Energy-Aware Dynamic Task Mapping for NoC-based MPSoCs

Marcelo Mandelli, Luciano Ost, Everton Carara, Guilherme Guindani, Thiago Gouvea, Guilherme Medeiros and Fernando G. Moraes
FACIN - PUCRS - Av. Ipiranga 6681- Porto Alegre - 90619-900 – Brazil
fernando.moraes@pucrs.br

***Abstract*—To cope with the dynamic workload of actual NoC-based MPSoCs, dynamic mechanisms are required to guarantee the application requirements. Application mapping may drastically influence the system performance and the energy consumption, which can be crucial to the success (or failure) of a product, even more for battery-powered embedded systems. In this context, the current work presents an energy-aware dynamic task mapping heuristic, which was evaluated in a real NoC-based MPSoC platform. Results show that the proposed heuristic may reduces up to 22.8% of the communication energy consumption compared to other dynamic mapping heuristics.**

Key words: *MPSoC, NoC, dynamic mapping. energy evaluation*.

I. INTRODUCTION AND RELATED WORK

NoC-based MPSoCs can execute simultaneously several and complex applications, as telecommunication protocols, multimedia streaming, GPS, and gaming. The workload of such systems may vary dynamically at execution time, according to various criteria (e.g. user and/or performance requirements) [1][2][3]. Such applications are composed by tasks, with different workloads and deadlines [4]. In this context, the mapping of such tasks into the NoC-based platform may drastically influence the system performance due to the traffic between tasks [5]. An unoptimized mapping may place communicating tasks far from each other, increasing the communication latency and energy, and also increasing network traffic leading to congestion inside the NoC.

Considering the moment when task mapping is executed, approaches can be either static or dynamic [5]. Static mapping defines task placement at design time, having a global view of the MPSoC resources. As it is executed at design time, it may use complex algorithms to better explore the MPSoC resources, resulting in optimized solutions. However, static mapping is not able to handle a dynamic workload, i.e., new applications loaded at run-time. To cope with this feature of actual MPSoCs, dynamic mapping defines the place of each task at runtime.

Wildermann et al. [1] evaluate the benefits of using a run-time mapping heuristic (communication and neighborhood cost functions), which allows decreasing the communication overhead. This heuristic was validated using master/slave applications executing on a homogeneous NoC-based MPSoC. The simulation environment (SystemC-based) can dynamically create tasks that are mapped into the processing elements (PEs) during the simulation, emulating a dynamic workload. Hölzenspies et al. [4] investigate another run-time spatial mapping technique, considering streaming applications mapped onto heterogeneous MPSoCs, aiming on reducing the energy consumption imposed by such application behaviors. Schranzhofer et al. [6] suggest a dynamic strategy based on pre-computed template mappings (defined at design time), which are used to define newly arriving tasks to the PEs at run-time.

Carvalho at al. [5] evaluate pros and cons of using dynamic mapping heuristics (e.g. path load and best neighbor), when compared to static ones (e.g. simulated annealing and Taboo search). The Authors adopted energy consumption and latency as performance metrics, which were evaluated in a heterogeneous NoC-based MPSoC model (RTL NoC and abstract PEs implemented in SystemC), regarding different application scenarios. Carvalho's approach was extended by Singh et al. [2][7], employing a packing strategy, which minimizes the communication overhead in the same NoC-based MPSoC platform. Additionally, Singh's approach was improved to support multi-task mapping onto the same PE. Different mapping heuristics were used to evaluate the performance. According to the Authors, the communication overhead of the whole system is reduced, decreasing the energy consumption.

In the previous work, one PE is responsible to execute the mapping – centralized approach. Faruque et al. [3] propose a decentralized agent-based mapping approach, targeting larger heterogeneous NoC-based MPSoCs (a 32x64 system is used as case study). The decentralized approach reduces the monitoring traffic (10.7 times) and the computational effort for the mapping (7.1 times) when compared to a centralized and to non-clustered centralized approaches, respectively. Note that the PE responsible to map the tasks should read them from an external memory (task repository), and then transmit these tasks to the PEs that will execute them. In the decentralized approach the bottleneck is transferred from the PE responsible to map the tasks to the memory interface, since several PEs will try to simultaneously access the task repository. A global evaluation of a decentralized approach execution in a real MPSoC is missing in the literature.

