

Dynamic Task Mapping for MPSoCs

Ewerson Luiz de Souza Carvalho, Ney Laert Vilar Calazans, and Fernando Gehm Moraes

Pontifícia Universidade Católica do Rio Grande do Sul

Multiprocessor-system-on-a-chip (MPSoC) applications can consist of a varying number of simultaneous tasks and can change even after system design, enforcing a scenario that requires the use of dynamic task mapping. This article investigates dynamic task-mapping heuristics targeting reduction of network congestion in network-on-chip (NoC)-based MPSoCs. The proposed heuristics achieve up to 31% smaller channel load and up to 22% smaller packet latency than other heuristics.

In this article, we investigate the performance of mapping algorithms for heterogeneous MPSoCs targeting dynamic workloads. Our main objective is to minimize congestion inside the network on chip (NoC) through channel usage optimization. The performance parameters we look at include execution time, NoC channel load,

and packet latency. We also compare static and dynamic mapping approaches in terms of their advantages and drawbacks in application scenarios that we simulated in two sets of experiments.

■ **THE EVOLUTION OF** deep-submicron technology has dramatically increased IC density, enabling the development of multiprocessor systems on a chip (MPSoCs), SoCs containing multiple processing elements (PEs). A homogeneous MPSoC contains a set of identical PEs, typically programmable processors. Whereas homogeneous MPSoCs simplify task migration, heterogeneous MPSoCs support a wider variety of applications because they integrate distinct PEs. Recently, Intel developed a homogeneous MPSoC with 80 PEs, following a trend of growing MPSoC complexity.

Applications (such as multimedia and networking) running in MPSoCs generally present a dynamic task workload. This implies a varying number of tasks running at any given moment, possibly exceeding the available hardware resources. Thus, it is necessary to control task operation and system resource use, including dynamic management of task-loading actions.

Task mapping—which consists of finding a placement for a set of tasks that meets a specific requirement such as energy consumption savings—is an important issue. Mapping decisions can drastically influence system performance. If we use the moment at which a task is defined as a classification criterion, task-mapping approaches can be either static or dynamic. Static mapping defines task placement at design time; dynamic mapping defines task placement at runtime.

Static mapping techniques

Static mapping techniques are suitable for specific platforms that don't need the insertion of new applications into the system at runtime. Because the mapping algorithm's running time is not a main concern, static mapping can use thorough information about the system to make decisions. This information includes the application graph's topology, volume and rate of inter-task communication, and total system usage.

Research into static mapping includes the work of Lei and Kumar, who proposed a genetic mapping algorithm to optimize application execution time.¹ In their work, graphs represent applications and the target architecture is an NoC. Wu, Al-Hashimi, and Eles also investigated genetic mapping algorithms.² By combining dynamic voltage scaling techniques with mapping, they achieved a 51% savings in energy consumption. Murali and De Micheli explored mappings for more than one application in NoC design, using the tabu search (TS) algorithm.³ Manolache, Eles, and Peng investigated task mapping in NoCs, trying to guarantee packet latency.⁴ For this purpose, both the task-mapping algorithm (TS) and the routing algorithm are defined at design time. Hu and Marculescu

presented a branch-and-bound algorithm to map a set of IP cores (IPs) into an NoC with bandwidth reservation.⁵ Their results show energy savings of 51.7% in the communication architecture. Marcon et al. investigated how to map modules into an NoC, targeting low energy consumption.⁶ They compared several algorithms, using a model that characterizes applications by their intertask communication volume.

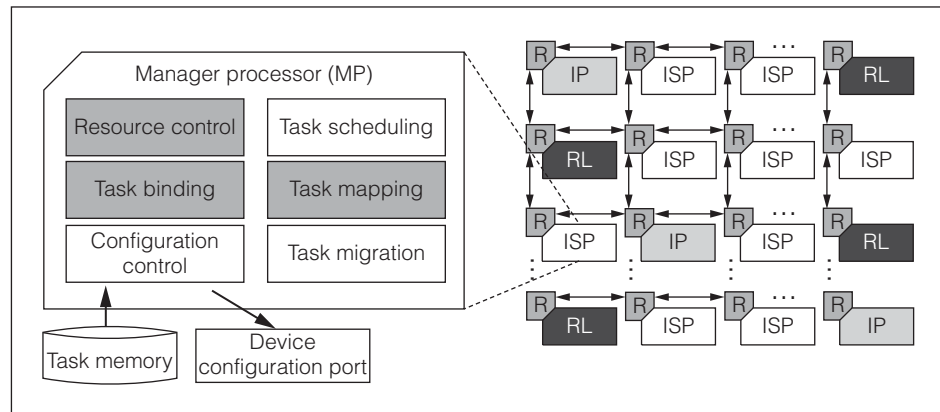


Figure 1. Conceptual MPSoC organization. The shaded blocks inside the MP are the modules addressed in this article. The Rs are NoC routers.

