Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2015, Article ID 902925, 13 pages
http://dx.doi.org/10.1155/2015/902925

Research Article

Using Genetic Algorithms for Hardware Core Placement and Mapping in NoC-Based Reconfigurable Systems

Jonas Gomes Filho, Marius Strum, and Wang Jiang Chau

Department of Electronics Systems, School of Engineering, University of Sao Paulo, Avenida Prof. Luciano Gualberto, Trav. 3, 05508-900 Sao Paulo, SP, Brazil

Correspondence should be addressed to Jonas Gomes Filho; jonasgfilho@lme.usp.br

Received 21 October 2014; Revised 24 December 2014; Accepted 26 December 2014

Academic Editor: Salvatore Pontarelli

Mapping of cores has been an important activity in NoC-based system design aimed to find the best topological location onto the NoC, such that the metrics of interest can be greatly optimized. In the last years, partial reconfigurable systems (PRSs) have included Networks-on-Chips (NoCs) as their communication structure, adding complexity to the problem of mapping. Several works have proposed specific and robust NoC architectures for PRSs, forming indirect and irregular networks, in which cases the mapping and placement problems must be treated altogether. The placement deals with the physical positioning of those cores inside the reconfigurable device. Up to now, to the best of our knowledge, the mapping-placement problem for those kinds of architectures has not been addressed yet. In this work, the problem formalization for the design-time hardware core placement and mapping in PRS-NoCs is proposed and methodologies for solving it with genetic algorithms (GAs) are presented. Several GA crossovers and methodologies are compared for obtaining the best solution. Results have shown that best GA solution obtained, in average, communication costs with 4% of penalty when compared with global minimum cost, obtained in a semiexhaustive approach. In addition, the algorithm presents low execution times.

1. Introduction

Run-time reconfiguration (RTR) FPGAs, also known as dynamically reconfigurable FPGAs (DRFPGAs), have been accepted as an important potential alternative for lowering costs of digital circuits, especially regarding the flexibility in rapidly changing the functions being performed and reducing area consumption. However, they add new dimensions to the system-on-chip (SoC) design space, due to the different possibilities of physical, temporal, and functional partitioning of the original application.

Considering the generalized use of robust communication resources in current complex SoCs, structured communication means, as network-on-chips (NoCs), have been included in partial reconfigurable systems, generating PRS based on Nocs, or PRS-NoCs, under different architectures [1–4]. Besides having the high scalability and modularity provided by NoCs, these PRSs can free the designer from the details of minimizing data retention and signals management, allowing him/her to focus on wrappers and computing logic, reducing the design effort.

The NoC-based PRS approach has not yet been largely adopted, partially due to the lack of more established design tools in the design cycle, as for process partitioning and mapping, or physical placement, which are recognized as hard problems, even for nonreconfigurable NoC-based systems [5]. Although methodologies and tools have been proposed [6–8], to deal with the increased design complexity of this class of circuits, solutions to the associated problems are still very ad hoc.

Mapping is the step in the SoC design flow where individual system processes are optimally assigned to NoC entry points (routers), through the reduction of traffic profile and consumed energy in the network, or other metrics defined by the designer. In a later point in the design flow, each process will be implemented either as a hardware core, a memory, or a programmable processor running a piece of software. Different mapping schemes have been presented for fixed, nonreconfigurable systems [5, 9], consisting basically of different algorithmic approaches for the resolution of different cost-function optimization problems. Mapping in MPSoCs environment has also been considered [10], but that

is treated as an operating system scheduling problem, with no regard to hardware cost criteria.

The reconfiguration capability adds a new dimension to the mapping problem since different cores are assigned to the same router but are present in the logic fabric in separate moments. There will exist several different configurations in time or contexts, each one showing a particular traffic profile. An optimized profile can be obtained for each context through the nonreconfigurable systems mapping methods, but the challenge is to consider all contexts in an interdependent form [11].

The placement problem deals with the allocation of resources (cores) inside the reconfigurable device; that is, given an assigned area, a set of cores must be placed in that area in a way that they do not overlap each other and do not exceed the space bounds. Traditionally, the placement problem is targeted to a regular NoC grid structure and performed after the core mapping. Some works have treated the placement as an execution time problem. One of the first works, made by Ahmadinia et al. [12], proposes an algorithm that finds the nearest possible position for an incoming core to the already placed cores. In [13], a complete real-time operating system was developed for tasks scheduling and placement in FPGAs. Although those approaches treat the placement problem, they cannot be applied to PRS-NoC architectures.

Some works on PRS-NoCs [1–4] have introduced advanced architectures for PRSs, in which physical area assigned to the routers is also reconfigurable. The router area may potentially be occupied by reconfigurable cores of varied sizes. The problem of placement and mapping for these architectures is extremely complex, actually, a combination of two NP-hard problems, with an explosive number of subcases to be treated. To the best of our knowledge, this problem has not been yet addressed, exception made for [14, 15]. In the first paper, the authors have treated the mapping-placement problem with focus in a regular an direct NoC architecture, where each node was composed of a reconfigurable slot, in which tasks can be allocated and placed as cores. In [15], a smart-exhaustive approach is presented for the mapping-placement problem for irregular and indirect reconfigurable NoCs; however, since the algorithm seeks the global minimum, it was not able to solve the problem for applications with more than 15 cores.

In this work, solutions based on genetic algorithms are presented for hardware core placement and/or mapping (in design-time) for PRS-NoC irregular and undirected NoC topologies and heterogeneous cores. The formalization of the problem is provided and the solution of the problem is obtained through genetic algorithms (GAs). For the best solution analysis, several GAs crossovers and population diversification techniques are applied and compared. The best GA solution is compared with the global minimum cost solution, obtained with the semiexhaustive algorithm described in [15]. Results have indicated GAs as a good alternative for solving the mapping-placement problem; the best GA solution has obtained, in average, communication costs with 4% of penalty when compared with global minima. Besides that, GAs applied to this problem have presented very low and adequate execution times, even for applications with 26 cores and 5 reconfigurable scenarios.

The rest of the paper is organized as follows: in Section 2 some related works are presented and a general overview of complex PRS-NoC architectures is shown in Section 3. The problem formulation is presented in Section 4 and the use of genetic algorithms for its solution is described in Section 5. Section 6 presents the experimental results and, finally, Section 7 shows the conclusions of this work.

2. Related Work

There are a few works about placement tools for PRS described in literature. One of the first works that considered the placement problem in SDRs [12] was for FPGAs with real-time reconfiguration, where the starting point is a scheme of scheduled and partitioned processes. The placement system is composed of three parts: a data base of modules, a placement request manager, and the placer. When a new module is up to operate, a request is made for the manager, and the algorithm analyses the occupied space to determine where to place the new module. The criterion of the placement is the optimization of communication between the modules. The work was improved in [16] with the inclusion of computational geometry techniques for empty spaces control.

In [7] a placement method for PRS-NoCs in FPGAs was proposed, considering heterogeneous processing elements (PEs) in design-time. A placement algorithm called Dyno-Place was proposed. It takes into account real aspects of FPGA families, making it possible to automate this process in the design methodology. The placement was organized in five steps: in the first one, the dimensions of PEs are processed and the modules are ordered; in the second one, a list scheduling algorithm is used for placing modules with same height sequentially in rows, from bottom to top and from left to right; the procedure places the larger modules first and the smaller ones later. The three last steps make the necessary adaptations for the DRFPGA specific architecture. This work does not treat the mapping problem and does not focus on NoC regularity, allocating PEs at one side of the device, while the routers are at the opposite side.