The main goal of this paper is to present an energy-aware heuristic for dynamic task mapping, named lower energy consumption based on dependencies-neighborhood (LEC-DN). The main features of our work include: (*i*) execution of the mapping heuristics in an RTL-modeled NoC-based MPSoC, leading to accurate results; (*ii*) real applications are used as benchmarks; (*iii*) previous dynamic heuristics considers only a master-slave dependence – the proposed LEC-DN heuristic extends the dependences to all communicating tasks; (*iv*) the main cost function of the previous heuristics was the distance in hops between

communicating tasks – the proposed LEC-DN heuristic extends the cost function to the communication volume, since the number of transmitted flits defines the communication energy.

II. NoC-based MPSoC Architecture and Task Mapping Protocol

HeMPS [8] is a homogeneous MPSoC, described in synthesizable VHDL, in which PEs are connected to the Hermes NoC. Each PE, named Plasma-IP, contains a MIPS-like processor (Plasma), a local memory (RAM), a DMA controller and a Network Interface (NI). Figure 1 illustrates a general view of a 2x2 instance of the HeMPS architecture.

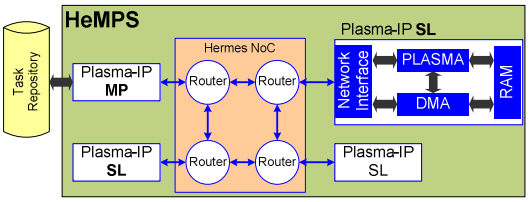


Figure 1 -Block diagram of 2x2 instance of the HeMPS Platform.

Two types of Plasma-IPs are used: slaves (SL) and master (MP). Plasma-IP SLs are responsible to execute application tasks. Each Plasma-IP SL runs a tiny operating system (named microkernel), responsible to manage and support task execution and task communication. This microkernel is a fully preemptive operating system where each task uses the CPU for a pre-defined period of time named *timeslice*. This microkernel supports multitasking and software interrupts (traps). The Plasma-IP MP is responsible to manage task mapping and system debug. The external memory, named *task repository*, contains all application tasks. According to the mapping heuristic, the Plasma-IP MP maps the tasks into the Plasma-IP SLs.

All communication among tasks occurs through message passing. The message passing is implemented using a global message pipe located in the microkernel, and two communication primitives: *Send*() and *Receive*() [8]. Each application has at least one initial task, (without data dependency) and its placement is manually defined (in any free PE). A task executing the communication primitive *Send*() is the source, and the one executing *Receive*() is the target. If the target task is not already mapped, the source microkernel sends a *RequestTask* to the Plasma-IP MP, which selects the task position at run-time according to the dynamic task mapping heuristic (as LEC-DN, described in Section III). This protocol is illustrated in Figure 2, where

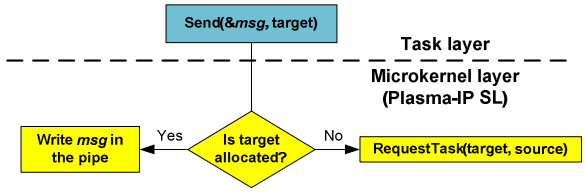


Figure 2 - Dynamic task mapping protocol in HeMPS.

When the Plasma-IP MP receives the *RequestTask* message, it configures its DMA module, which accesses the task repository and transmits the *target* task code to the target Plasma-IP SL memory. After this task transmission is finished, the Plasma-IP MP notifies all slaves with the *target* ID and its new position.

III. LEC-DN Mapping Heuristic

This section first presents the reference mapping heuristics – NN (Nearest Neighbor), BN (Best Neighbor) and PL (Path Load) [5]. In the sequel, it is presented the new mapping heuristic, LEC-DN.

A. Reference dynamic mappings heuristics

The Nearest Neighbor (NN) heuristic considers only the proximity of an available resource to execute a given task. NN starts searching for a free PE able to execute the *target* task near the *source* task. The search tests all *n*-hop neighbors, *n* varying between 1 and the NoC limits in a spiral way, stopping when the first PE free is found. The Path Load (PL) heuristic computes the load in each channel used in the communication path. PL computes the cost of the communication path between the *source* task and each one of the available resources. The selected mapping is the one with minimum cost. The Best Neighbor (BN) heuristic combines NN search strategy with the PL computation approach. The search method of BN is similar to NN, i.e., spiral searches from the source node. This avoids computing all feasible mapping solutions, as in the PL heuristic, reducing the execution time for the mapping. BN selects the best neighbor, according to PL equations, instead of the first free neighbor as in NN.

