Application Mapping onto 3D NoCs Using Differential Evolution

Maamar Bougherara[1], Nadia Nedjah[2], Djamel Bennouar[3], Rebiha Kemcha[1], and Luiza de Macedo Mourelle[4](✉)

[1] Département d'Informatique, Ecole Normale Supérieure de Kouba, Algiers, Algeria
bougherara.maamar@gmail.com, rebiha.kemcha@gmail.com

[2] Department of Electronics Engineering and Telecommunications, State University of Rio de Janeiro, Rio de Janeiro, Brazil
nadia@eng.uerj.br

[3] LIMPAF Laboratory, Bouira University, Bouíra, Algeria
djamal.bennouar@univ-bouira.dz

[4] Department of Systems Engineering and Computation, State University of Rio de Janeiro, Rio de Janeiro, Brazil
ldmm@eng.uerj.br

Abstract. Three-dimensional networks-on-chip appear as a new on-chip communication solution in many-core based systems. An application is implemented by a set of collaborative intellectual property blocks. The mapping of the pre-selected sets of these blocks on three-dimensional networks-on-chip is a NP-complete problem. In this work, we use Differential Evolution to deal with the blocks mapping problem in order to implement efficiently a given application on a three-dimensional network-on-chip. In this sense, Differential Evolution is extended to multi-objective optimization in order to minimize hardware area, execution time and power consumption of the final implementation .

Keywords: Three-dimensional networks-on-chip · Intellectual property mapping · Multi-objective optimization · Differential evolution

1 Introduction

A critical problem in the design of Multi-Processor Systems-on-Chip (MPSoCs) is the on-chip communication, where a Network-on-Chip (NoC) [15] can offer a scalable infrastructure to accelerate the design process. A MPSoC is designed to run a specific application, based on Intellectual Property (IP) blocks. A NoC consists of a set of *resources* (R) and *switches* (S), forming a *tile* [1]. Each resource of the NoC is an IP block, such as processor, memory, Digital Signal Processor (DSP), connected to one switch. Each switch of the NoC implements routing and arbitration, connected by *links*.

The way switches are connected defines the topology, such as the two-dimensional (2D) mesh topology [16]. However, the 2D NoC fails to meet the requirements of SoCs design in performance and area. Three-dimensional (3D) NoCs [2] have proved to be an effective solution to the problem of interconnection complexity in large scale SoCs by using integrated circuit stacking technology. A 3D mesh is implemented by stacking several layers of 2D mesh on top of each other and providing vertical links for interlayer communication, called Through Silicon Vias (TSV) [2]. Each switch is connected to up to six other neighbouring switches through channels, in the same way as 2D mesh does. Figure 1 shows the architecture of a 3D mesh-based NoC.

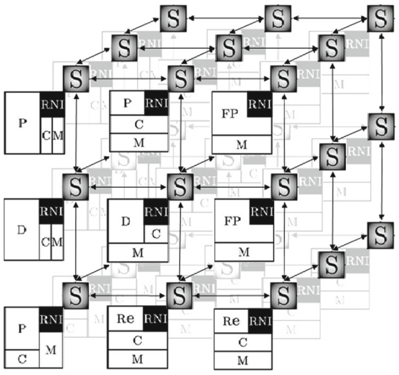

Fig. 1. 3D Mesh-based NoC with 27 resources

Different optimization criteria can be pursued depending on the detailed information of the application and IP blocks. The application is viewed as a graph of tasks called Task Graph (TG) [4]. The features of an IP block can be determined from its companion library [3]. The objectives involved in IP task assignment and IP mapping are multiple and have to be optimized simultaneously. Some of these objectives are conflicting because of their nature. So, IP assignment and IP mapping are classified as NP-hard problems [4]. Therefore, it is mandatory to use a multi-objetive optimization strategy, such as Multi-Objective Evolutionary Algorithms (MOEAs), with specific objective functions.

We use Differential Evolution (DE) as the MOEA [9], modified to suit the specificities of the assignment and mapping problems in a NoC with mesh topology, and also to guarantee the NoC designer's constraints. In previous work [12], we applied this strategy to the assignment problem in NoCs. In this paper, we describe the use of DE to the subsequent problem of mapping onto a 3D mesh-based NoC.

