, for scenarios B and E.

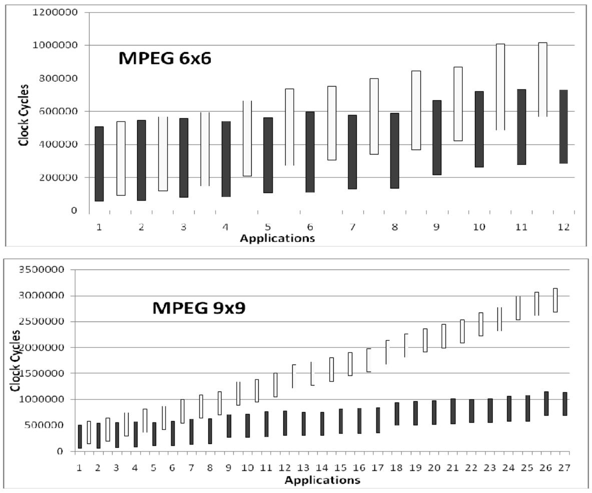

Figure 6 – Distributed (black bars) versus centralized mapping (white bars), for the MPEG benchmark with two NoC sizes, scenarios B and E.

It is important to observe in Figure 6 that all application instances have roughly the same execution time. Note that in the distributed mapping (black bars) a set of applications starts simultaneously. This is due to the distributed computation of the mapping heuristic, as illustrated in Figure 5(b). On the other hand, the centralized mapping (white bars) has to map the application tasks (5 tasks in the MPEG benchmark) and treat the request for new applications. This serialization of the mapping process is clearly observed in both figures, which also explain the results observed in Table II. As the MPSoC size increases, the execution time for the centralized mapping grows dramatically.

Another benefit of the distributed mapping is the reduction of the traffic around the GMP. Most control messages are treated inside the clusters, and the only control message sent to the GMP is the task request.

V. CONCLUSIONS AND FUTURE WORK

This paper proposed a distributed resource management scheme aiming to provide scalability to MPSoCs. The proposed approach divides the system in fixed-size clusters controlled by a local manager, reducing the computational load and traffic compared to a centralized management. The proposal was evaluated using accurate cycle-based simulation. Results demonstrate the effectiveness on distribute management using task mapping as case study. The negative aspect of the proposal is that some processors are reserved for resources management. In the Authors opinion, this is not a real drawback, since the percentage of reserved processors is low.

Future works include: (i) *re-clustering*, allowing modifying the cluster size at runtime, if a given application requires more resources than the available one in the cluster; (ii) *task migration*, to move tasks near to their neighbor tasks, to improve the application performance and to reduce the communication energy in the NoC.

VI. REFERENCES

[1] "The Intel® Xeon Phi™ Coprocessor". Available at: http://www.intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html

[2] Borkar, S. "Thousand core chips: a technology perspective"**.** In: DAC, 2007, pp. 746-749.

[3] Al Faruque, M.A.; Krist, R.; Henkel, J. "ADAM: Run-time Agent-based Distributed Appli-cation Mapping for on-chip Communication". In: DAC, 2008, pp. 760-765.

[4] Anagnostopoulos, I.; Bartzas, A.; Kathareios, G.; Soudris, D. "A divide and conquer based distributed run-time mapping methodology for many-core platforms". In: DATE, 2012, pp. 111-116.

[5] Cui, Y; Zhang, W; Yu, H. "Decentralized Agent Based Re-Clustering for Task Mapping of Tera-Scale Network-on-Chip System". In: ISCAS, 2012, pp. 2437-2440.

[6] Kobbe, S.; Bauer, L.; Lohmann, D.; Schroder-Preikschat, W.; Henkel, J. "DistRM: Distributed Resource Management for On-Chip Many-Core Systems". In: CODES+ISSS, 2011, pp. 119-128.

[7] Weichslgartner, A.; Wildermann, S.; Teich J. "Dynamic decentralized mapping of tree-structured applications on NoC architectures". In: NOCs, 2011, pp. 201-208.

[8] Shabbir, A.; Kumar, A.; Mesman, B.; Corporaal, H. "Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms". In: SAMOS, 2011, pp. 132-139.

[9] Mandelli, M.; Amory, A.; Ost, L.; Moraes, F.; "Multi-task dynamic mapping onto NoC-based MPSoCs". In: SBCCI, 2011, pp. 191-196.

[10] Carara, E. A.; Oliveira, R. P.; Calazans, N. L. V.; Moraes, F. G. "HeMPS - A Framework for Noc-Based MPSoC Generation". In: ISCAS, 2009, pp. 1345-1348.

[11] Tan, J.; Zhang, L.; Fresse, V.; Legrand, A.; Houzet, D. "A predictive and parametrized architecture for image analysis algorithm implementations on FPGA adapted to multispectral imaging". In: Int. Workshop on Image Processing Theory, Tools and Applications, 2008, pp 1-8.