B. LEC-DN

LEC-DN heuristic employs two cost functions: (*i*) proximity, in number of hops; (*ii*) communication volume among tasks. Differently from NN and BN heuristics, which map the *target* task as close as possible to its *source* task, the LEC-DN considers the proximity of the *target* task to all tasks it communicates already mapped. The second criterion is used when a given task communicates with at least two mapped tasks. In this situation, the new task is mapped closest to the task with higher communication volume.

LEC-DN employs two search methods to select the PE to receive the *target* task. When the *target* task has only one communicating task already mapped, LEC-DN uses the NN search method (spiral search). If there is more than one communicating task already mapped, the LEC-DN searches for a PE inside the bounding box defined by the position of such tasks. Consider the application of Figure 3(a), containing 4 tasks, where A and B are initial tasks. The mapping of task C is fired by the first communication with it, according to the protocol illustrated in Figure 2. The search space to map task C corresponds to the bounding box defined by the position of A and B tasks (Figure 3(b)). Task C will be mapped nearest to task A, since according to the application graph the communication volume A→C is higher than B→C. Note that task D is not yet mapped, since it depends from task C.

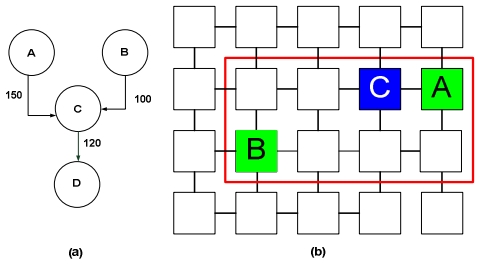


Figure 3 – (a) application graph G=<T,V> describing an application, where T is the task set and V the communication volume between tasks; (b) search space to map the C task, and one possible mapping for C.

The second search method uses the volume-based energy model proposed by Hu et al. [9] to select the position of the task to be mapped. Equation 1 computes the communication energy spent to transit 1 bit through a distance of *n* hops.

$$E_{bit}^{hops} = n_{hops} * E_{Sbit} + (n_{hops} - 1) * E_{Lbit} \quad (1)$$

where: E_{Sbit}, E_{Lbit} and n_{hops} corresponds to the energy consumption of the router, in the interconnection wires and the number of hops to transmit 1 bit, respectively.

Figure 4 shows the implementation of the LEC-DN heuristic. The heuristic starts by inserting the mapped tasks that communicate with the *target* task t_i in the *communicating_tasks_list* (line 3). If the *communicating_tasks_list* has just one element (*c0*), the NN algorithm is executed (lines 6-19). If the *communicating_tasks_list* has more than one communicating task mapped (lines 21-44), the algorithm first set the energy to a maximum value (line 22), and defines the coordinates of the bounding box according to the elements inside the *communicating_tasks_list* (line 23). Line 26 gets all PEs inside the bounding box. The loop between lines 27 to 39 computes the communication energy to map each PE inside the bounding box (line 32), selecting the PE that results in the mapping with the smallest energy cost. The bounding box size is increased one hop in all directions (line 41), if there is not a free PE inside the bounding box. The loop between lines 24-43 is repeated until finding an available *pe*, or all PEs were visited. If no *pe* is found, the PLASMA-IP MP schedules the task t_i to be mapped when a PE becomes available.

```
Input: A task ti to be mapped, the set Ti containing the tasks ti communicates with
Output: The PE to map the task pe
1.    pe ← -1
2.    // Get all ti communicating tsks already mapped
3.    communicating_tasks_list ← mapped_tasks(Ti)
4.    // If there is only one communicating task
5.    IF size(communicating_tasks_list) = 1 THEN
6.        dist ← 1    // Initializes the search distance to 1 hop
7.        // Search until the distance dist = NoC_size
8.        WHILE dist!= NoC_size DO
9.            c0 ← first element of the communicating_tasks_list
10.           pe_list ← neighbours(dist, c0) // Get all neighbours of c0 within a distance dist
11.           FOR ALL ELEMENTS pi IN pe_list
12.               // Checks PE occupation
13.               IF state(pi) = free THEN
14.                   pe ← pi
15                    return pe // PE is found, stop searching
16.               END IF
17.           END FOR
18.           dist ← dist + 1    // Go to next range of neighbours
19.       END WHILE
20.   END IF
21.   IF size(communicating_tasks_list) > 1 THEN
22.       min_energy ← ∞                          // Initializes energy with highest value
23.       bounding_box ← area(communicating_tasks_list)   // set coordinates of the bounding box
24.       WHILE bounding box<= NoC_size and pe=-1 DO
25.           // get the PEs inside the bounding box
26.           pe_list ← search_PEs(bounding_box)
27.           FOR ALL ELEMENTS pi IN pe_list
28.               //Checks PE occupation
29.               IF state(pi) = free THEN
30.                   energy ← 0
31.                   FOR ALL ci IN communicating_tasks_list
32.                       energy ← energy + communication_volume(ci,pi)*distance(ci,pi)
33.                   END FOR
34.                   //  Set the minimum energy and the target PE
35.                   IF energy_pe < min_energy THEN
36.                       min_energy ← energy_pe
37.                       pe ← pi
38.                   END IF
39.           END FOR
40.           IF pe = -1 THEN
41.               increase(bounding_box)
42.           END IF
43.       END WHILE
44.   END IF
45.   return pe
```