In Sect. 2, we present some related works, where a multi-objective strategy is applied in order to optimize some aspects of the design. In Sect. 3, we introduce the problems of IP assignment and mapping, concerning a SoC design over a NoC platform. In Sect. 4, we concentrate our attention on IPs mapping using DE for multi-objective optimization. In Sect. 5, we describe the objective functions for area, power consumption and execution time. In Sect. 6, we show some performance results, based on the E3S benchmarks suite. In Sect. 7, we draw some conclusions and future work, based on our experiments.

2 Related Work

In some works, the assignment and mapping steps are treated as a single NP-hard problem. The multi-objective nature of these steps is also not taken into account, addressing the problem as a single objective. Since we choose to treat the problem using a multi-objective optimization process, we present here some works that followed the same strategy when dealing with the mapping step.

In [5], the mapping step is treated as a two conflicting objective optimization problem, attempting to minimize the average number of hops and achieve a thermal balance. Every time data cross a switch, before reaching its target, the number of hops is incremented. To deal with this process, they used the multi-objective evolutionary algorithm NSGA.

The problem of mapping IPs/cores onto a mesh-based NoC is addressed by [6] in two systematic steps using NSGA-II. The key problem was to obtain a solution that minimizes energy consumption, considering both computation and communication activities, and also minimizes the link bandwidth requirements.

SPEA-II and NSGA-II are used in [8] for mapping, with some changes in crossover and mutation operators. Energy consumption and thermal balance were the main optimization objectives.

In [17], task mapping on a 3D mesh-based NoC is implemented using fuzzy logic in order to minimize thermal and power consumption.

In [18], the authors propose a multi-objective rank-based genetic algorithm for 3D mesh-based NoCs. Two different models are used for packet latency: under no congestion and with congestion situations.

In [19], a multi-objective immune algorithm is used, where latency and power consumption are considered as the objective functions, constrained by the heating function.

In [20], a centralized 3D mapping (C3Map) is proposed using a new octahedral traversal technology. Combining the C3Map and attractive/repulsive particle swarm optimization, they attempted to reduce energy and latency.

3 IP Assignment and Mapping Problems

The NoC design methodology for SoCs encourages the reuse of components to reduce costs and to reduce the time-to-market of new designs. The designer faces two main problems: selecting the adequate set of IPs (assignment step) and finding the best physical mapping of these IPs (mapping step) onto the NoC infrastructure.

The objective of IP assignment [4,10] is to select, from an IP library (IP repository), a set of IPs, exploiting re-usability and optimizing the implementation of a given application in terms of time, power and area requirements. During this step, no information about physical location of IPs onto the NoC is given. The optimization process must be done based on the Task Graph (TG) and IP features only. Each one of the nodes in the TG is associated with a task type, which corresponds to a processor instruction or a set of instructions. If a given processor is able to execute a given type of instruction, that processor is a candidate to be mapped onto a resource in the NoC structure and will be responsible for the execution of one or more tasks of the TG. The result of this step is a set of IPs that should maximize the NoC performance, *i.e.* minimize power consumption, hardware resources as well as the total execution time of the application. An Application Characterization Graph (ACG) is generated, based on the application's task graph, wherein each task has an IP associated with it.

We structured the used application repository, based on the E3S benchmark suite [21], using XML, both for the TG and the IP repository. Figure 2(a) shows the XML representation of a simple TG of ES3 and Fig. 2(b) shows a simplified XML representation of an IP repository. In previous work [12], we used DE during the assignment step in order to optimize area required, execution time and power consumption.

Given an application, the problem that we are concerned with here is to determine how to topologically map the selected IPs onto the network structure, such that the objectives of interest are optimized [4]. At this stage, a more accurate evaluation can be done taking into account the distance between resources and the number of switches and channels crossed by a data package during a communication session. The result of this process should be an optimal allocation of the prescribed IP assignment to execute the application on the NoC. Figure 3 shows the assignment and the mapping steps.

