Task Partitioning Optimization Algorithm for Energy Saving and Load Balance on NoC-based MPSoCs

Marco P. Stefani, Thais Webber, Ramon Fernandes, Rodrigo Cataldo, Letícia B. Poehls, César Marcon
PUCRS - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil – 90619-900
e-mail: marco.stefani@acad.pucrs.br, cesar.marcon@pucrs.br

Abstract

Multiprocessor System-on-Chip (MPSoC) based on Network-on-Chip (NoC) integrates a large amount of Processor Elements (PEs) to fulfill the performance requirements of several applications. These applications are composed of a set of intercommunicating tasks, which are dynamically mapped onto PEs of the target architecture. However, the efficient task-mapping requires some previous steps, among them partitioning, which organizes tasks considering their interaction before applying a mapping process. This paper introduces Partition Reduce (PR) - a task partitioning approach based on the MapReduce algorithm targeting homogeneous NoC based MPSoCs. We analyze the efficiency of PR for energy consumption (EC) minimization and load balance (LB). The results obtained from a set of experiments, with large number of tasks, demonstrate that PR is more effective on processing time and result quality when compared to the classic Simulated Annealing (SA). In addition, PR produces partitions with low energy consumption and rigorous load balance.

Keywords

Partitioning algorithm; homogeneous MPSoC; NoC; performance evaluation; energy efficiency; load balance.

1. Introduction

The International Technology Roadmap for Semiconductors (ITRS) [1] foresees that in the next decade a single silicon wafer will comprise several billions of transistors and hundreds of Processor Elements (PEs). This integration enables to implement complex applications into a single chip. However, it raises some research challenges, amongst them the design of efficient communication architectures and mechanisms able to map application tasks into PEs.

Network-on-Chip (NoC) is a communication structure composed of routers connecting other routers and PEs in 2D/3D silicon planes. NoC is a communication architecture that fulfills the parallelism and scalability required by Multiprocessors System-on-Chip (MPSoCs) containing several PEs [2]. A natural architectural evolution of 2D NoCs is three layered (3D) NoCs, since it reduces overall distances, minimizing latency and energy consumption while enhancing throughput and processing power with the larger number of PEs [3].

MPSoC architecture also introduces new challenges in application mapping design. Applications may be composed of a set of joint or disjoint tasks, each one performing instructions over a private or shared amount of data. These characteristics, combined with the distributed nature of PEs in NoCs, require techniques to map application and data dependencies over PEs, identifying situations where memory is shared between tasks, or where message exchanging becomes necessary. Grouping these tasks in an adequate manner is denominated *partitioning*.

Partitioning problems are NP-complete, which precludes exhaustive search for solutions. Thus, algorithms with different exploratory characteristics mostly require heuristics such as Simulated Annealing (SA) [4]. Nevertheless, data input has been increasing; thereby reducing the amount of testing that heuristics can perform in a timely manner, consequently, reducing the quality of the result. Therefore, optimized MPSoCs implementation requires the development of efficient partitioning approaches. The application partitioning and task mapping on NoC-based architectures has been addressed by several researchers [5]-[9]. Their researches contain algorithms for tasks partitioning onto groups of tasks, task mapping on PEs as well as PEs mapping on tiles, all targeting NoC-based architectures.

MapReduce [10] is widely used in distributed applications and it refers to the execution of two distinct tasks. The first task is mapping, which takes a set of data and converts it into a new representation, where individual elements are split into tuples (key/value pairs). The second task is reduction, which takes the mapping output (tuples) and combines them into a smaller set of tuples. Our paper proposes the Partition Reduce (PR) algorithm that explores the principles of MapReduce in the context of task partitioning targeting NoC-based MPSoC. The paper compares SA with PR using energy consumption and load balance as cost functions.

The remaining of this paper is organized as follows. Section 2 presents the problem formulation of tasks partitioning. Section 3 details the PR algorithm, while Section 4 shows experimental setup and results. Finally, Section 5 contributes with conclusions and further discussions.