Figure 4 - LEC-DN mapping heuristic pseudocode.

IV. APPLICATION MODELING

Real and synthetic applications are modeled in the present work using hierarchical UML diagrams and actor orientation [10], which provides several features for application development and modeling. According to the UML 2.0 specification, designers can model sequence diagrams by using combined fragments and interaction operators, such as option (opt), and parallel (par). Thus, applications can be specified with a communication behavior that includes parallel, optional, and iterative triggering of messages, providing more modeling flexibility than graph-based approaches. Figure 5 illustrates the sequence diagram of a segmentation image application, with 6 tasks. Each task is a lifeline (vertical lines, as task ME). Communication behavior is represented by message exchanges between lifelines, as message *m1* sent from *ME* to *P0*. The *par* blocks contain a set of communications that can be trigged in parallel, as *m17* to *m20*.

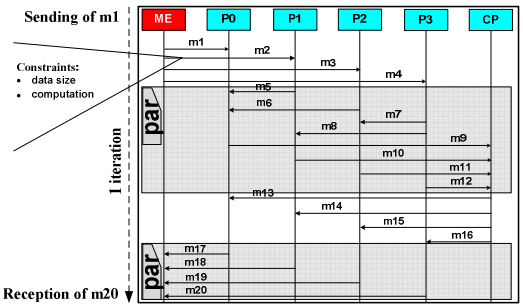


Figure 5 - Segmentation Image sequence diagram.

This abstract application model is simulated in an extension of the Ptolemy environment [11], to verify its correctness. Once the application is correctly modeled, a C code for each task is automatically generated. This C code contains the communication primitives (*Send() and Receive()*) enabling the communication among tasks, and the computation time of each task. Such method enables to execute real benchmarks in the MPSoC platform, linking the abstract modeling to the RTL model.

The application graph (Figure 3(a)), used during the mapping phase could be obtained directly from the UML diagram. However, it is directly extracted from the C code, parsing the communication primitives. This choice enables the designer to get the application graph either from UML diagrams, as well as standard C benchmarks.