4 IPs Mapping Using Differential Evolution for Multi-objective Optimization

DE [11] is a simple and efficient Evolutionary Algorithm (EA). It was, initially, used to solve single-objective optimization problems [9]. DE is a population-based global optimization algorithm, starting with a population of NP individuals, of dimension D. Each individual encodes a candidate solution, *i.e* $X_{i,G} = \{X^1_{i,G}, ..., X^D_{i,G}\}$, $i = 1, ..., NP$, where G denotes the generation to which the population belongs [12]. The initial population is generated randomly from the entire search space. The main operations of the DE algorithm are: mutation, crossover and selection.

4.1 Mutation

This operation changes the population with the mutant vector $V_{i,G}$ for each individual $X_{i,G}$ in the population at generation G. The mutation operation can be generated using a specific strategy. In this work, three strategies are used: Rand (Eq. 1); Best (Eq. 2); Current-to-Best (Eq. 3):

$$V_{i,G} = X_{r_1,G} + F.(X_{r_2,G} - X_{r_3,G}), \tag{1}$$

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<task_graph>
  <tasks>
    <task code ="0" name="src"     type="45" />
    <task code ="1" name="text"    type="44" />
    <task code ="2" name="sink"    type="45" />
    <task code ="3" name="rotate" type="43"/>
    <task code ="4" name="dith"    type="42" />
  </tasks>
  <EDGES>
    <edge name="a0_0" from="0" to="1" cost="1000"/>
    <edge name="a0_1" from="0" to="3" cost="7070"/>
    <edge name="a0_2" from="3" to="4" cost="7070"/>
    <edge name="a0_3" from="4" to="2" cost="7070"/>
    <edge name="a0_4" from="1" to="2" cost="1000"/>
  </edges>
</task_graph>
```

(a) Example of a TG

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<repository>
  <ips>
    <ip procName="AMD_ElanSC520-133_MHz--square"
     price="33.0" taskTime="9e-06" taskPower="1.6"
     area="9.61E-6" taskName="Angle2Time Conversion"
     type="0" procID="0" id="0"
   />
   <ip procName="AMD_ElanSC520-133_MHz--square"
    price="33.0" taskTime="2.3e-05" taskPower="1.6"
    area="9.61E-6" taskName="Basic floating point"
    type="1" procID="0" id="1"
   />
   <ip procName="AMD_ElanSC520-133_MHz--square"
    price="33.0" taskTime="0.00049"taskPower="1.6"
    area="9.61E-6" taskName="Bit Manipulation"
    type="2" procID="0"id="2"
   />
    . . .
  </ips>
</repository>
```

(b) Example of an IP repository

Fig. 2. XML codes

$$V_{i,G} = X_{best,G} + F.(X_{r_1,G} - X_{r_2,G}), \tag{2}$$

$$V_{i,G} = X_{i,G} + F.(X_{best,G} - X_{r_1,G}) + F.(X_{r_2,G} - X_{r_3,G}), \tag{3}$$

where $V_{i,G}$ is the mutant vector to be produced; r_1, r_2, r_3 are integer constants generated randomly in the range of $[1, NP]$, at each iteration; $X_{best,G}$ is the best individual at generation G; F is a scaling factor, which is a real constant usually chosen in the range of $[0, 1]$, controlling the amplification of the difference variation.