In [13] a reliable reconfigurable real-time operating system (R3TOS) was developed for reconfigurable systems without NoC. The R3TOS executes scheduling and placement of tasks using several different metrics and methods. The placement algorithms focus on preserving the maximum empty rectangle (MER) for the future use for as long as possible. The algorithm analyses the timeline, for preventing future fragmentations of modules, keeping the modules packed, and, consequently, achieving higher computation densities.

Although these works have made complex explorations in placement in PRS, they cannot be applied in NoC-based environments (with exception of [7]). With exception of [13], the placement algorithms only consider the instantaneous scenario, ignoring the general scenario with all configurations.

The NoC mapping problem has been widely explored in literature for nonreconfigurable architectures. Basically the associated methodologies and algorithms have focused

on minimizing different metrics, using different algorithmic solutions. The NMAP algorithm [9] minimizes the communication cost for single route multiple paths with an analytic approach; in [17], a study compared various algorithms for minimizing energy consumption: exhaustive search, simulated annealing, greedy incremental, and largest communication, among others. Several works also proposed a multiobjective approach like in [18], where an immune-adaptive algorithm was presented for reducing both power consumption and latency. A list of other nonreconfigurable NoC mapping works can be found in [5].

The work of Beretta et al. [14] proposes a placement and mapping methodology for simple PRS-NoCs, with a direct and regular NoC topology. The space of the FPGA is divided in big reconfigurable slots connected to routers, in which modules are placed. The placement and mapping are proposed for both application design and operating time and for multiapplications. In the preprocessing step, the solution is divided in basic mapping and specific configurations. The basic mapping is composed of the modules which are used by the most of the applications, and the specific configurations are a set of configurations that cannot be reused from the based mapping. Firstly, a genetic algorithm is used for placing modules of the base mapping. A second mapping is executed for placing the modules that belong to specific configurations. Other algorithms are proposed for real-time mapping: a greedy algorithm for configurations reuse or a SAT solver.

The only work to address the problem of mapping for the simple PRS-NoCs architectures is the one proposed for Beretta et al. However, it cannot be applied for the complex PRS-NoC architectures like the ones proposed in [1–4], which are the focus of the present work.

The only work to consider placement and mapping for complex PRS-NoC architectures was described in [15], where a semiexhaustive algorithm was proposed for solving the problem. The algorithm was tested on three synthetic applications, placing permutations of each combination of cores sequentially. As part of the algorithm, it prunes a permutation when it cannot be mapped due to extrapolation of the device space. The solution, however, could not treat applications with more than 15 cores due to the explosiveness of the problem.

3. Complex PRS-NoC Architectures

Complex PRS-NoC architectures are those with irregular and undirected NoC topology and heterogeneous cores, likes the ones proposed by Bobda et al. [1], Jovanovic et al. [3], and Killian et al. [4].

To characterize the complex PRS-NoC, the dynamic network-on-chip (DyNoC), proposed by Bobda et al. [1], is adopted as the base model. This architecture has been used due to its simplicity, and by using traditional NoC blocks, as the five-port router. The DyNoC is a mesh-based NoC topology where each router may be connected to a processing element (PE). The routers have five ports: east, west, north, south, and a local connection for a PE. The placement-mapping task is to allocate cores in the areas reserved to PEs.

An example of the architecture is illustrated in Figure 1, showing a mesh of routers (in orange) connected (or not) to local PEs, which can be of single (in blue) or large (other colors) sizes. In this example, four large size reconfigurable cores allocated in the network (C1–C4). When a core is placed, it occupies the specific area reserved to PEs; for large cores, the routers inside the occupied area are deactivated, and the communication internal to the cores is made with local buses (represented by dotted lines). Each core communicates to other NoC elements through an associated router located at the upper-right corner of the placement location. Whenever large cores exist, the NoC becomes indirect and irregular; not every router has one associated core, as, for instance, in Figure 1, the three ones at the bottom of the rightest column.

The communication is made through packets and the authors propose to use the surrounding XY (S-XY) routing algorithm, for the dynamic architecture. Basically, the algorithm detects when a placed module is on the routing way and surrounds it. The architecture provides an environment in which the packet always has a path from source to destination, since the placed cores are always surrounded by routers. This is guaranteed by positioning routers in all sides of the mesh, which makes the relative positioning between cores and routers inverted as in the left and bottom border in Figure 1. Having always a path between any two cores is specially interesting for PRSs, since the reconfigurable cores can be placed at any region.

4. Problem Formulation

Although we target DyNoC as the system model for the mapping and placement problem, in order to simplify the problem formulation and make it more general, it will be considered that all the routers are placed at the upper-right corner of the PEs. Hereinafter, the point of connection for each potential router in the mesh will be called allocation slot.

A reconfigurable device entry space, U, has $m \times n$ allocation slots. The set P indicates a set of positions in the device:

$$P = \{p_k \mid k = 1, 2, \ldots, m \times n\}, \quad p_k = \left(x_p, y_p\right). \tag{1}$$

A 7×7 entry space can be observed in Figure 2, where the lower-left corner has a position $p_1 = (0, 0)$, the upper-left corner has a position $p_{43} = (6, 0)$, and the upper-right corner has a position $p_{49} = (6, 6)$.

A scenario is a configuration defined in a given moment. Each reconfiguration is a different scenario. Each scenario s belongs to the set SC of all scenarios.

The classic problem formulations for NoC mapping are not adequate for describing the physical characteristics of the modules, which is necessary for solving the placement problem. Therefore, we propose for this situation an extended application characterization graph (EAPCG).

An EAPCG is given by $G(C, A)$, where each core $c_i \in C$ is a vertex with two weights (w_i, h_i), which represents the width and height of the core in terms of number of slots. The set C is subdivided in two groups: (1) fixed cores, present in all configuration scenarios, described as $\mathrm{Fc}_i \in \mathrm{FC}$, where FC is the set of all fixed cores; (2) reconfigurable cores present in

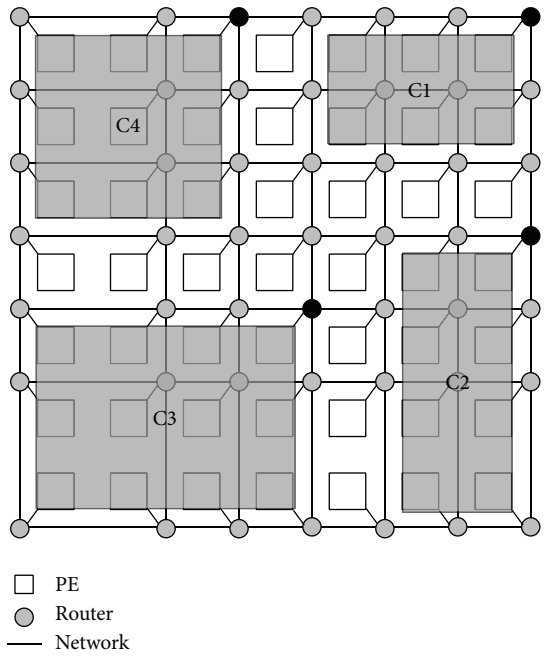

Figure 1: The DyNoC architecture [1].