Dynamic mapping techniques

In dynamic mapping, the time taken to map each task is crucial, since it influences overall application execution time. To reduce mapping overhead, it is important to use greedy algorithms because they can trade search space exploration quality for fast results. Although execution time for mapping is an important cost function, several works don't consider this overhead.^{7,8} These researchers assumed that overall system execution time compensates for the time spent mapping each task. On this assumption, many researchers used the same strategies in dynamic scenarios as those used in static scenarios. For example, Murali and De Micheli used simulated annealing (SA) algorithms for static mapping, and Ngouanga et al. used SA for dynamic mapping.^{3,7}

Smit, Hurink, and Smit presented an iterative hierarchical approach to mapping an application to an NoC-based SoC at runtime.⁸ A set of communicating tasks models the application. Aiming at energy consumption savings, the mapping algorithm tries to place each task near its communicating PEs. Similarly, besides the SA algorithm, Ngouanga et al. proposed a force-directed mapping algorithm, which aims to keep communicating tasks close to each other.⁷ Mehran, Khademzadeh, and Saeidi presented a mapping algorithm that searches for a placement following a spiral path, tending to place communicating tasks together.⁹ This method executes for a single application without cost function evaluation. Chou, Ogras, and Marculescu combined task-mapping and multiple-voltage-level schemes, allowing reduction of communication overhead and energy.¹⁰ They achieved a communication energy savings of up to 50%.

Al Faruque, Krist, and Henkel suggested a distributed agent-based mapping approach for larger MPSoCs, applying it to a 32×64 MPSoC.¹¹ The authors claim to achieve on average a 7.1 times lower computational effort for the mapping algorithm than with a simple nearest-neighbor heuristic.

Target MPSoC organization

A heterogeneous MPSoC is a set of different PEs interacting through a communication network. In our proposed NoC-based heterogeneous MPSoC model, each PE supports the execution of one task at a time. PEs can support either hardware or software task execution. Software tasks execute in instruction set processors (ISPs), and hardware tasks in reconfigurable logic (RL) or dedicated IPs.

Figure 1 illustrates the proposed heterogeneous MPSoC organization. Among the available resources, one processor—the manager processor (MP)—is responsible for resource control, configuration control, and four task manipulation issues: scheduling, binding, mapping, and migration. The MP fires each application's initial task. When a running task needs to communicate with another task not yet mapped, the latter is loaded in a PE from task memory.

A directed graph, in which vertices represent software or hardware tasks and edges define communicating PE pairs, models each application. Each edge defines a master-slave communication. The edge source vertex is the master of the communication, and the other is the slave. Each edge receives four parameters, which represent the volume and rate of data sent from master to slave and from slave to master. Similar models are common in several mapping algorithms.^{5,6,10} However, unlike our algorithm, these

algorithms consider only the communication volume for the application characterization graph (ACG),⁵ ignoring transmission rate information.

Intertask communications use messages transmitted through the network. Four message types comprise the adopted communication protocol. A *request* message from a running task to the MP contains the identification of a new task to be inserted in the system. A *release* message notifies the MP that a processing PE has finished its current task, freeing this PE for reuse by another task. After task mapping, the MP sends a *notify* message to both tasks involved in the communication that fired the mapping. This message contains tasks' current addresses, enabling correct packet transmission. Intertask communications employ *general* messages, which carry data some pairs of tasks need to exchange.

Dynamic mapping algorithms

The task-mapping problem is directly reducible to the quadratic assignment problem, a well-known NP-hard problem. Using heuristics to solve this problem at runtime is thus mandatory for even moderate-sized NoCs. The main cost function of the four dynamic mapping heuristics we propose here is the reduction of NoC channel congestion.

Each application has an initial task, which is the only one initially mapped when the application starts. Remaining application tasks are mapped on demand according to communication requests. The mapping of initial application tasks uses a clustering strategy that partitions the MPSoC into virtual regions called clusters, each initially assigned to a single application. Only one initial task can be mapped in each cluster, thus reducing the probability of tasks that belong to different applications sharing the same NoC region. This also reduces the sharing of NoC channels by communications of distinct applications, in contrast to several other mapping strategies.^{1,5,11} The clustering step is executed at system initialization, when specific PEs are statically defined to map initial tasks for a set of applications. If the number of applications exceeds the available clusters, some applications are scheduled to be mapped when resources become available.

Mapping all tasks at once might provide better task-mapping solutions by making it possible to consider global information about resource utilization and the entire application graph structure. However, in a dynamic workload scenario, it is not possible to define when each task will be needed.

