Journal of Systems Architecture 57 (2011) 761–777

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Fast placement and routing by extending coarse-grained reconfigurable arrays with Omega Networks

Ricardo S. Ferreira [a], João M.P. Cardoso [b,*], Alex Damiany [a], Julio Vendramini [a], Tiago Teixeira [a]

[a] Departamento de Informática, Universidade Federal de Viçosa, 36571-000 Vicosa, MG, Brazil
[b] Departamento de Engenharia Informática, Faculdade de Engenharia (FEUP), Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

ARTICLE INFO

Article history:
Received 19 May 2010
Received in revised form 23 December 2010
Accepted 22 March 2011
Available online 15 April 2011

Keywords:
Coarse-grained reconfigurable arrays
Multistage interconnection networks
Placement and Routing
FPGAs

ABSTRACT

Reconfigurable computing architectures are commonly used for accelerating applications and/or for achieving energy savings. However, most reconfigurable computing architectures suffer from computationally demanding placement and routing (P&R) steps. This problem may disable their use in systems requiring dynamic compilation (e.g., to guarantee application portability in embedded systems). Bearing in mind the simplification of P&R steps, this paper presents and analyzes a coarse-grained reconfigurable array (CGRA) extended with global multistage interconnect networks, specifically Omega Networks. We show that integrating one or two Omega Networks in a CGRA permits to simplify the P&R stage resulting in both low hardware resource overhead and low performance degradation (18% for an 8×8 array). We compare the proposed CGRA, which integrates one or two Omega Networks, with a CGRA based on a grid of processing elements with reach neighbor interconnections and with a torus topology. The execution time needed to perform the P&R stage for the two array architectures shows that the array using two Omega Networks needs a far simpler and faster P&R. The P&R stage in our approach completed on average in about 16× less time for the 17 benchmarks used. Similar fast approaches needed CGRAs with more complex interconnect resources in order to allow most of the benchmarks used to be successfully placed and routed.

1. Introduction

Reconfigurable computing architectures are becoming increasingly important in embedded and high-performance computing systems [1–3]. The dominant reconfigurable fabrics are the Field-Programmable Gate Arrays (FPGAs) [4], well-known by their flexibility to prototype digital circuits and entire systems, and to implement from application-specific architectures to common microprocessors (e.g., as *softcores*). Contemporary FPGAs provide fine-grained hardware resources, arithmetic units, and distributed memories [5]. Recently, FPGAs have been thought as a technology avenue for implementing a myriad of different architectures such as traditional microprocessors, VLIW-based architectures [6], systolic arrays [7], and coarse-grained reconfigurable arrays (CGRAs) [8,9]. In fact, FPGAs promise to reshape the computer architecture landscape by providing hardware resources that can be used to employ the best architecture for a section of an application, a particular application, or even a specific domain.

CGRAs are valuable extensions to microprocessors for many applications. They can be used to accelerate applications or to reduce overall energy consumption [10]. There have been several proposed CGRAs, both in academia [11–14] and in industry [15]. CGRAs are mainly based on a matrix of processing elements (PEs) seconded by routing resources.

Besides the manufacturing of a reconfigurable fabric with a CGRA architecture [16] or the manufacturing of a System-On-a-Chip (SoC) including a CGRA as a system component [17], CGRAs can be also used as *softcores* implemented by the fine-grained hardware resources of FPGAs. This kind of implementation has demonstrated to be an efficient option as it can be able to achieve speedups over embedded microprocessors and also over desktop microprocessors running at high-frequencies. As an evidence of this, e.g., the work in [9] presents speedups for a number of applications ranging from 115 to 298 and from 7 to 16 over an embedded processor and over a desktop processor, respectively. Another advantage of using CGRAs is the fact that they allow a mesh-based interconnect programmable layer on top of the FPGA, with regular structure, with placement and routing pre-optimized, and without requiring subsequent modifications. Based on this, one can achieve comparable performance for the implementations obtained by mapping well-known digital signal processing algorithms (most regular) into this layer (i.e., the one provided by the CGRA structures on top of the FPGA resources) when compared to implementations obtained by high-level synthesis [18]. Note that the

* Corresponding author.
E-mail addresses: ricardo@ufv.br (R.S. Ferreira), jmpc@acm.org (J.M.P. Cardoso).