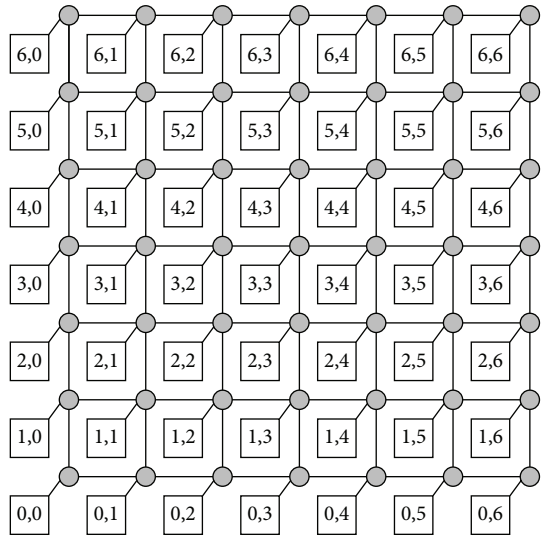

Figure 2: An entry space with 49 allocation slots (7 × 7).

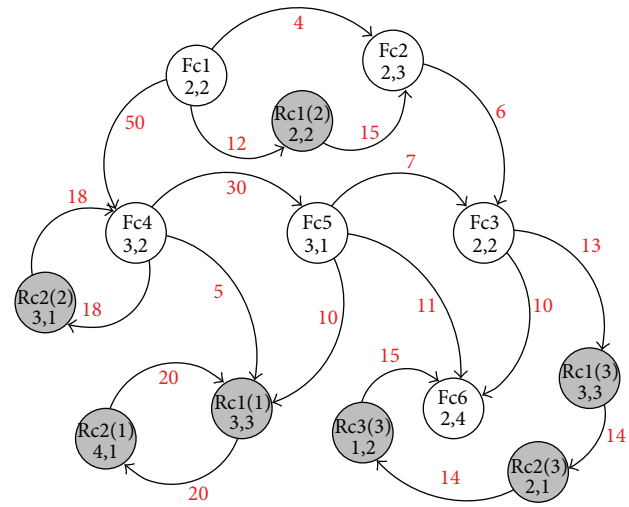

Figure 3: The EAPCG of the synthetic application Big Alpha.

only one scenario s, but missing in the next scenario $s + 1$, being described as $\mathrm{Rc}_i(s)$.

Each directed edge $a_{i,j} \in A$ represents the communication from c_i to c_j. For each edge, the parameter $b(a_{i,j})$ represents the communication bandwidth between c_i and c_j (in Mbits/s).

The $G(C, A)$ graph is illustrated in Figure 3 through the Big Alpha application, which will be used later in Section 6. This application has thirteen cores, being six fixed cores (Fc1–Fc6) and seven reconfigurable cores distributed in three scenarios (RC(1), RC(2), and RC(3)). The communication traffic between the cores which are represented in the arcs and the weights (width w_i, height h_i) is described in each vertex.

The mapping function in PRS-NoCs can be defined as

$$\Omega : C \longrightarrow P, \tag{2}$$

where each core c_i is associated to one reference position $p_k = (x_k, y_k)$, with $\Omega(c_i) = (x_i, y_i)$. To take into account the core sizes, the following function is also defined:

$$\phi : C \longrightarrow R, \tag{3}$$

where R is the set of all rectangular regions defined for two positions p_j and p_k, on opposite diagonals. $\phi(c_i) = r_i \in R$ is a rectangle with origin in $(x_i - w_i + 1, y_i - h_i + 1)$ and end in (x_i, y_i).

Each scenario s, composed of FC ∪ RC, has a specific mapping $\Omega\Omega(s)$, which is the union of all mappings $\phi(c_i)$ with $c_i \in \mathrm{FC} \cup \mathrm{RC}(s)$. The placement of the fixed modules remains unchanged for all scenarios s. Each mapping has the following restrictions:

$$\forall i \neq j, \quad \phi(c_i) \cap \phi(c_j) = \emptyset \tag{4}$$

$$\phi(c_i) \subseteq U. \tag{5}$$

Equation (4) guarantees that the cores will not overlap each other, whilst (5) assures that the placement will not exceed the device entry space.

A device architecture graph is a specific graph for each scenario s, $\mathrm{Arch}(\Omega\Omega(s)) = (T, \mathrm{Ch})$, where each router $t_i \in T$ is a vertex and each directed edge $\mathrm{ch}_{i,j} \in \mathrm{Ch}$ represents the communication channel between t_i and t_j. All routers are connected to each other in vertical and horizontal fashion in a $m \times n$ grid. For each edge, the parameter $\mathrm{bw}(\mathrm{ch}_{i,j})$ indicates the bandwidth capacity between t_i and t_j. $\mathrm{Path}(i, j)$ is the set of channels that make the path from the origin i to the destination j and $\mathrm{hops}(i, j)$ is the number of routers in which

the data must pass from i to j. Routers can be activated and deactivated, under the following definition:

$$\begin{aligned} &\text{for any router } t_i \text{ adjacent to } t_j, \\ &\text{bw}\left(\text{ch}_{i,j}\right) = 0 \qquad \text{bw}\left(\text{ch}_{j,i}\right) = 0. \end{aligned} \tag{6}$$

Equation (6) indicates that a deactivated router j does not have connections with any other router. If a router has at least one connection with another router, it is activated.

Derived from (2), the core mapping in each scenario s is given as

$$\Omega(s) : \text{FC} \cup \text{RC}(s) \longrightarrow P. \tag{7}$$

The existence of large size cores in the mapping process implies that the network topology may turns indirect. This issue will be modeled as follows.

Alongside the slot concept, the routers themselves are positioned in the space P. All the cores c_i communicate with other cores through an associated router in the same position of the placement:

$$\forall c_i \in C, \quad \exists t_i \in T \longrightarrow p(c_i) = p\left(t_j\right). \tag{8}$$

An occupied region $\text{Ro}(c_i) \subseteq R(c_i)$ is a region that do not allow active routers. This region exists for large core sizes if both height and width are greater than one and it is delimited by a rectangle with origin in $(x_i - w_i + 1, y_i - h_i + 1)$ and end in $(x_i - 1, y_i - 1)$. Figure 4 illustrates the concept of a region and an occupied region.

The routers inside an occupied region are deactivated whilst other routers are activated.

For each scenario s the communication between the modules cannot exceed the bandwidth capacity of the channels:

$$\forall \text{ch}_{i,j} \in \text{Arch}(\Omega(s)), \quad \text{bw}\left(\text{ch}_{i,j}\right) \geq \sum_{a_{i,j} \in \Omega(s)} Z_{i,j}, \tag{9}$$

where

$$Z_{i,j} = \begin{cases} b\left(a_{i,j}\right), & \text{if } \text{ch}_{i,j} \in \text{Path}\left(p(c_i), p\left(c_j\right)\right) \\ 0, & \text{otherwise,} \end{cases} \tag{10}$$

where $\text{Path}(p(c_i), p(c_j))$ indicates the sequence of the channels from c_i to c_j with the S-XY routing algorithm applied.

The mapping objective is to place the cores in order to minimize the communication cost in every configuration scenario, as described by

$$\text{comm\_cost} = \sum_{a_{i,j} \in \Omega(s)} b\left(a_{i,j}\right) \times \text{hops}\left(p(c_i), p\left(c_j\right)\right), \tag{11}$$

$$\min \sum_{s=1}^{N} \text{comm\_cost}. \tag{12}$$

The objective function is simple; however when the number of hops is minimized, several other metrics are optimized, for example, latency, communication delay, and energy consumption. This kind of objective function is widely used in literature, including [5, 9, 14, 17].