2. Problem Formulation

We assume partition as the activity of grouping elements of a set **A**, generating a new set **B**, where each element of **B** is a set composed of one or more elements of **A** (such that $|\mathbf{A}| \geqslant |\mathbf{B}|$). The partitioning follows a set of rules that define the cost function, which normally implies to fulfill some design requirement, e.g. energy minimization. An application can be described as a set of tasks and their interrelations. Therefore, the partitioning of an application generates a new description composed of groups of tasks. This paper exploits static partitioning to reduce mapping complexity on NoC-based MPSoCs containing homogeneous PEs.

The mapping process is defined as the activity that associates elements of the source group **A** to elements of the target group **B**. Each element of group **B** is associated with one or more elements of group **A**, but an element of **A** may not be associated with more than one element of **B**. We consider three types of mapping: (i) the mapping of tasks into PEs; (ii) the groups of tasks being mapped into PEs; and (iii) the mapping of PEs into the tiles of the target architecture.

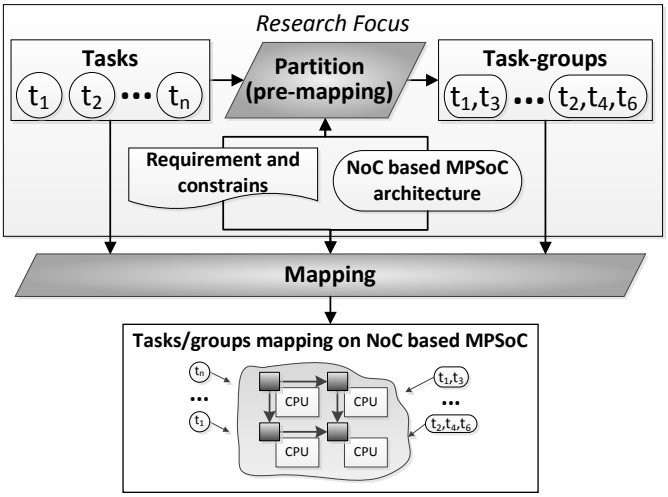

Figure 1: Application partitioning/mapping schemes.

Figure 1 illustrates the partitioning and mapping with application requirements and constraints (e.g. energy consumption, load balance) as well as MPSoC architecture (e.g. number of PEs, NoC topology).

3. Partition Reduce (PR) Algorithm

The proposed *Partition Reduce* (PR) algorithm is based on MapReduce [10], which has proven to be an efficient programming model for processing and generating large data sets. MapReduce is largely employed in distributed systems, since it deals with huge data sets such as distributed sort, inverted index construction, statistical machine translation, and machine learning, just to name a few.

A. Application Graphs Definition

Communication Weight Graph (CWG) models an application considering the communication amount occurring between each pair of tasks, which is given by the sum of all bits of all packets transmitted through the communication architecture. The CWG = <T, W> is a directed graph, where $T = \{t_1, t_2, \ldots, t_n\}$ is the set of application tasks and $W = \{(t_a, t_b, w_{ab}) \mid t_a, t_b \in T, w_{ab} \neq 0\}$ is the set of all bits transmitted from one task to another. CWG describes the application by tasks on vertices and the amount of communication between them being represented by edges.

The PR algorithm uses an internal graph IG that represents the application during the partitioning activity. Let $T_C = \{t_{C1}, t_{C2}, \ldots, t_{Cn}\}$ be the set of all application tasks characterized according to the target PE, such that $t_{Ci} = \{t_i, l_i, e_i, d_i, c_i \mid t_i \in T\}$, and l_i is the processing load needed for t_i operation. Then e_i, is the energy consumed by t_i running on the target PE, d_i is the t_i data size and c_i is the t_i code size, when the task t_i is compiled to the target PE. IG = {G, Z} is a tuple, where: (i) G set represents all groups of tasks. Initially, every task t_{Ci} is assigned to a single group g_i, and $|G| = |T_C|$. During PR execution, the tasks are grouped, therefore, reducing |G|; (ii) *Z* is a graph containing the grouping of tasks performed by PR. It has the same structure of CWG; however, each vertex of Z contains a group of tasks, and the edges are the set of communications between these groups.