We use two reference mapping methods: *first free* (FF) and *nearest neighbor* (NN). FF starts at resource R_{00} , walking the network column by column, from the bottom up. FF selects the first free resource according to binding definitions, without considering other metrics. FF might generate the worst results compared with the other heuristics presented here; however, we would expect even worse solutions if we used random mappings as a reference.¹⁰

Similar to FF, the NN method does not consider other metrics except the proximity of an available resource that can execute the required task. NN starts searching for a free node to execute the task near the requesting task. The search tests all n -hop neighbors, n varying spirally between 1 and the NoC limits, stopping when it finds the first resource that can execute the task.

The first heuristic proposed here is *minimum maximum channel load* mapping. MMCL evaluates all possible mappings for each new task inserted in the system. Its goal is to globally minimize channel usage peaks, reducing the occurrence of hot spots.

The second heuristic, *minimum average channel load*, aims at reducing the average usage of the NoC channels. MACL is similar to MMCL, replacing the maximum with the average function. While the MMCL heuristic tries to minimize channel usage peaks, MACL tries to homogeneously distribute the communication load in the NoC. The selected mapping is the one resulting in the lowest average channel usage.

The MMCL and MACL heuristics consider all NoC channels while mapping a new task. Since this evaluation can take a long time, the third heuristic, the *path load* (PL) algorithm, considers only the path segments used by the task under mapping. A path segment groups a set of network channels, defined as a function of the task placement and the routing algorithm. Given two tasks, A and B , a communication path between these tasks is represented by $CP(A, B)$. This path is composed of four segments individually taken from east, west, south, and north channels. A path segment can be empty—for example, if the master-slave pair maps to the same column or row of PEs. PL computes the cost of each mapping k , according to its respective path segments. For this purpose, PL adds the rates of the new task to the current rates in each channel in CP . The selected mapping is the one with minimum cost.

```

Inputs: A task  $t_i$  to be mapped, the master task  $m_i$ , the PE occupation matrix, and the NoC
channel occupation matrices
Output: The best PE -  $best\_pe$ 
1.   $dist \leftarrow 1$  // Initializes # of hops from  $m_i$  (smallest)
2.   $cost \leftarrow \infty$  // Initializes cost with highest value
3.  WHILE  $cost = \infty$  DO
4.    // Get all  $m_i$  neighbors within a distance  $dist$ 
5.     $pe\_list \leftarrow neighbors(dist, m_i)$ 
6.    FOR ALL ELEMENTS  $p_i$  IN  $pe\_list$ 
7.      // Tests PE occupation and type (i.e., task binding)
8.      IF  $state(p_i) = free$  and  $type(p_i) = type(t_i)$  THEN
9.        // Computes path load cost in NoC channels
10.        $cost\_pl \leftarrow 0$ 
11.        $channel\_list \leftarrow routing\_path(p_i, m_i)$ 
12.       FOR ALL  $c_i$  IN  $channel\_list$ 
13.          $cost\_pl \leftarrow cost\_pl + load(c_i)$ 
14.       END FOR
15.       // Set the minimum cost and the target PE
16.       IF  $cost\_pl < cost$  THEN
17.          $cost \leftarrow cost\_pl$ 
18.          $best\_pe \leftarrow p_i$ 
19.       END IF
20.     END IF
21.   END FOR
22.    $dist \leftarrow dist + 1$  // Go to next range of neighbors
23. END WHILE
24. return  $best\_pe$ 

```

Figure 2. Best neighbor (BN) mapping algorithm pseudocode.

PL evaluates all mapping possibilities. The fourth heuristic, *best neighbor* (BN), combines the NN search strategy and the PL computation approach. The BN search method is similar to NN, in its spiral searches from the source node. This avoids computing all feasible mapping solutions as in the PL heuristic, reducing the heuristic's execution time. BN selects the best neighbor, according to PL equations, instead of the first free neighbor as in NN.

To illustrate the heuristics' behavior, Figure 2 depicts the BN algorithm. Before executing, BN or another mapping algorithm must verify the existence of free resources compatible with the task to be mapped. If such resources are not available, the task is scheduled for later execution.

The algorithm starts by testing the closer neighbors—those at $dist = 1$ hop (line 1). If no feasible mapping is found during this iteration, the distance is incremented, testing a new group of neighbors (line 22). For this purpose, in each WHILE loop iteration, the function $neighbors(dist, m_i)$ computes the list of all PEs placed at $dist$ hops from task m_i (line 5). Next, for all PEs in this list, it is necessary to test the resource occupation (the free or used state)

and its compatibility (whether it is a hardware or a software task) with the task to be mapped (line 8). BN computes the cost only for feasible mappings.

The cost computation takes place at lines 11 to 14. To obtain the list of channels that comprise the communication path between m_i and p_i and back to m_i , the algorithm invokes function $routing_path(p_i, m_i)$ (line 11). This function returns a list of channels as dictated by the routing algorithm. The path cost ($cost_pl$) results from the accumulation of all channel loads in the list (lines 12 to 14). If the obtained cost is smaller than the previously computed cost, the cost is updated, and the same is true for the best PE (lines 16 to 19). Finally, when it finds at least one PE, the algorithm exits and returns the best PE.