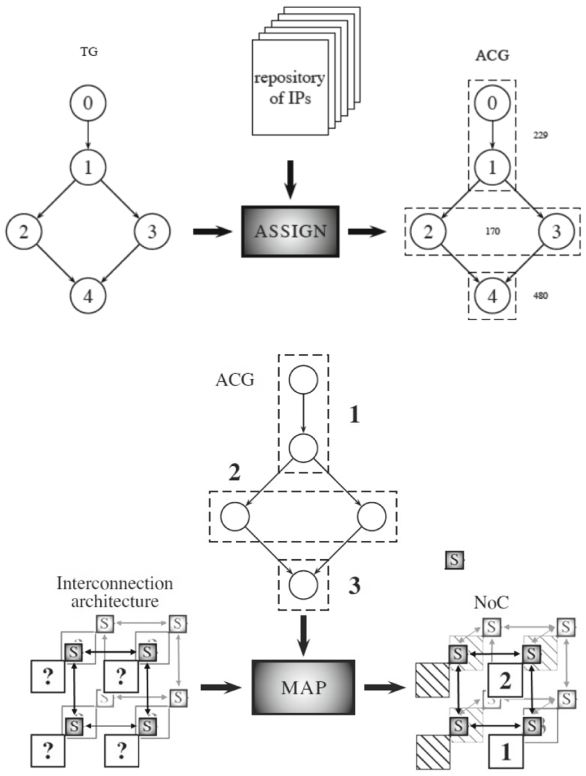

Fig. 3. IP assignment and mapping problems

4.2 Crossover

This operation improves the diversity of the population, being applied after the mutation operation. The crossover operation uses the mutation of the mutant vector $V_{i,G}$ to exchange its components with the target vector $X_{i,G}$, in order to form the trial vector $U_{i,G}$. The crossover operation is defined by Eq. 4:

$$U^j_{i,G} = \begin{cases} V^j_{i,G} & \text{if}(rand_j[0,1] \leq CR) or (j = j_{rand}) \\ X^j_{i,G} & \text{otherwise,} \end{cases} \tag{4}$$

where $j = 1, 2, ..., D$; $rand_j$ is the jth evaluation of an uniform random number generator within $[0, 1]$ [11]; the crossover rate CR is an user-specified constant within the range $[0, 1]$; j_{rand} is a randomly chosen integer within the range $[1, D]$ [9].

4.3 Selection

In order to keep the population size constant over subsequent generations, the selection operation is performed. The trial vector is evaluated according to the objective function and compared with its corresponding target vector in the current generation. If the trial vector is better than the target one, the trial vector will replace the target one, otherwise the target vector will remain in the population. The selection operation is represented by Eq. 5:

$$X_{i,G+1} = \begin{cases} U_{i,G} \ \text{if} f(U_{i,G}) \leq f(X_{i,G}) \\ \\ X_{i,G} \ \text{otherwise.} \end{cases} \tag{5}$$

The mutation, crossover and selection operations are applied for each generation until a termination criterion.

In order to extend the DE algorithm to solve multi-objective optimization problems, we should use the Pareto concept to deal with multiple objectives in order to select the best solution. If the newly generated trial vector dominates the parent vector, then the trial vector will replace the parent one. If the parent dominates the trial, the trial vector will be discarded. Otherwise, when the trial and the parent vectors are not related to each other, the trial vector will be added to the current population for later sorting. Algorithm 1 shows the main steps of the modified DE Multi-Objective (DEMO) algorithm.

Algorithm 1. Modified DEMO

```
initialize the individuals of the population
initialize best solutions in archive of leaders
iteration := 0
while iteration < max_iteration do
   for each individual do
      generate a mutated vector using a mutation operation
      generate a trial vector using crossover operation
      evaluate the trial vector
      if the trial vector dominates the individual then
         replace individual by the trial vector
      else if the individual dominates the trial vector then
         discard the trial vector
      else
         add trial vector to population
      end if
   end for
   update the leaders archive
   iteration := iteration + 1
end while
return result from the archive of leaders
```

5 Objective Functions

In this work, we adopted a multi-objective optimization strategy in order to minimize three parameters: area, power consumption and execution time. Here, we describe how to compute each of these parameters in terms of the characteristics of the application and those of the NoC.

5.1 Area

In order to compute the area required by a given mapping, it is necessary to know the area needed for the selected IPs and that required by the used links and switches. The total number of links and switches can be obtained by taking into account all the communication paths between the exploited tiles.

Each communication path between the tiles is stored in the routing table. We adopted an XYZ fixed routing strategy, in which data coming from tile i are sent first to the West or East of the current switch side depending on the target tile position, say j, with respect to i in the 3D mesh NoC, until it reaches the column of tile j. Then, it is sent to the South or North side, depending on the position of tile j with respect to tile i. Finally, it is sent to the Top or Bottom side until it reaches the target tile. The number of links in the described route represents the distance between tiles i and j, corresponding to the *Manhattan distance* [13] as defined by Eq. 6:

$$nLinks(i,j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|, \tag{6}$$

wherein (x_i, y_i, z_i) and (x_j, y_j, z_j) are the coordinates of tiles i and j, respectively.