V. RESULTS

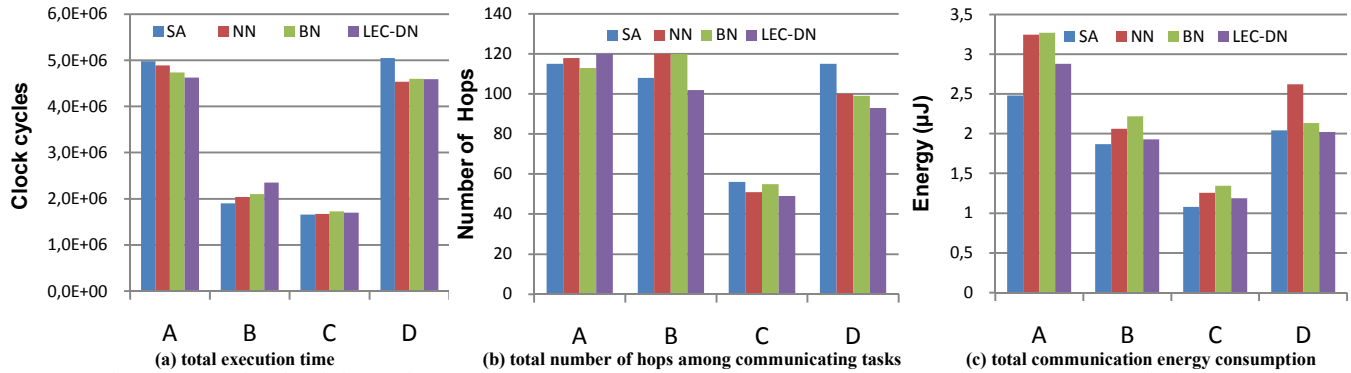
This Section evaluates the mappings heuristics. The HeMPS MPSoC is configured as follows: 7x6 mesh topology (42 PEs), XY routing algorithm, 16-bit flit size, packets with 128 flits and credit-based control flow. The energy model [12] was calibrated using the ST/IBM CMOS 65 nm technology at 1.0 V, adopting clock-gating, and a 100 MHz clock frequency. Additionally, to evaluate the BN heuristic a monitoring infrastructure is included in HeMPS to obtain the load in each NoC link.

Four application scenarios were evaluated:

A. MPEG (12 tasks), Video Object Plane Decoder (VOPD) (12 tasks), Vehicle (10 tasks) and Circuit (4 tasks);
B. MPEG, VOPD, Segmentation Image (6 tasks) and Synthetic (6 tasks);
C. MPEG and VOPD;
D. MPEG, Vehicle and Circuit.

The MPEG-4 application contains as relevant feature a high degree of connections: 2 of the 12 tasks are connected to up 7 other tasks. The VOPD application presents smaller inter-task dependency when compared to MPEG-4. Vehicle and Segmentation Image applications present several parallel communications. Circuit and Synthetic are synthetic applications, presenting dataflow behavior.

Scenarios *A* and *B* contains 38 and 36 tasks, respectively. Considering that the MPSoC contains 41 PLASMA-IP SLs, such scenarios corresponds to an MPSoC occupation equal to 93% and 86%, respectively. The dynamic mapping heuristics are stressed in these two scenarios, since the search space is drastically reduced when almost all tasks are already mapped. Scenarios *C* and *D* contains 24 and 26 tasks respectively, enabling to evaluate the mapping heuristics when the search space is not a constraint.



(a) total execution time **(b) total number of hops among communicating tasks** **(c) total communication energy consumption**

Figure 6 – Mappings heuristics results (SA, NN, BN, LEC-DN) for scenarios A, B, C and D, mapped onto a 7x6 HeMPS platform.

The evaluated mapping heuristics are: SA (simulated annealing), static mapping; NN and BN, reference dynamic heuristics, and the proposed LEC-DN. Figure 6 presents the total execution time to execute 10 application iterations, the total number of hops among tasks and the total communication energy consumption. Note that the communication energy of the BN heuristic is underestimated, since the monitoring packets to evaluate the path load are not considered.

The execution time for the static mapping (SA) would be expected to be the best case, because all tasks are mapped in the start-up of the system execution. In fact the NoC becomes congested in this moment, penalizing the execution time. In the dynamic mapping the tasks are mapped when they are requested, reducing the network load. In scenarios A and D, where the execution time of the vehicle application dominates, the proposed LEC-DN reduces the execution time by 7.2% and 9.1%, respectively, compared to SA. Compared to NN and BN the execution time increased in average by 3.2% and 1.9%, respectively. It is important to mention that the execution time of the static mapping (SA) is not considered, since it is executed at design time. The execution time of the 3 dynamic heuristics are considered, since these heuristics run during system execution. Such results demonstrate dynamic mapping heuristics do no impact in the total execution time.