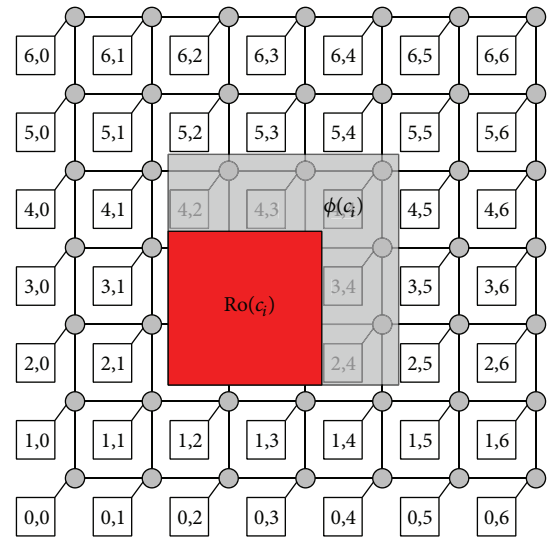

Figure 4: A region $R(c_i)$ and a occupied region $\text{Ro}(c_i)$.

The minimum cost problem is illustrated in Figure 5, for the Big Alpha application of Figure 3. The routers associated to the cores are the black ones. It may be noticed that cores with critical communication, like from Fc1 to Fc4, with 50 MB/s, are close to each other. This can also be observed when the core Rc2(2) is placed on the second scenario: it is only two hops away from Fc4, since they send and receive 18 MB/s to each other. Similar communication treatment for other pairs of cores may be observed in the mapping results. Here the costs for the first, second, and third scenario are, respectively, 454, 449, and 452, with a total cost of 1355.

5. Using Genetic Algorithms

In this section, we present the algorithmic solution to be applied in the design-time for a PRS-NoC, in order to obtain the optimized placement and mapping. For solving the problem, we have selected genetic algorithms (GAs). GAs have been successfully used as a solution for a large set of minimization problems, including mapping problems [5, 14], and they are able to deal with the hard constrains as the ones previously described. In addition GAs greatly accelerate the execution time when searching for a good solution, being adequate for the mapping and placement problem which is known to be explosive [5]. In this section, the use of GAs for solving the problem presented in Section 4 is presented in detail.

5.1. GA Coding. For the chromosome coding, we have developed a simplified representation for the placement and mapping process. Since dispersed cores require more routers to hold communication, we start on with the assumption that packed cores leads, generally, to a better communication cost. Therefore, we will not consider empty spaces between cores, what will be reflected in the chromosome, which makes the computation process simpler and prevents fragmentation. Using a generalist approach and considering the empty slots

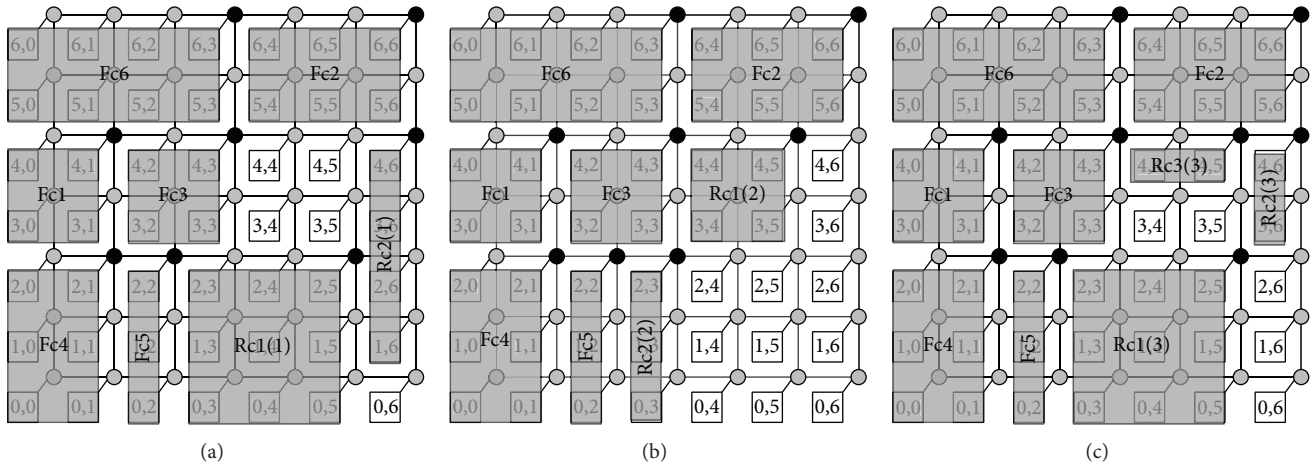

Figure 5: Mapping result for Big Alpha application: (a) scenario 1, (b) scenario 2, and (c) scenario 3.

would make the problem complexity proportional to the factorial of the number of slots in the architecture while the simpler representation makes the problem complexity proportional to the factorial of the number of modules.

Each core represents a gene and the set of cores, ordered according to the order of the scenarios, represents a chromosome.

Figure 6 shows the coding for an application with thirteen cores and three scenarios. The first eight elements of the chromosome represent the set of 5 fixed cores besides the 2 reconfigurable cores that compose reconfigurable scenario 1. The ninth and tenth elements represent the reconfigurable cores of scenario 2 and the last three elements represent the reconfigurable cores of the last scenario, remembering that it is implicit that the fixed cores appear in all three scenarios. The core position into the chromosome represents the physical position of the module, which can be seen in Figure 5. Cores are placed according to two priorities: (1) from top to bottom; (2) from left to right. The placement-mapping for the first scenario described in Figure 6 is shown in Figure 5(a). In the second scenario, reconfigurable cores of first scenario are unplaced and Rc2(2) and Rc3(2) are placed instead, using the top-bottom and left-right priority, as shown in Figure 5(b). Figure 5(c) shows the same process for the last three elements of the chromosome.

5.2. General Structure. According to the GA technique, each solution of the problem can be coded as chromosomes. A chromosome is a set of genes, and a set of chromosomes compose a population. A classical GA follows four main steps: (1) generation of a random population; (2) selection of the fittest individuals; (3) crossover of selected individuals; (4) mutation. Steps (2), (3), and (4) are repeated for the refinement of the new population (offspring), and the previous population is substituted (killed) [19].

The implemented algorithm is described as in Algorithm 1 as a pseudocode. The algorithm will be tested later with several types of crossover and with population diversification techniques; however, this general structure will be the reference.

Initially, a set of random solutions is generated (line 2). The "population size" (PZ) parameter defines the number of chromosomes in each population. The fitness function is defined as the inverse of communication cost, represented by (12).

The main algorithm is composed by two loops: the first one (starting at line 4) refers to the refinement of population which is limited by the number of generations (NG). In each generation, a second loop (starting at line 5) performs the selection, crossover, and mutation for all elements of the population.

The selection step is performed by the roulette wheel technique [19] in which the probability P of a chromosome being selected is proportional to its fitness value (13). In line 6 the roulette process is used to select chromosomes cr_sel1 and cr_sel2. Consider

$$P(n) = \frac{\text{fitness}(n)}{\sum_{m=1}^{\text{PZ}} \text{fitness}(m)}. \tag{13}$$

At the end of each iteration, the previous population is killed and the offspring becomes the new population. Line 27 saves the best solution of the current generation.

The crossover is executed with a probability P(Crossover), typically between 0.5 and 1 [20]. The crossover operator is applied in different parts of the chromosome which represents a scenario. The crossover between chromosomes cr_sel1 and cr_sel2 generates two new chromosomes, where cr_sel1 is the donor of genetic material for the first one and cr_sel2 is the donor for the second one.