In order to compute efficiently the area required by all used links and switches, the ACG is associated to a routing table, in which the links and switches necessary interconnect tiles are described. The number of hops between tiles, along a given path, leads to the number of traversal switches. The area is, then, computed summing up the areas required by processors, switches and links involved.

Equation 7 describes the computation involved to obtain the total area for the implementation of a given IP mapping M:

$$Area(M) = \sum_{p \in Proc(ACG_M)} area_p + (A_l + A_s) \times Links(ACG_M) + A_s, \tag{7}$$

wherein function $Proc(.)$ provides the set of distinct processors used in ACG$_M$ and $area_p$ is the required area for processor p; function $Links(.)$ gives the number of distinct links used in ACG$_M$; A_l is the area of any given link; and A_s is the area of any given switch.

5.2 Power Consumption

The total power consumption of an application NoC-based implementation consists of the power consumption of the processors, while processing the computation performed by each IP, and that due to the data transportation between the tiles, as presented in Eq. 8:

$$Power(M) = Power_p(M) + Power_c(M), \tag{8}$$

wherein $Power_p(M)$ and $Power_c(M)$ are the power consumption of processing and communication, respectively, as detailed in [13].

The power consumption due to processing is simply obtained summing up attribute *taskPower* of all nodes in the ACG and is as described in Eq. 9:

$$Power_p(M) = \sum_{t \in ACG_M} power_t. \tag{9}$$

The total power consumption of sending one bit of data from tile i to tile j can be calculated considering the number of switches and links each bit passes through on its way along the path. It can be estimated as shown in Eq. 10:

$$E_{bit}^{i,j} = nLinks(i,j) \times E_{L_{bit}} + (nLinks(i,j) + 1) \times E_{S_{bit}}, \tag{10}$$

wherein $E_{S_{bit}}$ and $E_{L_{bit}}$ represent the energy consumed by the switch and link tying the two neighboring tiles, respectively [22]. Function $nLinks(.)$, defined by Eq. 6, provides the number of traversed links (and switches too) considering the routing strategy used in this work and described earlier in this section.

The communication volume ($V(d_{t,t'})$) is provided by the TG in terms of number of bits sent from the task t to task t' passing through a direct arc $d_{t,t'}$. Let us assume that the tasks t and t' have been mapped onto tiles i and j respectively. Equation 11 defines the total network communication power consumption for a given mapping M:

$$Power_c(M) = \sum_{t \in ACG_M, \forall t' \in Targets_t} V(d_{t,t'}) \times E_{bit},^{Tile_t, Tile_{t'}}, \tag{11}$$

wherin $Targets_t$ provides all tasks that have a direct dependency on data resulted from task t and $Tile_t$ yields the tile number into which task t is mapped.

5.3 Execution Time

The execution time for a given mapping takes into account the execution time of each task, its schedule and the additional time due to data transportation through links and switches along the communication path. *taskTime* attribute in TG provides the execution time of each task. Each task of the TG needs to be scheduled into its own cycle. Therefore, we used the As-Soon-As-Possible (ASAP) scheduling strategy [10], scheduling tasks in the earliest possible control step.

The routing table allows us to count the number of links and switches required. Two scenarios can lead to the increase in the execution time of the application: (1) when a task in a tile needs to send data to parallel tasks in different tiles through the same initial link, data cannot be sent to both tiles at the same time; (2) when several tasks need to send data to a shared target task, one or more shared links would be needed simultaneously along the partially shared path, which means that the data from both tasks must be pipelined and will not arrive to the target task at the same time.

The overall application execution time takes into account the execution time regarding the underlying task computation and communication for the applied mapping M. It is also necessary to take into account the delay concerning the two aforementioned situations, regarding task scheduling. Therefore, the overall application execution time can be modelled according to Eq. 12:

$$Time(M) = Time_p + Time_c + t_l \times (F_1(M) + F_2(M)), \tag{12}$$

wherein $Time_p$ returns the time necessary to execute the tasks of the TG; $Time_c$ the time spent due to communication among tasks; function F_1 computes the delay caused by the first scenario; function F_2 computes the delay caused the second scenario.