The total number of hops among communicating tasks is the metric used to evaluate the mapping quality in most mapping heuristics. It is noticeable in scenarios B, C and D that the proposed LEC-DN reduces the hop number compared the SA (108/102, 56/49 and 115/93 hops respectively). In scenarios C and D, with more free resources, all dynamic heuristics reduce the total hop number compared to the SA. These results show that LEC-DN is able to explore the available resources in an efficient way, generating optimized solution.

The last graph shows the total communication energy. Note that if only the total hop number is the parameter used to measure the mapping quality, unoptimized mapping can be selected, since the communication volume is not considered. The static mapping, even with a higher number of hops in some scenarios, is the mapping heuristic with the smallest energy consumption (in average the proposed LEC-DN consumes 7.1% more compared to the SA). This is an expected result, since such mapping heuristic has a global view of the application being mapped. Comparing LEC-DN to NN and BN, the average energy reduction is 11.4% and 10.4% respectively. In scenario D the energy reduction of LEC-DN compared to NN achieves 22.8%.

VI. CONCLUSION AND FUTURE WORK

This work improved the quality of the dynamic mapping for NoC-based MPSoCs including in the mapping heuristic two components: (*i*) cost of the already mapped tasks connected to the task being mapped, while previous approaches consider only master-slave connections; (*ii*) inclusion of the communication volume among tasks together with the number of hops, while previous approaches consider only the total number of hops. Such improvements reduced the energy consumption (11% in average) compared to two well known dynamic mapping heuristics (NN and BN). Compared to a static solution, which is not usable in a dynamic workload scenario, the execution time is not penalized, and the communication energy consumption overhead is an average only 7.1% higher.

Future works includes: (*i*) extension of LEC-DN to multitasking; (*ii*) evaluate the total MPSoC energy consumption, including the PEs consumption; (*iii*) evaluate strategies to decentralize the mapping, for instance including the mapping heuristic as a service of the microkernel.

ACKNOWLEDGEMENTS

This research was supported partially by CNPq and FAPERGS (Brazilian Research Agencies), projects 301599/2009-2 and 1008149.

REFERENCES

[1] Wildermann,S.; Ziermann,T.; Teichet,J. “Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures”. In: FPT'09, pp. 514 - 517.

[2] Singh,A.; Srikanthan,T.; Kumar,A.; Jigang,W. "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms". Journal of Systems Architecture, vol. 56(7), 2010, pp. 242 - 255.

[3] Faruque,M.; Krist,R.; Henkel,J. “ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication”. In: DAC'08, 2008, pp. 760-765.

[4] Hölzenspies,P.; Hurink,J.; Kuper,J.; Smit,G. “Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)”. In: DATE'08, 2008, pp.212-217.

[5] Carvalho,E.; Calazans,N.; Moraes,F. “Dynamic Task Mapping for MPSoCs”. IEEE Design & Test of Computers, v. 27, p. 26-35, 2010.

[6] Schranzhofer,A.; Jian-Jia,C.; Santinelli,L.; Thiele, L.. “Dynamic and adaptive allocation of applications on MPSoC platforms”. In: ASP-DAC, 2010, pp. 885 – 890.

[7] Singh, A.K. et al. “Eficient heuristics for minimizing communication overhead in noc-based heterogeneous mpsoc platforms”. In: RSP’09, 2009, pp. 55 - 60.

[8] Carara,E.; Oliveira,R.; Calazans,N.; Moraes,F."HeMPS - a Framework for NoC-based MPSoC Generation". In: ISCAS’09, 2009, pp. 1345 - 1348.

[9] Hu, J.; at al. “Energy-aware mapping for tile-based NoC architectures under performance constraints”. In: ASP-DAC’03, 2003, pp. 233-239.

[10] Indrusiak, L.S.; et al. "Executable System-Level Specification Models Containing UML-Based Behavioral Patterns". In: DATE'07, 2007, pp. 301-306.

[11] Määttä, S.; et al. "Characterising Embedded Applications using a UML Profile". SoC'09, 2009, pp. 172-175.

[12] Ost, L.; Guindani, G. ; Indrusiak, L.; Määttä, S; Moraes, F. “Using Abstract Power Estimation Models for Design Space Exploration in NoC-based MPSoC”. IEEE Design & Test of Computers (Preprint), 2010.