The BN algorithm can produce either the PL or NN algorithms by using selected simplifications. First, eliminating lines 10 to 19 of BN and substituting them with line 24 produces the NN algorithm, which returns the first neighbor compatible with the test in line 8. For producing the PL algorithm from BN, the starting point is to suppress the WHILE loop (lines 3 to 5, 22, and 23). In addition, the list pe_list must contain all the MPSoC's PEs.

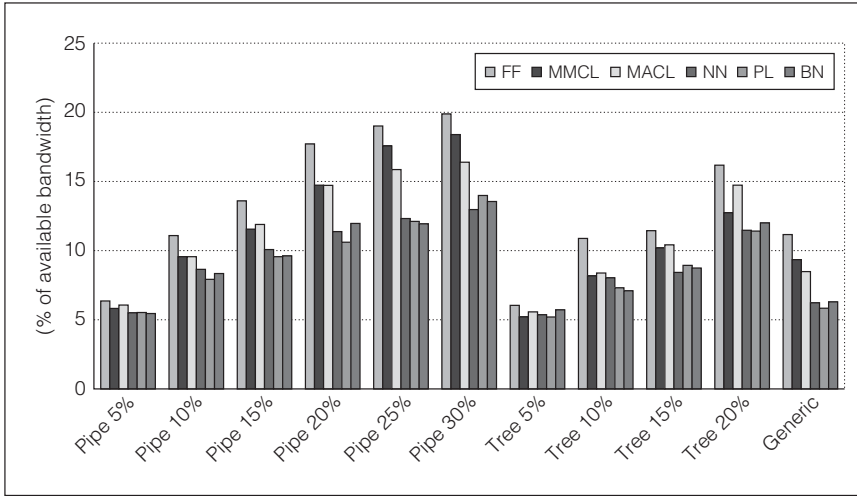


Figure 3. Average channel load (% of available bandwidth) for scenarios A, B, and C at varying injection rates.

Dynamic mapping evaluation

To evaluate the proposed dynamic mapping heuristics, we used a simulation environment that employs VHDL RTL code for the NoC description.¹² The NoC is a 2D, wormhole, packet-switched mesh architecture that uses XY routing. PEs are described in terms of SystemC RTL threads. To hold system usage status, one thread describes the MP, containing scheduling queues, channel occupation matrices, and the PE occupation matrix. A generic and parameterizable thread describes each PE. When a given task is mapped to a given PE, this thread reads a configuration file that describes the task's communication rates and volumes, as well as its execution time. With parameterization, PEs can emulate several tasks during system simulation. This strategy allows the addition of new applications to the system, as already proposed by Chou, Ogras, and Marculescu.¹⁰

For our first set of experiments, we used three simulation scenarios:

- **Scenario A (pipe)**—20 identical pipeline-like applications (such as typical dataflow applications), each with 10 tasks (seven software and three hardware) and injection rates varying from 5% to 30% of available channel bandwidth.
- **Scenario B (tree)**—20 identical tree-like applications (such as typical parallel benchmarks), each with 10 tasks (eight software and two hardware) and injection rates varying from 5% to 20%.

- **Scenario C (generic)**—20 different generic applications produced with the TGFF (Task Graphs for Free) tool, each application consisting of 5 to 10 tasks, with varying numbers of hardware and software tasks and injection rates randomly chosen from 5% to 30%.

For these three scenarios, we adopted an 8×8 heterogeneous MPSoC. One node (the router with its PE) was the MP, 16 nodes were hardware resources, and 47 nodes were software resources. To uniformly spread resources over the system area, a set of

PEs is reserved for initial tasks. The set of experiments arbitrarily assumed a maximum of 15 simultaneously running applications.

Figure 3 shows the various heuristics' *average channel load* (avg.), which represents NoC usage. All algorithms reduced average channel load in comparison with the FF method. MMCL and MACL reduced channel load by an average of 14% (with respect to FF), a result inferior to 29.95% achieved with the NN algorithm, which did not consider traffic during mapping but explored the proximity of communicating tasks. The BN heuristic had a gain similar to NN (29.74%), and PL displayed the best gain (31.36%).

Channel load standard deviation (SD) measures traffic distribution inside the network. Lower values correspond to a homogeneous traffic distribution, while higher values suggest that some channels have higher loads, and others are not used at all. For the simulated scenarios, BN reduced channel load SD by 20%, with values similar to the NN algorithm. Again, PL presented the best results, with a gain of 22% over the FF reference mapping.

Table 1 shows the average packet latency of the proposed heuristics. *Average packet latency* depends on the distance between source and target PEs and the communication path's congestion. The last line of the table shows each algorithm's percentage gain over FF mapping. In most cases, we obtained the best results with the PL heuristic. Additionally, PL, BN, and NN presented quite similar packet latency values, resulting in similar gains ($\approx 15\%$ with respect to FF).