B. Partitioning Algorithm

The MapReduce algorithm is a procedure of two distinct steps. Firstly, the mapping step: a set of elements are grouped in tuples generating a new set. Secondly, the reduce step: all generated tuples are processed to create a new and smaller set of groups of tasks. These two steps are performed repeatedly until the produced set fulfills the design constraints. The output of the reduce step is the input for the next mapping step. MapReduce reduces the amount of elements at each cycle due to its characteristic of grouping. PR algorithm employs similar approach, taking into account that the mapping step is actually a partitioning step with some differences following discussed.

Figure 2 describes the flow of PR partitioning with required inputs: (i) the application description (i.e. the CWG representing the application as a set of communicating tasks); (ii) the characterization of each task on the target PE (i.e. power consumption by task execution, code and data size, required processing load); (iii) the NoC topology with quantity of PEs; (iv) the constraints and the partitioning requirement.

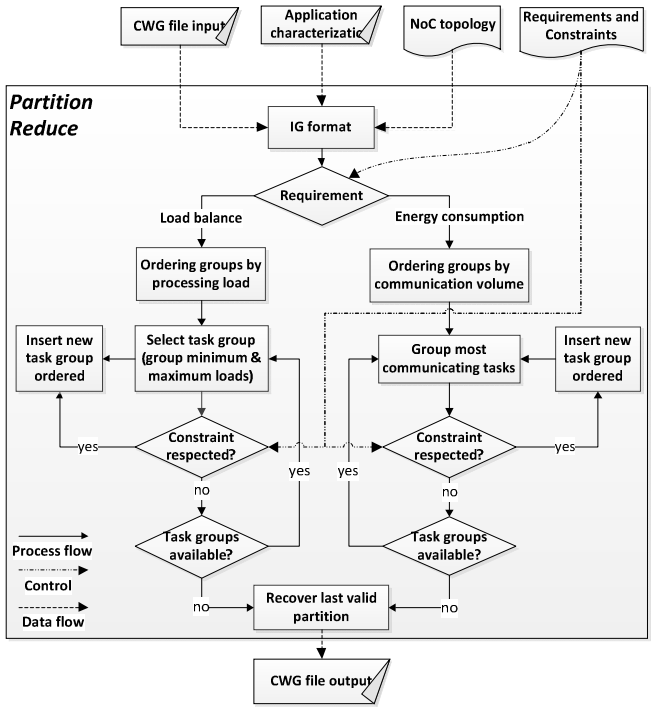

Figure 2: Activities flow of PR partitioning.

Differently from the original MapReduce algorithm, PR is not executed in a distributed manner. It uses the concept of search and generates task associations (i.e. map step) and thereby reduces the original set of tasks (i.e. reduce step). It is relevant to note that, statistically, every new cycle of IG computation tends to be faster than the previous one, since |G| is reduced when tasks are grouped.

Energy Consumption Requirement

The partitioning aiming reduced energy consumption sorts the group of tasks according to the energy spent on the communication between two groups of tasks.

The communication power consumption is carried out with the *EBit* model [11] to estimate the dynamic energy consumption for each bit.

$$EBit_{ij} = \eta \times ERbit + (\eta - 1) \times ELbit \quad (1)$$

Equation (1) describes the power consumption model, where *ERbit* is the energy dissipated at the router (router wires, buffers and logic gates); *ELbit* is the dynamic energy dissipation on links between tiles and considering regular mesh NoCs with square tiles to compute the dynamic energy consumed by a single bit traversing the NoC, from tile τ_i to tile τ_j and to tile τ_z, where η corresponds to the number of routers through which the bit passes. Let (x_i, y_i, z_i) and (x_j, y_j, z_j) be the coordinates of tiles τ_i, τ_j and τ_z, respectively, then $\eta = (|x_j - x_i| + |y_j - y_i| + |z_j - z_i| + 1)$ is the number of routers on a given communication.