1383-7621/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2011.03.006

application-specific architectures generated by high-level synthesis tools, at a first glance would be expected to achieve higher performance, may suffer from non-optimal P&R and from long interconnections and irregular structures [19].

The possibility to efficiently and dynamically map computations on FPGAs in general, and CGRAs in particular, is foreseen as a feature needed for the acceptance and success of those architectures in the embedded computing domain. In most embedded systems, applications' portability is very important and is commonly ensured by virtual machines, just-in-time (JIT) compilers, and dynamic binary translation. However, typical CGRAs (e.g., ADRES [14], XPP [15]) need computationally demanding P&R steps (albeit being simpler than the P&R steps of FPGAs). Those steps make difficult the use of reconfigurable computing architectures in JIT compilation environments. Thus, in order to dynamically compile segments of an application to CGRAs, the P&R steps need to be simplified. To be successful, we believe, this simplification needs to rely in both architectural support and algorithm simplification. One of the hypothetical possibilities would be to extend a CGRA with routing resources that allow to directly connect every two PEs. In this case, placement can be reduced to the assignment of instructions to the PEs (without location awareness) supporting the operations involved, and routing is also simplified. However, this reach interconnect possibility is inefficient and impractical for CGRAs with many PEs.

In this paper we propose a novel solution that includes a CGRA architecture with local interconnections between neighborhood PEs and global interconnections allowed by multistage interconnect networks [20], specifically Omega Networks [21]. Our approach, as most CGRAs, relies on a compile/mapping time step to generate the configurations that define the placement of operations and the routing between PEs. During execution of the computations in a configuration, data flow through the routing paths "statically" defined by the mapping. Thus, we are not addressing CGRAs with the capability to do dynamic routing. A first version of the architecture has been presented in [22]. Our proposed architecture allows a fast P&R, with polynomial computing complexity and memory usage efficiency. It permits to achieve light dynamic P&R steps and permits a dynamic dispatch approach possibly without analyzing large instruction windows to achieve efficient results. The main contributions and results presented in this paper are:

- A novel CGRA using Omega Networks to achieve a low-cost global network is proposed. An analysis and study of the performance and the hardware resources overhead are presented for this novel CGRA.
- A simplified P&R algorithm, with polynomial complexity, is presented for the CGRA proposed. The simplified P&R and the novel CGRA topology allowed, on average, the mapping of dataflow graphs in 16× less execution time than using a grid-based CGRA with a reach interconnect topology.

This paper is organized as follows. Section 2 introduces the main concepts related to CGRAs and multistage interconnect networks, and the motivation behind our work. Section 3 describes our approach using CGRAs with Omega Networks and presents the P&R algorithm for the proposed CGRA. Section 4 presents experimental results and Section 5 describes related work. Finally, Section 6 draws some conclusions.

2. Background and motivation

Before introducing our approach, we briefly present in this section the main properties of coarse-grained reconfigurable arrays (CGRAs) and of multistage interconnection networks (MINs).

2.1. Coarse-grained reconfigurable architectures

CGRAs mainly consist of a set of processing elements (PEs) connected with a certain interconnect topology. Each PE is associated with one or more functional units (FUs) which are responsible for performing a number of operations (including arithmetic, logic, and special operations to deal with conditional branches). CGRAs support both spatial and temporal computations and fulfill high degrees of parallelism (e.g., from operation- to task-level).

Different interconnect topologies have been proposed for CGRAs [1,14,15,23]. For instance, grid topologies form a 2-D array of PEs with neighborhood connections between them (as illustrated in Fig. 1). Each PE includes hardware structures to allow the connection of each input port to an operand of the FU and each output of the FU to an output port of the PE. More complex PEs may also include interconnect resources to route inputs directly to the outputs without passing through the FU [14].