Table 1. Average packet latency (in clock cycles). The lowest latencies are shaded.

Scenario (injection rate)	Reference mappings		Proposed heuristics			
	FF	NN	MMCL	MACL	PL	BN
A—pipe (5%)	164	141	152	156	142	147
A—pipe (10%)	269	239	255	258	239	238
A—pipe (15%)	371	334	350	356	328	330
A—pipe (20%)	527	431	479	478	425	450
A—pipe (25%)	656	542	645	579	531	531
A—pipe (30%)	815	665	781	711	673	664
B—tree (5%)	156	143	147	153	146	146
B—tree (10%)	271	247	255	246	233	242
B—tree (15%)	370	327	349	352	329	330
B—tree (20%)	514	433	447	493	430	434
C—generic	343	286	319	313	285	291
Gain with respect to FF (%)	—	14.98	6.15	8.07	15.59	14.66

Table 2. Summary of results for all mapping algorithms normalized to first free (FF) results. The best values are shaded.

Performance parameter	Reference mapping		Proposed heuristic			
	FF	NN	MMCL	MACL	PL	BN
Mapping complexity (x is 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 (SD)	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 (SD)	1.00	0.66	1.33	0.89	0.98	1.19
Execution time (vol.)	1.00	1.00	1.25	1.14	1.09	1.03
Execution time ($10 \times$ vol.)	1.00	0.98	1.00	1.00	0.99	0.99

Packet latency standard deviation is an important metric for systems with quality-of-service constraints. Packets that originate from a given source must keep a regular interval between them to respect the injection rate. Latency variation incurs jitter, which can lead to packet loss in applications with strict temporal deadlines such as real-time applications. In scenario C, all proposed heuristics improved latency SD (that is, reduced deviation). However, MMCL and MACL presented 10% and 16% gains respectively, while BN presented a 14% gain over FF. NN and PL reduced packet latency SD by around 21%.

The last aspect we evaluated is *execution time overhead*. This parameter's goal is to measure the impact of implementing the mapping heuristics on overall

execution time. The complexity of MMCL and MACL imposed a large penalty (25% and 13% respectively). PL and BN presented execution time overheads of 8.8% and 2.5% respectively. We considered this an acceptable cost, since these heuristics reduced channel load. Varying parameters in the performed experiments showed that a tenfold increase in communication volume can cancel this overhead. In that case, BN executed in 0.8% less time than FF, and PL in 1.13% less time than FF.

Table 2 summarizes the experimental results, normalized to the FF reference mapping. The PL heuristic achieved the best results in most performance figures: 31% smaller channel usage and 22% smaller packet latency. NN executed faster, due to its smaller

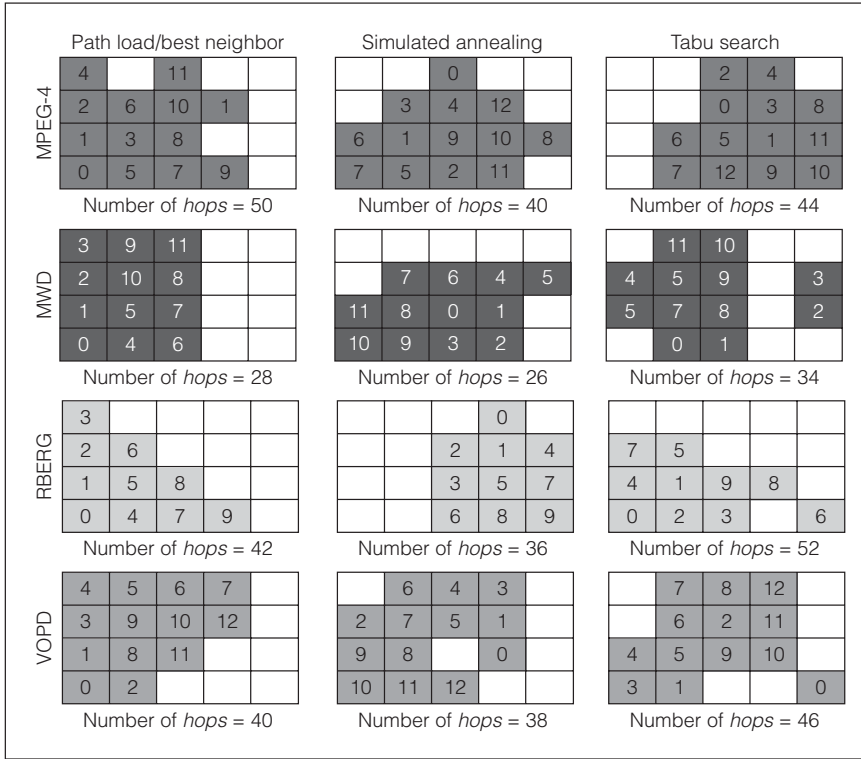


Figure 4. Mapping results for scenario D. Each square represents one PE. The number inside each square identifies the task allocated in that PE.

algorithmic complexity. Note that with increased transmission volume, mapping execution time did not affect total execution time.