Function F_1 computes the additional time due to parallel tasks that have data dependencies on tasks mapped in the same source tile and yet these share a common initial link in the communication path. Function F_2 computes the additional time due to the fact that parallel tasks producing data for the same target task need to use simultaneously at least a common link along the communication path. Equation 13 defines the time spent with communication between tasks t and t', based on [23]:

$$Time_c = \sum_{t \in APG_M, \forall t' \in Targets_t} \left\lceil \frac{V(d_{t,t'})}{phit} \right\rceil * T^{t,}_{phit}, \tag{13}$$

wherein $V(d_{t,t'})$ is the volume of bits transmitted from task t to task t'. Equation 14 defines the time spent transferring a *phit*:

$$T^{t,t'}_{phit} = nLinks(t,t') \times T_{l_{phit}} + (nLinks(t,t') + 1) \times T_{p_{phit}}, \tag{14}$$

wherein $T_{l_{phit}}$ is the link transmission time and $T_{p_{phit}}$ is the switch processing time used to transfer one phit between two neighboring tiles. A *phit* represents the physical unit given by the channel width and a *flit* represents the flow unit, which is a multiple of the *phit*.

6 Results

In order to evaluate the performance of the DEMO algorithm for the mapping step and compare it to that obtained using MOPSO (Multi-Objective Particle Swarm Optimization) algorithm [13], we used the same benchmarks. These are provided by the E3S benchmarks suite, constituted of the characteristics of 17 embedded processors. These characteristics are based on execution times of 16 different tasks, power consumption based on data-sheets, area required based on die size, price and clock frequency. We use 5 random task graphs used in [13], generated by Task Graph For Free (TGFF) [14] to perform experiments and evaluate the performance.

We exploit a population size of 100, with F set to 0.5 and CR set to 0.9. These parameters were set based on simulations. The algorithm was run for

500 iterations. It is noteworthy to emphasize that the same objective functions are used in both works. Besides comparing the algorithms, two topologies are used in the test: 2D mesh with 5×5 and 3D mesh with $3 \times 3 \times 3$. Also, we used As-Soon-As-Possible (ASAP) as the schedulling strategy.

Figure 4 shows the power consumption for the mapping yield by the compared strategies, regarding best results. We can see that the three mutation variants adopted for DEMO offer better results than MOPSO. Among these three, Best shows the best performance.

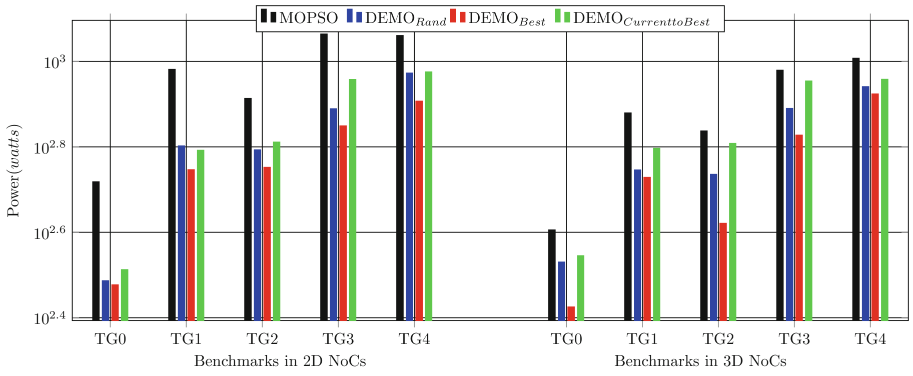

Fig. 4. Comparison of power consumption obtained for each mapping strategy

Also, Fig. 5 provides a comparison of the results regarding execution time, considering the best quality mapping. As can be seen, the performance of DEMO, based on Best mutation strategy, is better than those obtained by MOPSO and the other mutation strategies for DEMO.