An example of a 2-D Grid is depicted in Fig. 1. It uses 4/4 (unidirectional inputs/outputs) PEs and a torus topology. Each PE receives as inputs the signals from north (N), east (E), west (W) and south (S). These inputs are connected to the FU using a multiplexer for each FU operand, and the FU result is connected to the PE outputs directly or passing through multiplexers. In the presence of PEs which consider all the routing possibilities, four multiplexers are needed as shown in Fig. 2(a). These multiplexers ensure that the output of the FU and the inputs of the PE are able to reach each of the PE outputs. A simpler option, depicted in Fig. 2(b), reduces the number of output multiplexers by only allowing that the PE outputs come from the output of the FU. In this latter case, the connection between an input to an output of the PE is achieved by a bypassing operation in the FU. In this case, however, the PE resources are solely used for routing and without the possibility for being used for both, i.e., routing and operation. In addition, when using this limited routing internal structure only a single PE input can be connected to the PE outputs. There are, however, other possibilities between the two PE internal routing structures illustrated in Fig. 2(a and b). Also note that each PE usually has its outputs registered.

In practice, 2-D CGRAs have been enriched by more neighborhood and non-neighborhood interconnections. Previous work has shown that PEs with eight inputs and eight outputs, and interconnect topologies using 1-hop (interconnect resources able to connect directly one PE to non-neighborhood PEs), achieve the best placement and

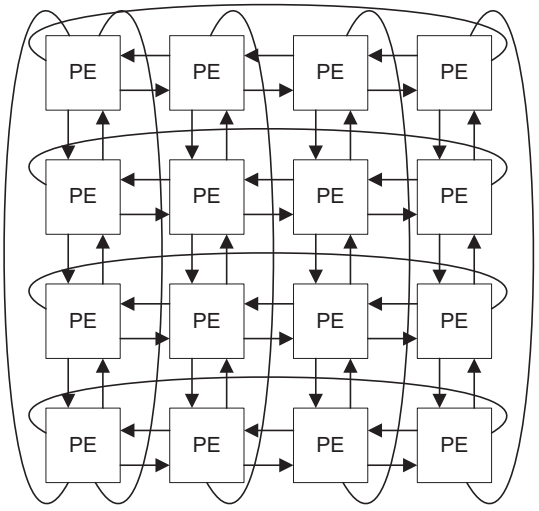

Fig. 1. A 2-D Grid-based CGRA with PEs with four inputs and four outputs, and considering neighborhood and torus connections (not all shown).

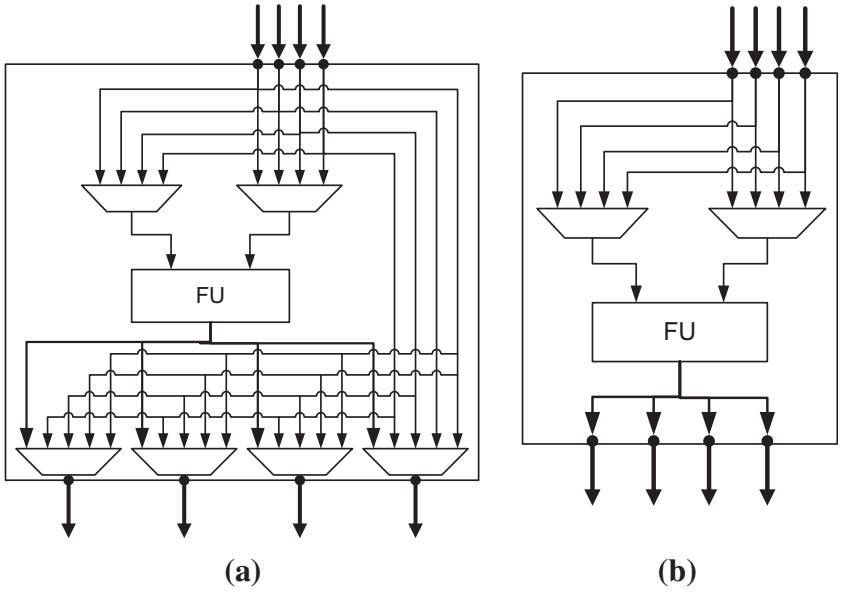

Fig. 2. PEs with four inputs and four outputs: (a) interconnect resources to connect the PE to other PEs (e.g., the neighbors in the grid) and to permit the routing of inputs directly to one or more of the outputs; (b) a simpler PE without including routing of inputs to outputs, unless they pass through the FU.