The two most energy demanding groups are gathered together and processed by the reduce method, which checks if the constraints determined by the project are respected, thus the association is established; otherwise, the association is marked as processed and ignored. These two methods are executed repeatedly until only one group of tasks remains or there is no more group of tasks to be processed.

Load Balance Requirement

The partitioning aiming load balance sorts the group of tasks according to higher values of processing load. The most demanding group of tasks is gathered together with the least demanding group. The partitioning and reduce methods are executed repeatedly until only one group of tasks remains or there is no more group of tasks to be clustered.

C. Synthetic Example of PR Execution

Figure 3 illustrates a synthetic application composed of 12 tasks, where edges represent the number of bits transmitted from a task to another and the vertices contain the task identifier and the processing load (in percentage).

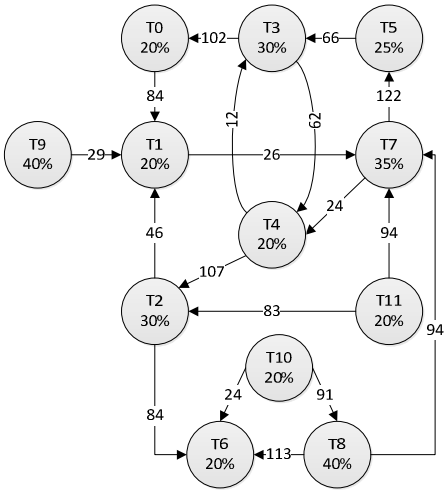

Figure 3: Example of a synthetic application representation.

The partitioning algorithm aims at the reduction of energy consumption, and considers 100% of processing load as the design constraint. Consequently, PR will produce groups of tasks with less than 100% of processing load aiming to reduce the total communication volume.

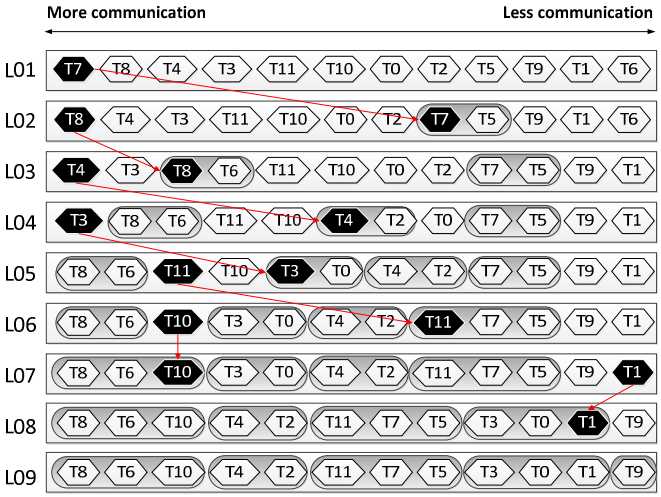

Figure 4: Flow of data at every step of the partitioning.

Figure 4 shows the output of each cycle executed in the partitioning. The tasks are chosen according to their communication volume. The most communicating demanding task is grouped with its receiving task in the communication tuple, and so on. If all of those groups respect the processing load constraint, the first cycle of the partitioning is done. Otherwise, the first and most demanding task is grouped with the receiving task, and so on. Note that, as the partitioning is repeatedly refined, the representation of tasks can change from a unique task (e.g. T6) to multiple tasks (e.g. T9, T6). Figure 5 illustrates the output according to partitioning process rules.

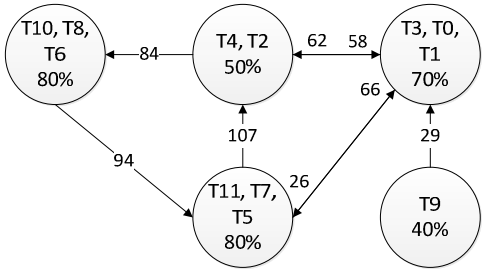

Figure 5: Partitioning graph generated by PR algorithm.

The partitioning output is a new CWG derived from the processing of input information, containing the partitioned application. The task group values are calculated in the partitioning through cost function minimization.