Dynamic versus static mapping

Our second set of experiments compared both static and dynamic mapping alternatives, and we have enumerated the advantages and drawbacks of each. The experiments investigated two static mapping algorithms originally proposed by Marcon et al.,⁶ simulated annealing (SA) and tabu search (TS), and the two dynamic algorithms that presented the best results in the experiments we described previously, PL and BN.

The set of graphs used in these experiments consisted of four synthetic application graphs (obtained with the TGFF tool), containing from seven to nine tasks, and the following four real application graphs:

- MPEG-4 decoder—13 tasks;
- video object plane decoder (VOPD)—13 tasks;
- Romberg integration method (RBERG)—10 tasks;
- multiwindow display (MWD)—12 tasks.

A relevant feature of the MPEG-4 graph is that one of the 13 tasks is connected to eight other tasks. The VOPD application has less intertask dependency than the MPEG-4. The Romberg graph has significant intertask dependency, with most tasks communicating with four other tasks.

We evaluated two simulation scenarios:

- *Scenario D (single application mapping)*. This scenario employed a 5×4 homogeneous MPSoC. The evaluation goal was to compare mapping algorithms according to different performance figures for a single application. Thus, a static mapping could explore all MPSoC resources without the interference of other applications.
- *Scenario E (multiple application mapping)*. This scenario employed a 9×9 homogeneous MPSoC. The goal was to compare mapping algorithms for a set of simultaneously mapped applications.

Figure 4 presents the mapping of the real application graphs in scenario D for each algorithm. The figure shows the number of hops required to execute all bidirectional communication between all pairs of connected tasks for all mappings. The SA algorithm obtained the smallest number of hops for all applications, which meant fewer channels were used to execute all communications. BN and PL, which had a local view of the application, achieved results near SA's results and superior to TS's. The exception was the MPEG-4 application, in which task 1 had 8 neighbors. For BN and PL, task 1 was mapped near task 0 (its caller), and the remaining tasks connected to task 1 were mapped far from it because of a lack of available resources. Algorithms considering the whole application graph (SA and TS) mapped MPEG-4 task 1 in a more adequate position.

Table 3 presents the results obtained in scenario D, normalized to the SA algorithm. The channel

occupation performance parameters (average and SD) represent NoC usage. The cost overhead of the dynamic algorithms was low, approximately 6% for the average NoC occupation. The cost of the average packet transmission latency was even lower, 1%. The total number of hops increased 14% for the dynamic approaches.

Concerning total execution time, Table 3 shows that the values obtained with the BN and PL algorithms were on average only 4% worse than SA's result. Note that the time for mapping each task is considered when dynamic algorithms execute. Because static mapping algorithms execute at design time, there was no mapping overhead during application execution.

The last line of Table 3 presents average energy consumption, estimated according to Marcon et al.⁶ Dynamic mapping algorithms were penalized in this performance parameter by the additional network hops involved in the communication. MPEG-4 was the application that most penalized dynamic mapping algorithms. If this application were not included in the average, the energy consumption penalty would be smaller (17%). This comparison shows that dynamic mapping has smaller costs than static mapping when application graphs don't have strongly connected tasks. Situations like that of the MPEG-4 benchmark are not commonplace in actual applications.

Mapping quality can be estimated as a function of application area fragmentation. Figure 5 shows the application area distribution for the BN mapping. BN and PL generated more-isolated applications. The smaller fragmentation observed in dynamic mapping came from the clustering strategy, which increased the size of contiguous blocks.

Table 4 presents the results for scenario E, normalized as a function of the TS algorithm (for larger benchmarks, TS outperforms SA). The overall overhead imposed by dynamic mapping in such a complex scenario was small, as shown in columns PL and BN: 7% and 13% in average channel occupation, 7% and 10% in average packet latency, 4% and 3% in total execution time, and 18% and 13% in communication energy consumption.

MPSoCs CONTAINING several tens of processors have recently been launched as commercial products. To function effectively, these complex MPSoCs must incorporate distributed or centralized PE management.

Table 3. Summary results for scenario D. The values have been normalized with regard to the SA algorithm results.