After the crossover, the algorithm executes the mutation according to its probability (usually lower than 0.05 [20]). The mutation process is basically the swapping of two genes of each part of the chromosome. The mutation is useful to avoid focusing on a particular solution, maintaining a wider searching space. In line 19, the possibility of the mapping is verified, that is, if it is valid and fits into the entry space with respect to the restriction described in (5). If it does, fitness function is computed; otherwise fitness function value is zero. In line 26 offspring becomes the current population, while

Figure 6: Placement-mapping chromosome coding for GA.

```
(1) best_solution = ∞
(2) population = generate_random_population(PZ)
(3) ∀ population, fitness = 1/comm_cost
(4) while generation ≤ NG do
(5)    while  i ≤ PZ do
(6)       cr_sel1, cr_sel2 = roulette(population, fitness)
(7)       if  P(crossover) then
(8)          Offspring(i) = Crossover(cr_sel1, cr_sel2)
(9)          Offspring(i + 1) = Crossover(cr_sel2, cr_sel1)
(10)      else
(11)         Offspring(i) = cr_sel1
(12)         Offspring(i + 1) = cr_sel2
(13)      end if
(14)      if  P(mutation) then
(15)         Offspring(i) = mutate(cr_sel1)
(16)         Offspring(i + 1) = mutate(cr_sel2)
(17)      end if
(18)      for  i, i + 1  do
(19)         if offspring(i).maps then
(20)            fitness = 1/comm_cost
(21)         else
(22)            fitness = 0
(23)         end if
(24)      end for
(25)   end while
(26)   population = offspring
(27)   if fittest(generation) < best_solution then
(28)      best_solution = fittest(generation)
(29)   end if
(30)   generation++
(31) end while
(32) return best_solution
```

Algorithm 1: Pseudocode of a GA for reconfigurable placement and mapping in NoCs.

previous population is killed. If a better solution is found, it is saved at the end of iteration.

5.3. Crossovers. Normally, crossover operators are applied all over the chromosome. However, the representation adopted in this work, as in Figure 6, subdivides the chromosome in parts corresponding to each configuration scenario; an adaptation was performed: the crossover is executed in several parts of the same chromosome. Three types of crossovers are presented.

5.3.1. OX. The first crossover is the classic ordered crossover [19]. Figure 7 illustrates an example of the OX process in a mapping with three scenarios, where the first chromosome is the donor and the second one is the receiver. Figure 7(a) shows the first step, where two chromosomes are selected: the first chromosome will donate a subset of each part (represented in gray) to the receiver in the same position. In the second step, the receiver eliminates from its codes the cores donated by the donor, as shown in Figure 7(b). The remaining cores in the receiver are circularly shifted to the left inside its own area (e.g., the first part performs shifts only in the eight first slots) as illustrated from Figure 7(c) to Figure 7(d), occupying empty spaces and opening the required slots for the donor subset. Finally Figure 7(d) shows the OX result.

5.3.2. NWOX. The OX presents a shifting characteristic that may not preserve the absolute positions of the parents in the chromosome of the son: the circular shift moves the genes from extreme left to the extreme right. For preserving the absolute positions in the crossover, the nonwrapping ordered crossover (NWOX) [21] was proposed.

Figure 8 shows the NWOX crossover process. Figures 8(a) and 8(b) present the same situation seen in the OX case;

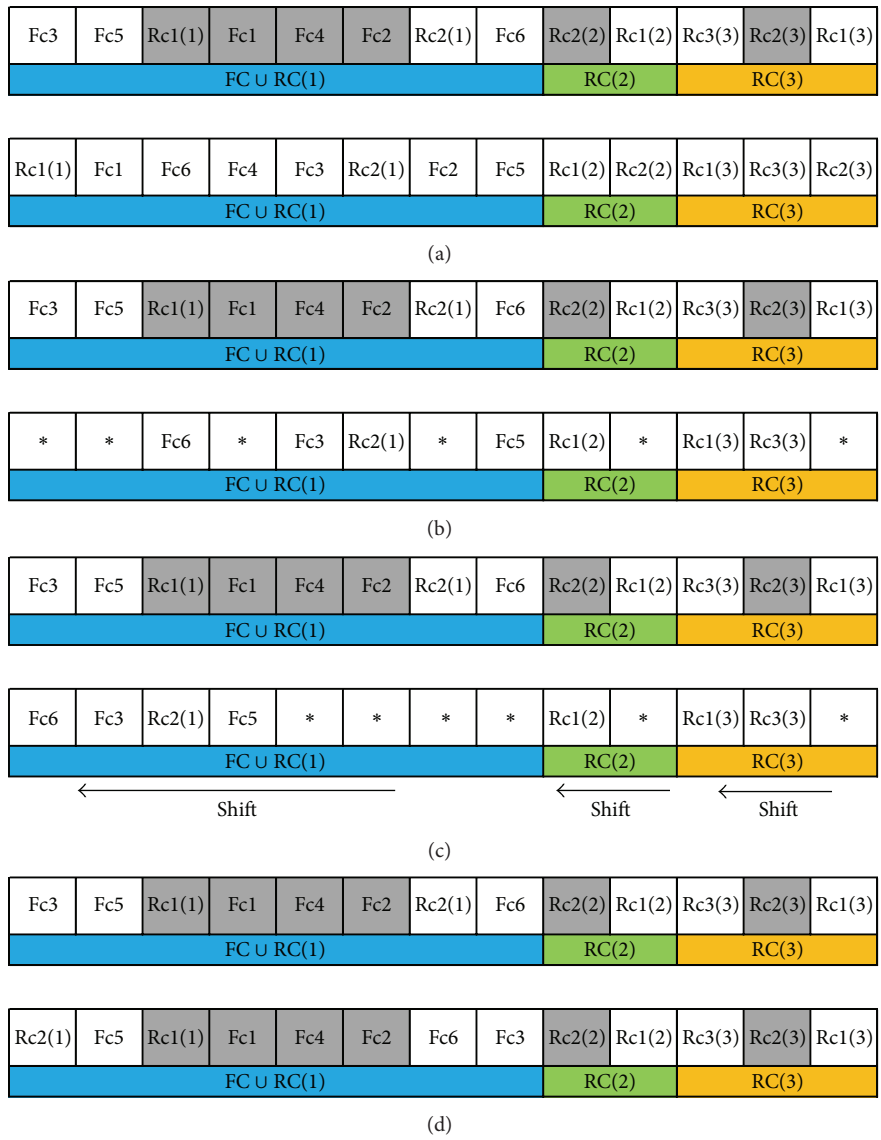

Figure 7: Ordered crossover (OX).

as in the second figure, the order of Fc6, Fc3, RC2(1), and Fc5 is established. The difference turns out to be clear in the third step, where the space for reception of new cores is allocated between the ordered genes as in Figure 8(c).

Note that the relative order of the receptor is preserved. Instead of performing the circular shift and moving the modules Fc6 and Fc3 to the right side of the chromosome as in the OX, those are maintained closer to their original positions. In this way, the final result of the NWOX is shown in Figure 8(d).

5.3.3. PMX. The partially matched crossover (PMX) [19] uses a principle of position swapping inside the chromosome.

Figure 9(a) presents the donor and receptor (superior and inferior, resp.). In Figure 9(b) the first swapping process occurs, where the position of the receptor gene swaps positions so that it can correspond to the same positions of the donor in the gray area. As the gene Fc4 was swapped to a region outside of the gray zone, its position must be changed again with a new swap, until the gray region turns identical to the gray zone of the donor as illustrated in Figure 9(c).