4. Experimental Results

This section presents the evaluation of the proposed PR algorithm compared with SA [4]. In the first scenario, the partitioning aims to balance processing load, while the second scenario aims to minimize energy consumption. The following criteria are used for comparison: (i) algorithm execution time (in seconds); (ii) amount of PEs required by partitioning; (iii) load balance through Mean Square Error (MSE); and (iv) energy consumption (in millijoule - mJ).

Experimental results consist of two task sets: the 'Equal Tasks' set assumes that all tasks are identical, i.e. with the same code and data size, consuming the same amount of energy when running on the target PE, and requiring 20% of the overall processing load. Tasks communicate with up to ten other tasks, and each communication has up to 500 bits of data. The 'Random Tasks' set creates random tasks, where the code has 50 to 500 bits, data size between 300 and 1000 bits, energy consumption between 10 and 100 phits, and processing load between 5 and 80%. A task communicates with up to 13 other tasks, and each communication has between 5 and 100 bits of data. All variables quantities in both sets are randomly generated by an in-house tool. Additionally, all groups of tasks are limited to 100% of processing load. The experiments contain from 8 up to 16,384 tasks. Results are shown in Table I and Table II, containing the evaluation of SA versus PR with respect to energy consumption and load balance.

Table I shows that energy consumption in both algorithms is similar for most cases. Initially, for 8 tasks, PR consumes approximately 28% more energy and the peak of difference is reached with 64 tasks, with approximately 31% of increase in energy consumption by PR in comparison with SA, for the same task set. However, this difference is reduced as the number of tasks increases, and from 128 tasks on, PR saves more energy than SA. These differences occur because PR searches by local minima, whereas SA tries n-steps searching for an optimal solution. PR readily converges to a good solution, but not to the best one, while SA loses quality as the number of tasks grows, consequently increasing execution time. Although energy saving is similar for most results, the execution times are expressively different.

Table I: Evaluation SA versus PR for energy consumption.

		Equal Tasks		Random Tasks	
#Tasks	Alg.	Time (sec)	EC (mJ)	Time (sec)	EC (mJ)
16,384	SA	n/a	n/a	n/a	n/a
	PR	262.76	199,436.2	296.40	559,741.4
8,192	SA	n/a	n/a	n/a	n/a
	PR	61,25	99760.2	68,47	281,283.9
4,096	SA	n/a	n/a	n/a	n/a
	PR	10.94	49,746.1	15.82	139,624.8
2,048	SA	n/a	n/a	n/a	n/a
	PR	2.44	24871.8	3.04	69,864.0
1,024	SA	6,132.32	12,931.0	7,879.4	38,441.1
	PR	0.70	12,406.5	0.62	34,226.8
512	SA	1259.5	6397.9	1731.33	18,925.0
	PR	0.22	6160.5	0.19	17,035.5
256	SA	307.07	3,132.6	429.38	9,539.5
	PR	0.16	3,038.2	0.08	8,587.9
128	SA	71.35	1470.0	112.47	4,598.4
	PR	0.09	1443.2	0.07	4240.2
64	SA	18.25	667.3	28.64	2,266.9
	PR	0.05	696.7	0.03	2,105.7
32	SA	5.04	268.0	8.02	975.5
	PR	0.03	311.3	0.02	958.8
16	SA	1.63	88.0	3.25	330.8
	PR	0.02	123.8	0.02	349.8
8	SA	1.04	19.1	1.74	89.6
	PR	0.01	26.8	0.01	99.9

The execution time of SA increases much faster than PR. When comparing both approaches, it is evident that SA becomes unfeasible for larger number of tasks, since for 1,024 tasks it already takes around two hours of processing time to achieve a result, making results for more tasks non available. In contrast, PR converges with less than a second for an experiment with 1,024 tasks, and only around four minutes for 16,384 tasks, which noticeably demonstrates the advantage of using PR for larger applications in such settings, as Figure 6 further illustrates.