routing (P&R) results [14,23]. However, when a 1-hop routing PE with eight-inputs/eight-outputs is considered, the associated cost might be too high, e.g., requiring ten 8:1 multiplexers per PE.

The use of the internal routing resources in each PE of the CGRA (illustrated with the two examples in Fig. 2) affects its maximum clock frequency and the hardware resources used. Fig. 3 shows results when mapping PEs with different number of input/outputs. We show normalized results to the maximum clock frequency and to the number of LUTs used by the FU itself. The target for this study was a Xilinx Virtex-5 FPGA. This study shows that it is not a good approach to increase too much the number of input/outputs for maintaining the possibility to route internally the inputs to the outputs of the PE, e.g., for a 4/4 PE and for an 8/8 PE we may need 1.7× and 2.6× more hardware resources between the simpler and the more complex PE of Fig. 2, respectively.

At a certain point, one needs to make a trade-off between the overhead needed by richer interconnect resources and the complexity of the P&R stage. An ideal routing topology would make possible that each PE could directly connect with zero or low-cost overhead to any other PE in the CGRA. This routing topology would make an easy and fast P&R stage. However, such rich interconnect topologies have in practice high and usually unacceptable costs. For instance, a full crossbar has a high connectivity and could be one such option, but its area cost of $O(N^2)$ prevents its use for typical CGRA sizes. In another extreme of the spectrum of interconnect resources is the bus. Buses have been used in many CGRAs [1] and augment the interconnect resources with lines that can connect one source PE (from multiple PEs) to one or more destination PEs. Note, however, that the use of buses in typical PEs is defined during the placement and routing and there are not time-multiplexed routing paths during the same configuration (*bitstream*), i.e., in each configuration each bus is defined to perform an interconnect pattern. To simplify the mapping and the execution process, most CGRAs are configured to execute the current application without generically considering the time-multiplexing of hardware resources such as buses. Thus, some architectures need to use several buses to maintain routing capacity at high levels. As long lines can be expensive resources (especially when targeting to FPGAs), segmented lines are usually used because they allow a lower cost solution. We propose in this paper an intermediate solution, which takes advantage of multistage interconnection networks.

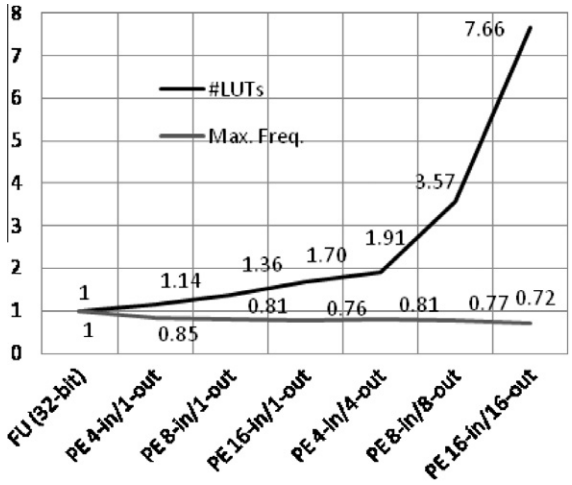

Fig. 3. The effect of the number of input/outputs of the PE in the hardware resources needed and in the maximum clock frequency achieved (normalized values).

Although one can think about CGRAs where the routing paths are defined and adapted dynamically, our approach addresses CGRAs with all the operations on the PEs and the routing paths defined during the mapping and programmed by each configuration.