5.4. Population Diversification Techniques. One of the known problems of GAs is the premature convergence to a local minimum [22]. To deal with this issue, several methodologies were proposed to force the diversification of population, maintaining its health. Two techniques were selected among the existing ones, due to their known effectiveness and easy implementation: the random offspring generation (ROG) [23] and the adaptative genetic algorithm (AGA) [20].

5.4.1. ROG. One of the factors that lead to a premature convergence is the great number of individuals with the same genetic material. In case that happens, there is a great probability of performing crossover with two identical parents,

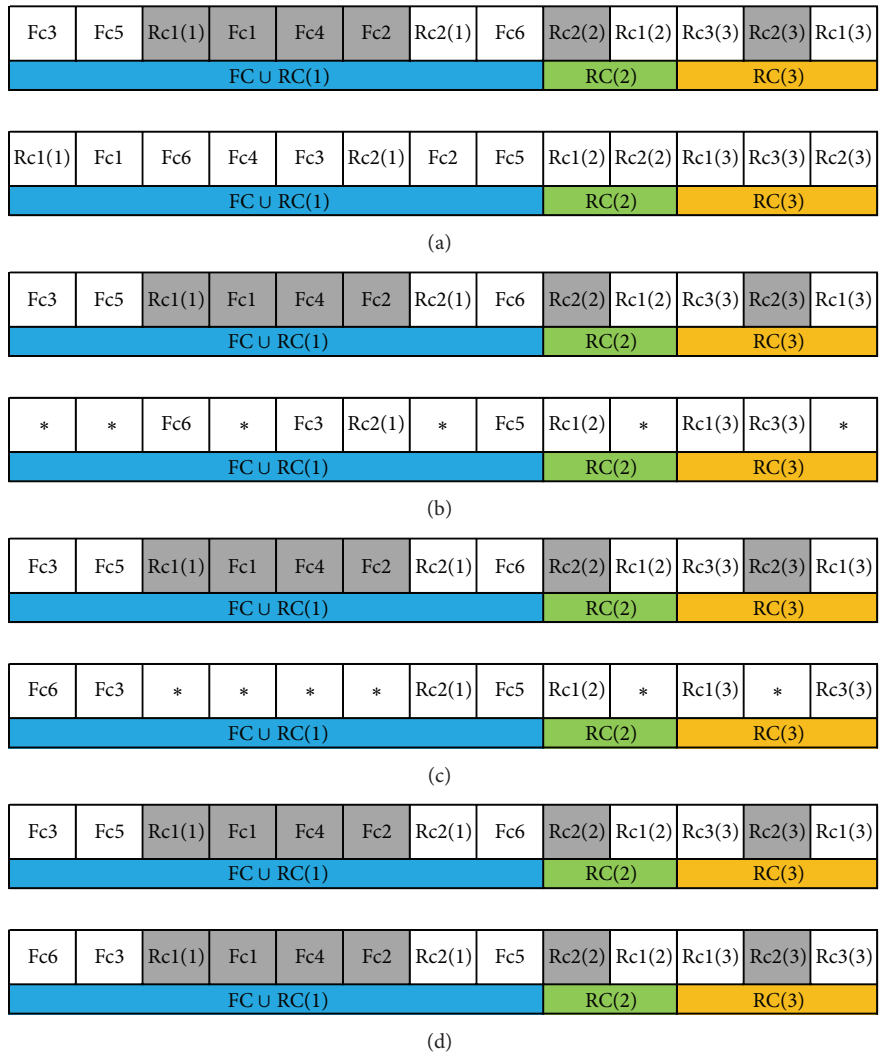

Figure 8: Nonwrapping ordered crossover (NWOX).

leading to a simple cloning process, and consequently to a unchanged situation.

To deal with this problem, the ROG technique was proposed, where before each crossover process the genetic codes are verified. In case the parents are identical, one of the two son's chromosome is completely randomized, and the other one is just cloned.

5.4.2. AGA. The AGA technique [20] does not have fixed probabilities for crossover and mutation. These probabilities are dynamically changed according to the evolution of the algorithm.

For a good evolution, a GA must have two characteristics: convergence to a minimum (local one or global one) and the wide exploration of the search tree. The balance between these two characteristics is controlled by the probability of crossover P(crossover) and probability of mutation P(mutation). For adapting the probability to the momentary situation, the authors proposed varying probabilities according to the fitness value:

$$P\,(\text{crossover}) = k_1 \frac{f_{\max} - f'}{f_{\max} - \overline{f}},$$
$$P\,(\text{mutation}) = k_2 \frac{f_{\max} - f}{f_{\max} - \overline{f}}, \tag{14}$$

where $f_{\max}$ is the greatest fitness of population, f' is the greater fitness between the crossing individuals, $\overline{f}$ is the population fitness mean, and f is the fitness value of the chromosome that will be mutated. k_1 and k_2 are control parameters of the AGA and must vary from 0 to 1.

In addition, in order for the 100% probability not to be exceeded, the functions must follow the restrictions:

$$P\,(\text{crossover}) = k_3 \quad \text{if } f' < \overline{f},$$
$$P\,(\text{crossover}) = k_4 \quad \text{if } f < \overline{f}, \tag{15}$$

where k_3 and k_4 are also control parameters.

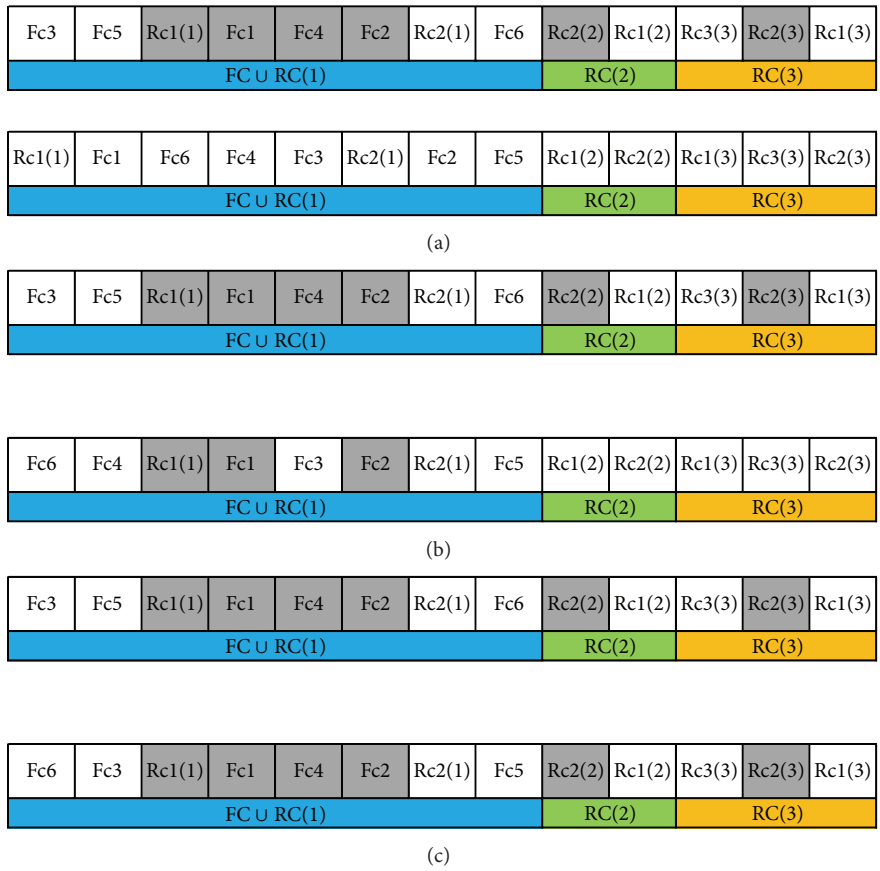

Figure 9: Partially matched crossover (PMX).