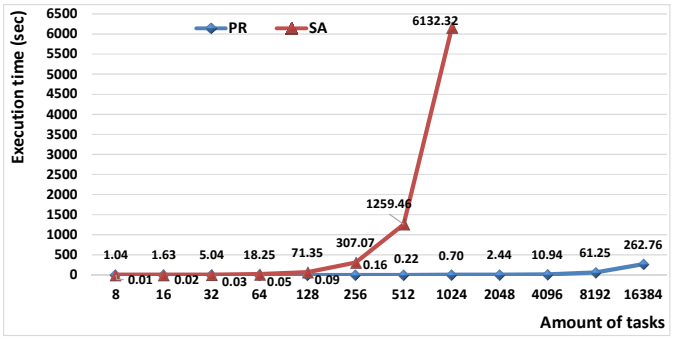

Figure 6: Comparison of execution time of SA versus RP taking into account energy consumption requirement.

According to Table II, the load balancing partitioning aims to achieve the least quantity of task groups with minimum MSE (Minimum Square Error). The partitioning considers 100% of processing load as the constraint to limit task grouping. Additionally, when the processing load is fairly distributed among the groups, MSE is minimized (e.g. MSE is zero when all groups have the same processing load).

Table II: Evaluation SA versus PR for load balance.

		Equal Tasks			Random Tasks		
#Tasks	Alg.	Time (Sec)	#PEs	LB (MSE)	Time (Sec)	#PEs	LB (MSE)
16,384	SA	471.29	4,048	541.8	1,108.57	11,898	537.1
	PR	12.63	3,277	0.1	16.60	7,723	5.5
8,192	SA	186.27	2,025	529.35	451.71	5953	531.2
	PR	2.65	1,639	2.2	3.76	3,889	5.8
4,096	SA	40.91	1,012	492.4	105.90	2,980	534.9
	PR	0.58	820	7.8	0.50	1,927	6.6
2,048	SA	16.83	504	447.5	32.27	1507	543.6
	PR	0.20	410	3.9	0.13	956	7.9
1,024	SA	8.03	253	375.4	15.05	741	513.0
	PR	0.16	205	1.9	0.05	481	19.5
512	SA	3.87	127	213.8	7.16	386	455.5
	PR	0.11	103	34.6	0.04	244	7.4
256	SA	2.41	64	62.5	3.79	191	428.7
	PR	0.08	52	120.7	0.03	120	9.9
128	SA	1.92	32	0.0	2.56	107	354.4
	PR	0.07	26	59.2	0.03	61	5.867
64	SA	1.52	16	0.0	2.00	51	294.0
	PR	0.02	12	28.4	0.02	30	71.7
32	SA	1.25	8	0.0	1.73	25	154.3
	PR	0.03	7	440.82	0.01	16	109.7
16	SA	1.15	4	0.0	1.4	12	215.3
	PR	0.01	4	1200.00	0.00	9	302.2
8	SA	1.07	2	0.0	2.15	6	108.5
	PR	0.01	2	400.0	0.01	4	462.0

As illustrated in Figure 7, the PR algorithm is inefficient to produce load-balanced partitions for small applications (i.e. applications with less than 64 tasks). This behavior occurs because PR locally optimizes the groups of tasks, and the remaining groups are usually less load balanced compared to the previous ones, since there are fewer tasks to group and therefore less diversity of processing load. However, when the

quantity of tasks increases, the global approach of SA cannot capture adequate partitions due to the increasing complexity; note that the partition problem grows exponentially according to the quantity of tasks.

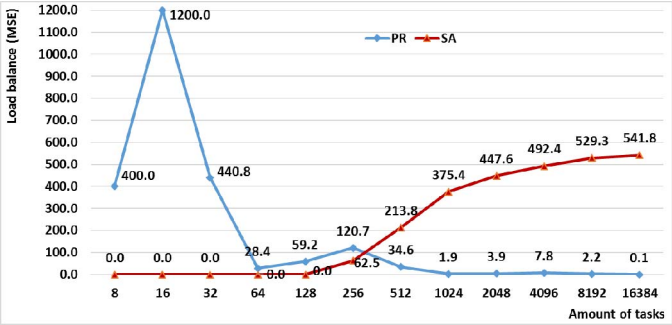

Figure 7: Comparison of load balancing requirements by the number of PEs for an equal set of tasks.