Finally, Fig. 6 shows the comparison of the required hardware area related to the mapping obtained by the compared strategies. Here, also, the performance

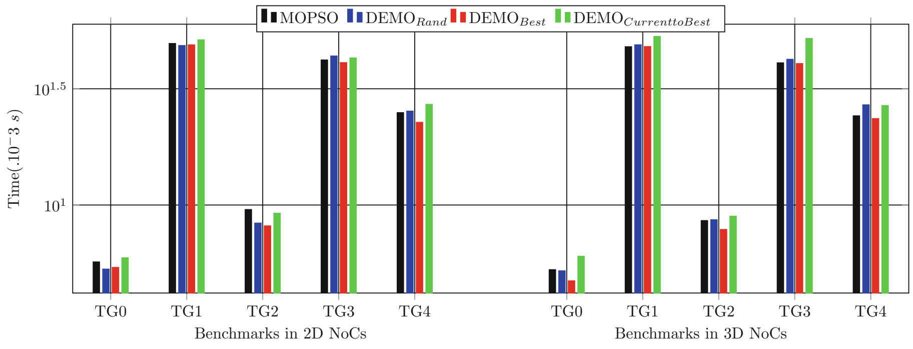

Fig. 5. Comparison of execution time obtained for each mapping strategy

of DEMO is better than that of MOPSO. Among the mutation strategies for DEMO, we can see that Best proves to be the best option.

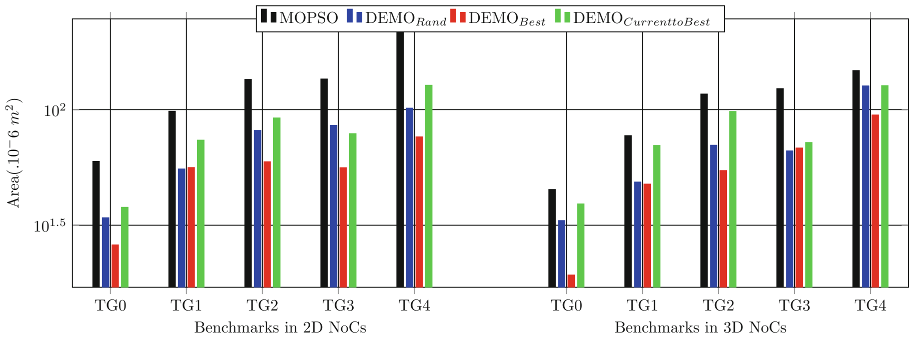

Fig. 6. Comparison of area requirements obtained for each mapping strategy

7 Conclusion

The problem of assigning and mapping IPs into a NoC platform is NP-hard. There are three objectives to be optimized: area required, execution time and power consumption. In this paper, we propose a Multi-Objective algorithm based on Differential Evolution (DEMO) to help NoC designers during the mapping step onto a 3D mesh-based NoC platform, based on pre-selected set of IPs.

We use the same objective functions and the same TGs used in [13], where we applied a Multi-Objective algorithm based on Particle Swarm Optimization (MOPSO), to evaluate the performance of the proposed algorithm. Besides this, the DEMO algorithm offers three strategies, related to variations of the mutation operation: Rand, Best, Current-to-Best. The strategy based on Best presents better performance than the other strategies. We also compare the results obtained by the DEMO algorithm to those obtained by MOPSO algorithm, where DEMO proves to be better than MOPSO. It is interesting to highlight that DEMO requires only two parameters to be set, while MOPSO requires three parameters.

For future work, we plan to experiment with other strategies in the DEMO algorithm and also to use other scheduling algorithms, such as List Schedulling and As Late as Possible.

References

1. Hu, J., Marculescu, R.: Energy-aware mapping for tile-based NoC architectures under performance constraints. In: Proceedings of the 2003 Asia and South Pacific Design Automation Conference, pp. 233–239. ACM, January 2003