6. Experimental Results

6.1. Synthetic Applications. Given the difficulty for finding real applications adequate to PRS-NoCs in the technical literature and open access databases, nine synthetic applications were created. There are several benchmarks that could fit in simple PRS-NoCs that were used in our previous work [11]. However, those benchmarks do not fit for the placement problem.

The nine synthetic applications are based on three arrangement of cores named Alpha, Beta, and Gamma. The arrangements are described in Table 1, where the first column shows the applications, the second one shows the number of fixed cores, and the last 5 columns show the number of cores for each configuration scenario.

Starting from the basic arrangements, three core sizes variations are created for each application: (1) small cores, in which the placed cores will not violate the entry space; (2) varied cores, where small and big cores are mixed, with the placement sometimes violating the entry space; (3) big cores, in which placement will almost always violate the entry space, reducing the number of possible solutions. The application Alpha with big cores is the one presented in Figure 3. The other applications are not illustrated here due to space limitations.

6.2. Semiexhaustive Solution. In order to have an algorithmic reference to be compared to the GA solution, an exact and semiexhaustive solution was developed. The objective is to find the minimum communication cost represented by (12).

The semiexhaustive algorithm is a classic branch-and-bound algorithm to search in every permutation of each scenario. For instance, for the application Alpha with big cores (presented in Figure 3) the algorithm would test every permutation of the first scenario, and, for each one of those permutations, the two possible variations of the second scenario with the six possible variations of the third scenario. Each possibility represents a branch; when the branch violates the placement restrictions imposed by (4) and (5), it is bounded. The bound also occurs when the bandwidth capacity, represented by (9), is exceeded.

This semiexhaustive approach is similar to the one presented in [15] but more accurate, guaranteeing to achieve the global minimum. In [15], a smart-exhaustive algorithm was presented but some of the results were stuck to the local minima.

Table 2 shows the results for the semiexhaustive algorithm. The first column shows the applications and their variations are shown in second column. The global minimum cost obtained by the algorithm is shown in the third column and the algorithm execution time is in the fourth column.

Table 1: Characteristics of synthetic applications.

Application	Fixed cores	Reconfigurable cores				
		Sc 1	Sc 2	Sc 3	Sc 4	Sc 5
Alpha	6	2	2	3	N/A	N/A
Beta	4	3	5	2	4	N/A
Gamma	7	3	5	5	3	3

Table 2: Results of the semiexhaustive algorithm.

Application	Cores	Best cost	Execution time (s)	% of violation
Alpha	Small	877	197	0
	Varied	1317	89	71.3
	Big	1355	60	94.8
Beta	Small	2364	6552	0
	Varied	2518	4486	15.3
	Big	2659	698	91.4
Gamma	Small	N/A	$>8.6 \times 10^4$	N/A
	Varied	N/A	$>8.6 \times 10^4$	N/A
	Big	N/A	$>8.6 \times 10^4$	N/A

The last column shows the percentage of cases that exceeded the entry space, and consequently, were not mapped.

The algorithm has worked well for the first case, with acceptable execution times. The algorithm also presented good results for the Beta application; however, the execution time increased significantly, as expected, when compared to the first case. The explosiveness of the problem is evident when the last case is analysed. Even after 24 hours of simulation, the algorithm was not able to conclude the placement-mapping analysis.

Observing the data presented in Table 1, it is clear that the size of the NoC and the number of fixed and reconfigurable cores cause an increase in the algorithmic effort for the solution of placement-mapping problem. Therefore, for real-life nonsynthetic applications with large number of cores, there is a need for an algorithm that can solve the problem in polynomial time.

6.3. GA Results. For the simulation, three cases of crossover described in Section 5 were tested. For each type of crossover, three types of GA were considered for simulation: the ordinary GA (OGA), with no techniques for population diversification, the Adaptive GA (AGA), and the GA with random offspring generation (ROG). All algorithmic derivations were tested for each one of the nine applications, which resulted in 81 simulations. Each simulation was repeated 10 times for obtaining the average value. This was done since all the simulations are different due to the random generation of the first population.

The algorithms were simulated on MATLAB, on which the codes of the GA were developed. The simulation of the S-XY algorithm was performed on the same platform for the calculation of the communication cost.

For all simulations, it was empirically defined that the number of generations NG, of 300, offers a good trade-off between quality of solutions and computational effort. Following the same reasoning, the size of population was defined as 100 individuals. For the ordinary GA, the probability of crossover and mutation was set to 0.7 and 0.02, respectively. In the AGA the control parameters were defined as $k_1 = k_3 = 1$ and $k_2 = k_4 = 0.5$ as suggested by [20].

Table 3 shows the costs obtained from the GA for all the cases. The first column shows the applications, while their variations are listed in the second column. Columns 3, 4, and 5 show the costs for the crossover NWOX with the ordinary GA, and the variations AGA and ROG, respectively; in the same sequence, columns 6, 7, and 8 present the costs for PMX and the same process is repeated in the last three columns for the OX.

For a better visualization of results, they are presented in a color scale where the dark green represents the best solution and the red represents the worst solution.

The ordinary GA presents results, from moderate to bad results when compared to other alternatives. This can be explained by the premature convergence to a local minimum.

The approach with the ROG technique shows some good results for the Alpha application and bad results for the other applications. It was noted during the simulations that the variation of population was too high, showing that the technique is not appropriate for the specific mapping problem, preventing the convergence to a global optima in larger applications.

The best results were obtained from the AGA, showing that its mechanisms for the diversification of population have fitted better to the placement-mapping problem.

The PMX was the best of the three types of crossover, which makes the AGA with PMX the best presented solution.

In order to analyze the general quality of results obtained with the GA approach, Table 4 compares the results from GA with the results of the exact and semiexhaustive algorithms.

Table 3: Minimum communication costs obtained from the GAs.

Application	Cores	NWOX			PMX			OX		
		OGA	AGA	ROG	OGA	AGA	ROG	OGA	AGA	ROG
Alpha	Small	1011.4	907.5	924.6	1003.6	900.7	968.8	974.0	903.0	942.9
	Varied	1477.1	1417.8	1520.6	1559.1	1386.8	1456.0	1530.5	1424.0	1557.0
	Big	1686.5	1630.1	1537.7	1697.5	1512.7	1636.5	1645.7	1594.3	1537.8
Beta	Small	2576.0	2485.0	2953.3	2503.8	2395.0	2796.1	2569.0	2377.8	2644.3
	Varied	2550.7	2532.1	2827.1	2541.9	2533.3	2815.7	2607.8	2535.1	2764.4
	Big	2820.3	2771.8	3107.2	2729.9	2760.2	3148.2	2899.0	2797.5	3027.3
Gamma	Small	3932.7	3660.9	4407.8	3970.8	3679.6	4535.6	3926.2	3657.4	4386.7
	Varied	4072.8	3931.6	4565.6	4102.0	3888.6	4435.4	4141.2	3964.8	4463.6
	Big	4268.2	4109.2	4783.7	4159.2	3974.0	4822.2	4223.1	4049.6	4814.6

Table 4: Cost penalty from gas when compared to the minimal solution.