Figure 8 shows that random set of tasks presents a similar behavior than the one illustrated in Figure 7.

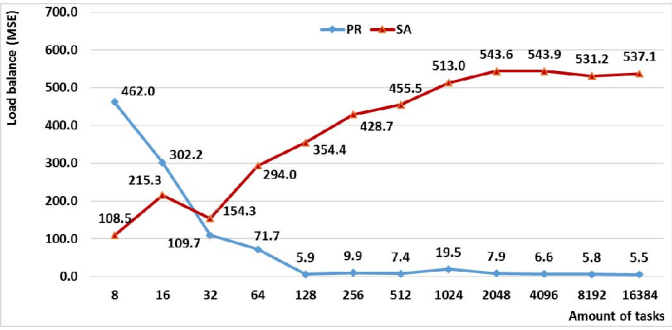

Figure 8: Comparison of load balancing requirements by the number of PEs for a random set of tasks.

One of the most important benefits of PR is the reduction of generated task groups, which may improve the quantity of target PEs in the target architecture. The SA algorithm presents an exponential evolution of execution time without making testing unfeasible. However, PR may be considered instantaneous in comparison with SA from any amount of tasks. Seeing the result in Table II, it is evident that PR produces less and better balanced groups with a very low cost of time while energy saving is maintained according to the target architecture.

5. Conclusion

This paper explores the effect on energy consumption and load balance when applying task partitioning for applications targeting NoC-based MPSoCs. Results show that task partitioning using load balance as the primary constraint demonstrates that SA algorithm performs very poorly compared to PR, which is more effective for large amounts of tasks. PR algorithm presents less complexity, brings satisfactory results in less time than SA, and has the additional advantage of reducing the number of PEs required for mapping. Furthermore, PR promises to be a great alternative compared to heuristic algorithms, since it is capable of finding satisfying results in an acceptable time, while including the extra advantage of being a deterministic algorithm.

References

[1] ITRS. www.itrs.net/Links/2011ITRS/2011SysDrivers.pdf, pp. 6-7, 2011.

[2] R. Patti. Impact of Wafer-Level 3D Stacking on the Yield of ICs. Future Lab Intl., issue 23, Jul. 2007.

[3] B. Feero, P. Pande. “Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation”, IEEE Transactions on Computers, v. 58, n. 1, pp. 32-45, Jan. 2009.

[4] H. Faragardi et al., “Reliability-Aware Task Allocation in Distributed Computing Systems using Hybrid Simulated Annealing and Tabu Search”, HPCC-ICESS, pp. 1088–1095, 2012.

[5] K. Siozios, I. Anagnostopoulos, D. Soudris. “A High-Level Mapping Algorithm Targeting 3D NoC Architectures with Multiple Vdd”, ISVLSI, pp. 444-445, 2010.

[6] R. Chen, H. Chen, B. Zang. “Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling”, PACT, pp. 523-534, 2010.

[7] Y. Tei, M. Marsono, N. Shaikh-Husin, Y. Hau. “Network partitioning and GA heuristic crossover for NoC application mapping”, ISCAS, pp. 1228-1231, 2013.

[8] C. Marcon et al. “Pre-mapping Algorithm for Heterogeneous MPSoCs”, VLSI Design, pp. 252-257, 2014.

[9] X. Zhang, B. Hu, J. Jiang. “An Optimized Algorithm for Reduce Task Scheduling”, Journal of Computers, v. 9, n. 4, pp. 794-801, Apr. 2014.

[10] J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, Communications of the ACM, v.51, n.1, pp. 107-113, Jan. 2008.

[11] C. Marcon et al. “Time and Energy Efficient Mapping of Embedded Applications onto NoCs”, Design Automation Conference, pp. 33-38, Jan. 2005.