2.2. Multistage interconnection networks

A realistic possibility, proposed and analyzed in this paper, is to diminish the number of possible interconnections by using multistage interconnection networks (MINs) [20]. MINs offer a good cost/connectivity balance and are an intermediate solution for global routing, much cheaper than crossbars. They consist of a number of input/output terminals and stages of switches (also called interchange boxes). Each stage contributes to the global routing of the inputs to the output terminals.

MINs are networks with N inputs and N outputs and composed by M switch stages (see Fig. 4(b)). The more common switches are

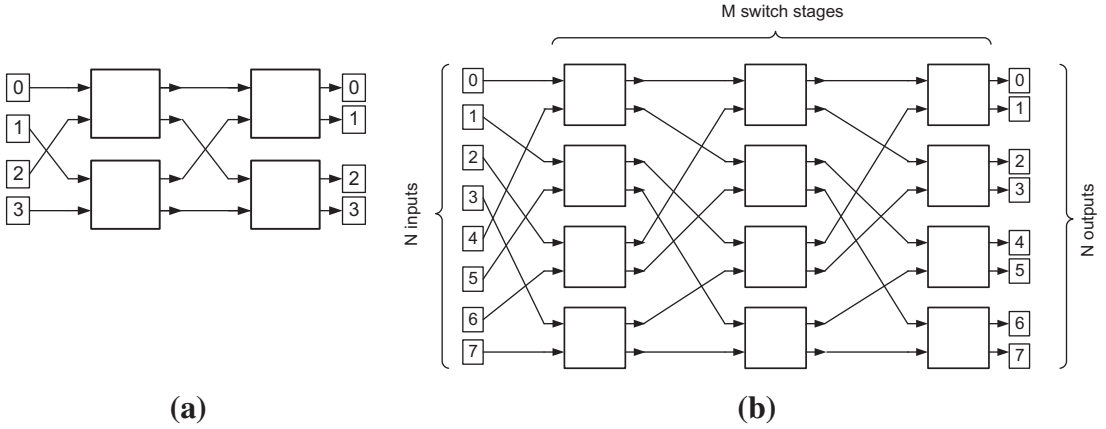

Fig. 4. Examples of Omega MINs: (a) 4×4 and (b) 8×8.

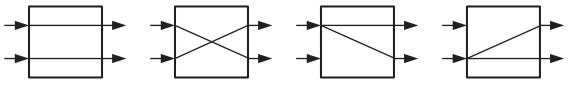

Fig. 5. 2×2 Interchange boxes (switches) and the possible connections: straight, crossover, upper broadcast, and lower broadcast.

the 2×2 switches shown in Fig. 5. The MINs using these switches are usually known as radix-2 MINs. In this case, and being N de number of inputs of the network, MINs have a hardware resource cost of $O(N \times \text{Log}_2(N))$. Most MINs consider for N, power of two values, with M ranging from $\text{Log}_2(N)$ to $2 \times \text{Log}_2(N)$ [24–26]. MINs include $N/2$ switches per stage resulting in a total of switches from $S = N/2 \times \text{Log}_2(N)$ to $S = N \times \text{Log}_2(N)$. Note that it is possible to build MINs with larger switches, such as 4×4 switches (i.e., radix-4 switches). In this case the number of stages decreases, but the complexity of each switch is higher.

Although the 2^S possible permutations and $N!$ possible allowed connections[1] for a number of stages equal to $M = \text{Log}_2(N)$ and $S = N/2 \times M$ switches, the MIN connectivity is restricted. According to the connections to route, path and output port contentions/conflicts may exist. These MINs are classified as blocking networks, and some input/output permutations are not possible to co-exist. In these blocking networks, each input/output connection has a unique path [4] and the intermediate switches can be shared by paths.

However, a MIN with $2 \times \text{Log}_2(N)$ stages can be a rearrangeable non-blocking, and in this case it is possible to rearrange some connections to solve the routing conflicts. For instance, a Benes network [27,28] is an example of a rearrangeable non-blocking network. An intermediate solution is to extend a MIN with K additional switch stages. In this case, the network has $\text{Log}_2(N) + K$ stages, where $0 \leqslant K \leqslant \text{Log}_2(N)$ [26]. When $K = 0$, the MIN has a minimum cost and delay, but it is still a blocking network. When $K = \text{Log}_2(N)$, the area cost and the delay/latency double, but the connectivity is significantly higher. An acceptable trade-off can be obtained using low values of K.