2. Davis, W.R., Wilson, J., Mick, S., Xu, J., Hua, H., Mineo, C., Franzon, P.D.: Demystifying 3D ICs: The pros and cons of going vertical. IEEE Des. Test Comput. **22**(6), 498–510 (2005)
3. Ogras, U.Y., Hu, J., Marculescu, R.: Key research problems in NoC design: a holistic perspective. In: Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 69–74. ACM, September 2005
4. Da Silva, M.V.C., Nedjah, N., de Macedo Mourelle, L.: Evolutionary IP assignment for efficient NoC-based system design using multi-objective optimization. In: 2009 IEEE Congress on Evolutionary Computation, pp. 2257–2264. IEEE, MAy 2009
5. Zhou, W., Zhang, Y., Mao, Z.: Pareto based multiobjective mapping IP cores onto NoC architectures. In: APCCAS, pp. 331–334. IEEE (2006)
6. Jena, R.K., Sharma, G.K.: A multi-objective evolutionary algorithm based optimization model for network-on-chip synthesis. In: ITNG, pp. 977–982. IEEE Computer Society (2007)
7. Radu, C.: Optimized algorithms for network-on-chip application mapping ANoC, Ph.d. thesis, University of Sibiu, engineering Faculty Computer Engineering Department (2011)
8. Radu, C., Mahbub, M.S., Vintan, L.: Developing domain-knowledge evolutionary algorithms for network-on-chip application mapping. Microprocess. Microsyst. Embed. Hardware Des. **37**(1), 65–78 (2013)
9. Robič, T., Filipič, B.: DEMO: differential evolution for multiobjective optimization. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 520–533. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31880-4_36
10. Bougherara, M., Nedjah, N., de Mourelle, L., Rahmoun, R., Sadok, A., Bennouar, D.: IP assignment for efficient NoC-based system design using multi-objective particle swarm optimisation. Int. J. Bio Inspired Comput. **12**(4), 203–213 (2018)
11. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)
12. Bougherara, M., Kemcha, R, Nedjah, N., Bennouar, D., de Macedo Mourelle, L.: IP assignment optimization for an efficient noc-based system using multi-objective differential evolution. In: International Conference on Metaheuristics and Nature Inspired Computing (META) 2018, pp. 435–444 (2018)
13. Bougherara, M., Nedjah, N., Bennouar, D., Kemcha, R., de Macedo Mourelle, L.: Efficient application mapping onto three-dimensional network-on-chips using multi-objective particle swarm optimization. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11620, pp. 654–670. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24296-1_53
14. Dick, R.P., Rhodes, D.L., Wolf, W.: TGFF: task graphs for free. In: Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE 1998), pp. 97–101. IEEE, March 1998
15. Benini, L., De Micheli, G.: Networks on Chip - Technology and Tools. Morgan Kaufmann Publishers, San Francisco (2006)
16. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks - An Engineering Approach. Morgan Kaufmann Publishers, San Francisco (2003)
17. Mosayyebzadeh, A., Amiraski, M.M., Hessabi, S.: Thermal and power aware task mapping on 3d network on chip. Comput. Electr. Eng. **51**, 157–167 (2016)
18. Wang, J., Li, L., Pan, H., He, S., Zhang, R.: Latency-aware mapping for 3D NoC using rank-based multi-objective genetic algorithm. In: 2011 9th IEEE International Conference on ASIC, pp. 413–416. IEEE (2011)

19. Sepulveda, J., Gogniat, G., Pires, R., Chau, W., Strum, M.: An evolutive approach for designing thermal and performance-aware heterogeneous 3D-NoCs. In: 2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI), pp. 1–6. IEEE (2013)
20. Bhardwaj, K., Mane, P.S.: C3Map and ARPSO based mapping algorithms for energy-efficient regular 3-d noc architectures. In: Technical papers of 2014 International Symposium on VLSI Design, Automation and Test, pp. 1–4. IEEE (2014)
21. Dick, R.P.: Embedded system synthesis benchmarks suite (E3S). http://ziyang.eecs.umich.edu/~dickrp/e3s/. Accessed 17 Apr 2020
22. Hu, J., Marculescu, R.: Energy-aware mapping for tile-based NoC architectures under performance constraints. In: Proceedings of the 2003 Asia and South Pacific Design Automation Conference, pp. 233–239. ACM (2003)
23. Kreutz, M., Marcon, C.A., Carro, L., et al.: Design space exploration comparing homogeneous and heterogeneous network-on-chip architectures. In: Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design, pp. 190–195. ACM (2005)