Performance parameter	Dynamic		Static	
	PL	BN	SA	TS
Channel occupation (avg.)	1.07	1.06	1.00	0.96
Channel occupation (SD)	0.97	0.92	1.00	0.94
Packet latency (avg.)	1.01	1.01	1.00	1.01
Packet latency (SD)	0.99	0.99	1.00	1.00
Number of hops	1.14	1.14	1.00	1.26
Total execution time	1.04	1.03	1.00	1.01
Energy consumption	1.38	1.38	1.00	1.11

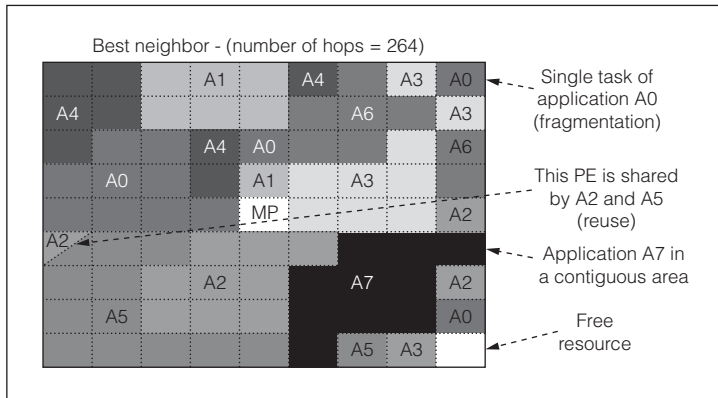


Figure 5. Application distribution for the BN algorithm in scenario E (A_n = applications; MP = manager processor). Each distinct shade of gray corresponds to a distinct application.

Table 4. Summary of results for scenario E, normalized with regard to the TS algorithm.

Performance parameter	Dynamic		Static	
	PL	BN	SA	TS
Channel occupation (avg.)	1.07	1.13	1.13	1.00
Channel occupation (SD)	1.18	1.27	1.36	1.00
Packet latency (avg.)	1.07	1.10	1.16	1.00
Packet latency (SD)	1.03	1.11	1.05	1.00
Number of hops	0.93	0.92	1.59	1.00
Total execution time	1.04	1.03	1.02	1.00
Energy consumption	1.18	1.13	1.35	1.00

Such management includes, for example, frequency and voltage control, processor and network load monitoring, and task mapping. The research described here contributes to the last item, task mapping.

According to our first set of experiments, the PL heuristic is the best solution. Compared to ad hoc mapping, the proposed algorithms allow up to 31% smaller channel load and up to 22% smaller packet latency. Additionally, the heuristics present an acceptable overhead cost. Local decisions taken by the algorithms make MMCL and MACL ineffective. When a new mapping does not reduce the average link load (or maximum load), these algorithms select the first available mapping option. NN and BN display some improvements but with gains inferior to PL.

Our second set of experiments shows that the cost overhead of dynamic mapping compared to static mapping for a complex scenario was on average 10% in channel occupation, 8.5% in latency, 3.5% in total execution time, and 15.5% in communication energy consumption. This is an acceptable overhead, considering the advantages offered by dynamic mapping:

- Smaller systems can be used, since only tasks being executed need be mapped into the system.
- The number of tasks can be greater than available system resources.
- The inclusion of new applications after design extends the MPSoC's usability.

Dynamic mapping's main weakness is that it takes a partial view of the application graph, considering only the intercommunication of the task being mapped and its caller task. On the other hand, static mapping considers all tasks and resources together, using algorithms more expensive in terms of execution time. In future research, we plan to take several directions to improve the performance of dynamic mapping. For example, when a given task starts its execution, the MP could start the mapping of all its slave tasks or reserve resources for them. A new heuristic could adopt a task migration strategy when communication cost becomes too high for a given task. Finally, we could increase the dynamic heuristic's view to two neighbor levels. ■

References

1. T. Lei and S. Kumar, "A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," *Proc. Euromicro Symp. Digital System Design (DSD 03)*, IEEE Press, 2003, pp. 180-187.
2. D. Wu, B. Al-Hashimi, and P. Eles, "Scheduling and Mapping of Conditional Task Graphs for the Synthesis of

- Low Power Embedded Systems," *Proc. Design, Automation and Test in Europe (DATE 03)*, IEEE CS Press, 2003, pp. 90-95.
3. S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," *Proc. Design, Automation and Test in Europe (DATE 04)*, IEEE CS Press, 2004, pp. 896-901.
4. S. Manolache, P. Eles, and Z. Peng, "Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC," *Proc. 42nd Annual Design Automation Conf. (DAC 05)*, ACM Press, 2005, pp. 266-269.
5. J. Hu and R. Marculescu, "Energy- and Performance-Aware Mapping for Regular NoC Architectures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, 2005, pp. 551-562.
6. C. Marcon et al., "Comparison of NoC Mapping Algorithms Targeting Low Energy Consumption," *IET Computers & Digital Techniques*, vol. 2, no. 6, 2008, pp. 471-482.
7. A. Ngouanga et al., "A Contextual Resources Use: A Proof of Concept through the APACHES Platform," *Proc. Design and Diagnostics of Electronic Circuits and Systems (DDECS 06)*, IEEE Press, 2006, pp. 42-47.
8. L. Smit, J. Hurink, and G. Smit, "Run-Time Mapping of Applications to a Heterogeneous SoC," *Proc. Int'l Symp. System-on-Chip (SoC 05)*, IEEE Press, 2005, pp. 78-81.
9. A. Mehran, A. Khademzadeh, and S. Saeidi, "DSM: A Heuristic Dynamic Spiral Mapping Algorithm for Network on Chip," *IEICE Electronics Express*, vol. 5, no. 13, 2008, pp. 464-471.
10. C.-L. Chou, U. Ogras, and R. Marculescu, "Energy- and Performance-Aware Incremental Mapping for Networks on Chip with Multiple Voltage Levels," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, 2008, pp. 1866-1879.
11. M. Al Faruque, R. Krist, and J. Henkel, "ADAM: Run-Time Agent-Based Distributed Application Mapping for On-Chip Communication," *Proc. 45th Annual Design Automation Conf. (DAC 08)*, ACM Press, 2008, pp. 760-765.
12. E. Carvalho and F. Moraes, "Congestion-Aware Task Mapping in Heterogeneous MPSoCs," *Proc. Int'l Symp. System-on-Chip (SoC 08)*, IEEE Press, 2008, pp. 1-4.