Application	Cores	NWOX			PMX			OX		
		OGA	AGA	ROG	OGA	AGA	ROG	OGA	AGA	ROG
Alpha	Small	15.3%	3.5%	5.4%	14.4%	2.7%	10.5%	11.1%	3.0%	7.5%
	Varied	12.2%	7.7%	15.5%	18.4%	5.3%	10.6%	16.2%	8.1%	18.2%
	Big	24.5%	20.3%	13.5%	25.3%	11.6%	20.8%	21.5%	17.7%	13.5%
Beta	Small	9.0%	5.1%	24.9%	5.9%	1.3%	18.3%	8.7%	0.6%	11.9%
	Varied	1.3%	0.6%	12.3%	0.9%	0.6%	11.8%	3.6%	0.7%	9.8%
	Big	4.4%	2.6%	15.0%	1.0%	2.2%	16.5%	7.3%	3.5%	12.0%
Gamma	Small	**7.5%**	**0.1%**	**20.5%**	**8.6%**	**0.6%**	**24.0%**	**7.3%**	**0.0%**	**19.9%**
	Varied	**4.7%**	**1.1%**	**17.4%**	**5.5%**	**0.0%**	**14.1%**	**6.5%**	**2.0%**	**14.8%**
	Big	**7.4%**	**3.4%**	**20.4%**	**4.7%**	**0.0%**	**21.3%**	**6.3%**	**1.9%**	**21.2%**

The percentage indicates how much the cost is above the minimum cost; this index will be named cost penalty. The Gamma application (bold font), was compared with the best cost obtained by the GA, since it was not possible to obtain the global minimum cost with the semiexhaustive algorithm, as previously described in Table 2.

The best solution (PMX-AGA) presented penalties between 0.6% and 11.6% with an average penalty of 4%. Other crossovers presented similar results: the NWOX-AGA reached from 0.6% to 20.3% with 6.6% of average and the OX-AGA presented from 0.6% to 17.7% with 5.6% of average penalty.

The mappings for the Alpha application with large size cores presented the worst results. As the occupation of the PRS-NoC was the largest among the synthetic applications, it was difficult for the algorithm to achieve convergence to a global minimum, since a great number of mappings violated the restrictions.

The Alpha application took at most 48 seconds to be mapped using the GA. The Beta and Gamma applications took at most 73 and 113 seconds to be mapped, respectively. All the simulations were executed in a PC with an Intel Core I7-3770 processor and 8 GB of RAM. The execution times have shown that the algorithm is completely acceptable for a design-time activity.

7. Conclusion

This work presented GAs solutions for placement and mapping for NoC-based reconfigurable systems. The problem formulation was developed in an efficient way, since all important aspects of PRS-NoC were successfully described, in order to include all important aspects of a PRS-NoC under irregular and indirect communication network. The formulation was the base for the development of GA solutions.

The problem was adapted for GAs, which enabled the solution for a great variety of applications. Three crossovers were tested: the NWOX, the PMX, and the OX. Each crossover was tested for two techniques of population diversification: the AGA and the ROG.

The different types of crossover presented similar results, where the PMX was slightly better. The ordinary GA and the ROG technique tended to converge to a global minimum, while the AGA fitted better to the placement-mapping problem. The AGA results were close to the global minimum cost when combined with any crossover type. The AGA combined with PMX presented the best results, with penalties between 0.6% and 11.6% and an average penalty of 4%.

The execution time was 48, 73, and 113 seconds for applications with 13, 16, and 26 cores, respectively, to execute the entire placement-mapping process. The execution times

are completely acceptable for a design-time activity. It also shows that the algorithm is able to solve the problem for a relative large number of cores.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors would like to thank to the Brazilian Council for Scientific and Technological Development (CNPq) for the financial support.

References

[1] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "Dynoc: a dynamic infrastructure for communication in dynamically reconfugurable devices," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 153–158, August 2005.

[2] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to networks-on-chips," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 1–6, August 2006.

[3] S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda, "CuNoC: a scalable dynamic NoC for dynamically reconflgurable FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 753–756, Amsterdam, The Netherlands, August 2007.

[4] C. Killian, C. Tanougast, F. Monteiro, and A. Dandache, "A new efficient and reliable dynamically reconfigurable network-on-chip," *Journal of Electrical and Computer Engineering*, vol. 2012, Article ID 843239, 16 pages, 2012.

[5] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3–21, 2009.

[6] M. Huebner, T. Becker, and J. Becker, "Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration," in *Proceedings of the 17th Symposium on Integrated Circuits and Systems Design (SBCCI '04)*, pp. 28–32, IEEE, September 2004.

[7] M. Raffo, J. G. Filho, M. Strum, and W. J. Chau, "A placement tool for a NOC-based dynamically reconfigurable system," in *Proceedings of the VI Southern Programmable Logic Conference (SPL '10)*, pp. 47–52, March 2010.

[8] G. Haiyun, "Survey of dynamically reconfigurable network-on-chip," in *Proceedings of the International Conference on Future Computer Sciences and Application (ICFCSA '11)*, pp. 200–203, June 2011.

[9] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, vol. 2, pp. 896–901, February 2004.

[10] F. Wronski, E. Briao, and F. Wagner, "Evaluating energy-aware task allocation strategies for MPSoCs," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, pp. 215–224, Springer, Braga, Portugal, 2006.

[11] J. G. Filho, M. Strum, and W. J. Chau, "A strategy for mapping reconfigurable cores in NoCs," in *Proceedings of the IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS '13)*, pp. 1–4, March 2013.

[12] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 1825–1831, April 2004.

[13] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, T. Arslan, and I. Martinez, "Runtime scheduling, allocation, and execution of real-time hardware tasks onto Xilinx FPGAs subject to fault occurrence," *International Journal of Reconfigurable Computing*, vol. 2013, Article ID 905057, 32 pages, 2013.

[14] I. Beretta, V. Rana, D. Atienza, and D. Sciuto, "A mapping flow for dynamically reconfigurable multi-core system-on-chip design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1211–1224, 2011.

[15] J. G. Filho and W. J. Chau, "Exploring the problems of placement and mapping in NoC-based reconfizurable systems," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '13)*, pp. 1–4, IEEE, Cancún, Mexico, December 2013.

[16] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. C. van der Veen, "Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 673–680, 2007.

[17] C. A. M. Marcon, E. I. Moreno, N. L. V. Calazans, and F. G. Moraes, "Comparison of network-on-chip mapping algorithms targeting low energy consumption," *IET Computers and Digital Techniques*, vol. 2, no. 6, pp. 471–482, 2008.

[18] M. J. Sepúlveda, W. J. Chau, G. Gogniat, and M. Strum, "A multi-objective adaptive immune algorithm for multi-application NoC mapping," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 3, pp. 851–860, 2012.

[19] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, vol. 412, Addison-Wesley, Reading, Mass, USA, 1989.

[20] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.

[21] V. A. Cicirello, "Non-wrapping order crossover: an order preserving crossover operator that respects absolute position," in *Proceedings of the 8th ACM Annual Conference on Genetic and Evolutionary Computation*, pp. 1125–1132, 2006.

[22] J. Holland, "Reproductive plans and genetic operators," in *Adaptation in Natural and Artificial Systems*, chapter 6, pp. 89–118, MIT Press, Cambridge, Mass, USA, 1992.

[23] M. Rocha and J. Neves, "Preventing premature convergence to local optima in genetic algorithms via random offspring generation," in *Multiple Approaches to Intelligent Systems*, Lecture Notes in Computer Science, pp. 127–136, Springer, Berlin, Germany, 1999.