MINs have been extensively explored over the past years [29]. They have been used in many application domains such as SIMD and MIMD parallel machines, ATM switches, and more recently new reconfigurable fabrics [30], and large cluster-based supercomputers [31]. Although it is clear that MINs can be efficiently implemented at layout level [32], their efficiency when implemented by FPGA resources was not clear. We also analyze in this paper this efficiency.

[1] Note that for N terminals we have N^N possible connections.

MINs are bit permutation networks, where the connections between two stages are based on bit operations. MINs differ in the interconnect pattern used between stages. There are various examples of MINs [20]: Baseline, Banyan, Benes, etc. A particular case of MINs using a Perfect Shuffle interconnect pattern and using switches with only straight and crossover operations (see Fig. 5) are the Omega Networks [21]. Examples of 4×4 and 8×8 Omega Networks are illustrated in Fig. 4. An Omega Network uses a circular rotate-left for the bit permutation, e.g., the line 5 (1 0 1) in the output of stage i is connected to line 3 (0 1 1) of the input stage $i + 1$.

An important characteristic of the MINs is the fact that routing can be accomplished with algorithms with complexity $N \times \log_2(N)$ [33,34]. Another important characteristic of most blocking MINs is their self-routing property, i.e., the local routing decision in each cell is done only based on the binary representation of the address of the target output port. Omega MINs exhibit these characteristics and permit a very easy routing scheme based on SHIFTs by pre-defined constants and AND operators. This is an important aspect for CGRAs, because we can include specific hardware for dynamically routing through the MIN. Note, however, that although this is one of the properties that strained us to select the Omega as the MIN for our CGRAs, extending MINs with hardware to support dynamic routing is out of the scope of this paper. Even thought that MINs such as Baseline and Butterfly have been considered topologically and functionally equivalent [35] and could be used in our approach as well, their routing scheme involves bit insertion operators.

Omega Networks are blocking. Thus, it may be impossible to connect two different input/output pairs at the same time. Consider a 4×4 Omega Network and three connections needing routing: $3 \rightarrow 1$, $0 \rightarrow 2$ and $2 \rightarrow 3$, as shown in Fig. 6(a). After successfully routing $3 \rightarrow 1$ and $0 \rightarrow 2$, one can see that the routing of connection $2 \rightarrow 3$ needs to use resources already used by the routing of connection $0 \rightarrow 2$, thus producing a conflict. Using this Omega Network we are only able to route at the same time the connection $0 \rightarrow 2$ or the $2 \rightarrow 3$ and not both. Fig. 6(b) shows how an extra stage ($K = 1$) resolves the conflict illustrated in Fig. 6(a) and allows the routing of the three connections.

The calculation of the routing capacity of a MIN with extra stages ($K > 0$) is a complex problem [36]. As the number of possible permutations grows faster ($N!$), we study here the routing capacities of Omega MINs by using sets of sample permutations. Fig. 7 shows the percentage of successfully routed connections (i.e., without conflicts) over a sample of 10^8 permutations considering Omega MINs with N terminals (N inputs and N outputs). We present percentages of successfully routed connections for different Omega MINs considering the inclusion of 1, 2 and 4 extra stages

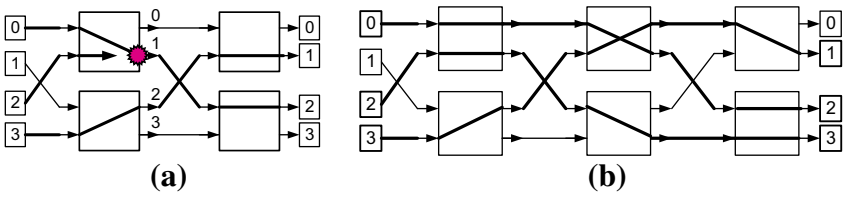

Fig. 6. Congestion in a 4 × 4 Omega, multistage interconnect network (MIN): (a) with two stages and a routing congestion; (b) with three stages without the previous routing congestion.