Ewerson Luiz de Souza Carvalho is currently a hardware designer at Datacom, Brazil. He worked on the research described in this article while a doctoral student at Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brazil. His research interests include MPSoC design and fast digital system

prototyping with FPGAs. Carvalho has a PhD in computer science from PUCRS, Brazil.

Ney Laert Vilar Calazans is a professor in the Faculty of Informatics at PUCRS. His research interests include MPSoCs, NoCs, GALS, and nonsynchronous systems. Calazans has a PhD in microelectronics from Université Catholique de Louvain, Belgium. He is a member of IEEE and the Brazilian Computer Society (SBC).

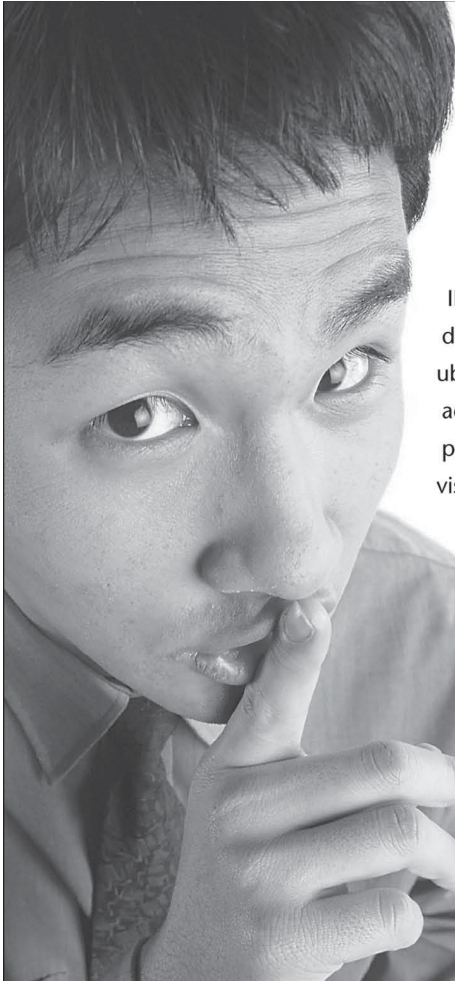
Fernando Gehm Moraes is a professor in the Faculty of Informatics at PUCRS. His research interests include NoCs, MPSoC design, and reconfigurable computing. Moraes has a PhD in microelectronics

from Laboratoire d'Informatique Robotique Microélectronique de Montpellier, France. He is a member of IEEE and the SBC.

■ Direct questions and comments about this article to Fernando Moraes, Faculty of Informatics, Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS), Hardware Design Support Group (GAPH), Av. Ipiranga, 6681, Building 32, Room 717, 90619-900, Porto Alegre, Brazil; fernando.moraes@pucrs.br.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.



Shhh. Pervasive Computing, **Pass it On...**

IEEE Pervasive Computing delivers the latest developments in pervasive, mobile, and ubiquitous computing. With content that's accessible and useful today, the quarterly publication acts as a catalyst for realizing the vision of pervasive (or ubiquitous) computing

Mark Weiser described nearly a decade ago — the creation of environments saturated with computing and wireless communication yet gracefully integrated with human users.

Editor in Chief: M. Satyanarayanan
Carnegie Mellon Univ. and Intel Research
Pittsburgh

Associate EICs: Roy Want, Intel Research; Tim Kindberg, HP Labs; Deborah Estrin, UCLA; Gregory Abowd, GeorgiaTech.; Nigel Davies, Lancaster University and Arizona University



UPCOMING ISSUES:

- ✓ Sensor and Actuator Networks
- ✓ Arts and Entertainment
- ✓ Handheld Computing



SUBSCRIBE NOW! <http://computer.org/pervasive/subscribe.htm>