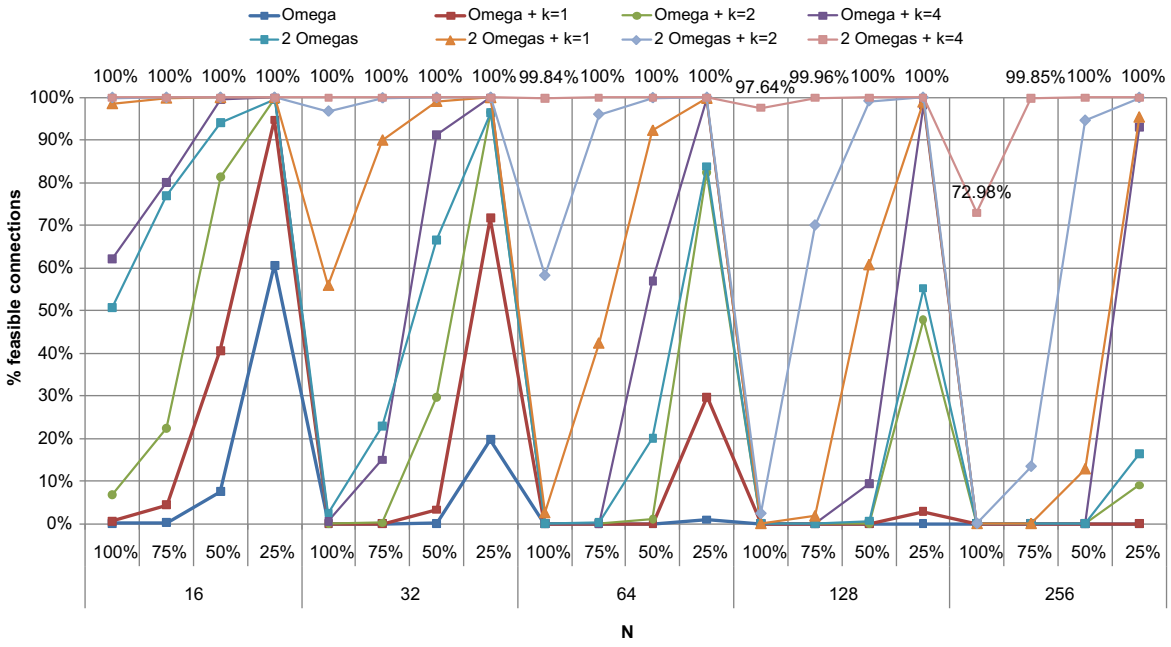

Fig. 7. Percentage of connections successfully routed when using a specific number of terminals (*N* inputs and *N* outputs) of a multistage interconnection network over a maximum of 10,00,00,000 permutations.

(i.e., $K = 1,2,4$), and the individual percentages achieved when considering a maximum of 10^8 possible connections using 25%, 50%, 75% and 100% of the terminals of the network.

As expected, the routability increases as *K* increases and as the number of possible connections decreases. The inclusion of an additional Omega MIN and extra stages (1, 2 and 4) greatly increases the connections successfully routed. As an example, for an Omega MIN of 64 terminals (64 × 64) and considering 50% of terminals used (32), we increase from 57% to 92% the percentage of connections routed by using two 64 × 64 Omega MINs with one extra stage in each one (seven stages) instead of using only one 64 × 64 Omega MIN with 10 stages. An important conclusion of these results is the fact that with two Omega MINs with two extra stages each one, and considering the use of half of the terminals at the maximum, we obtain high percentages of successful routing connections, even for Omega MINs very large (256 terminals). For solutions requiring less than 30% of the Omega MIN terminals, this approach guarantees almost 100% of routability. As the results presented will show (cf. experimental results), this 30% is enough for the benchmarks we have evaluated.

The work presented in this paper extends CGRAs with global MINs, in particular Omega Networks, in order to reduce the P&R complexity. Although MINs can be used in the context of dynamic routing, the work presented in this paper considers that the MINs are used to implement a set of routing paths in a configuration and this set does not change during the use (execution) of the configuration. Thus, the blocking property of the Omega Networks may prevent some routing paths, but this is dealt during compilation, more specifically, during placement and routing. The congestions that have been referred are considered during placement and routing performed prior to the generation of the *bitstreams* that will be part of a particular configuration.

To the best of our knowledge, this is the first work analyzing both the use of global Omega Networks in 2-D CGRAs in terms of area and delay overhead and in terms of the complexity of the P&R. The following section presents the proposed CGRA.

3. Coarse-grained reconfigurable array plus Omega Networks

As already referred, previous work on CGRAs proposed the use of global buses, line and/or column buses, inter-cluster or crossbar networks [5], as routing resources to achieve successful routing on 2-D arrays. Buses have inherently low cost, but to solve congestion problems may require time-sharing of resources and scheduling. To reduce the complexity of the type of interconnect resources needed, to enrich the routing possibilities and the associated P&R algorithms, we propose a hybrid solution with only two types of resources for local and global interconnects, respectively.

3.1. Architecture

While most previous CGRAs addressed PEs with routing resources (see, e.g., [14,23,28]), our proposed CGRA simplifies the

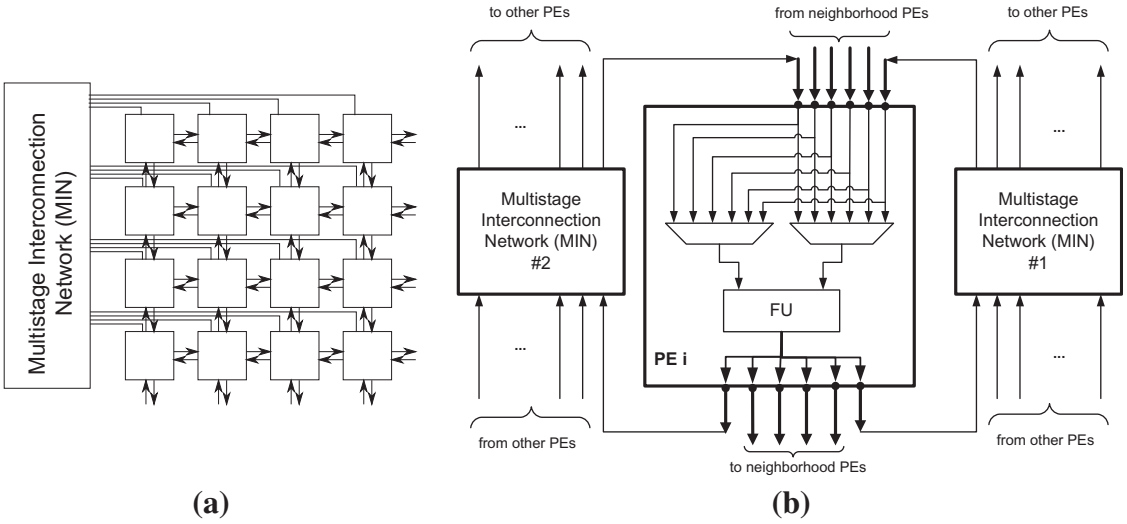

Fig. 8. Architecture of the proposed CGRA: (a) CGRA with a multistage interconnection network (MIN) and (b) example of PE with interconnections to two MINs.

PE. By using MIN structures for achieving global connections between PEs, we do not need, regarding routing local support, the PE complexity exhibited in Fig. 2(a). In fact, our approach allows the use of PEs as the one presented in Fig. 2(b), which represent the extremity of the spectrum regarding the internal routing complexity of the PE. This type of PE uses only two FU multiplexers, and no bypass routing is included (i.e., direct routing from inputs to outputs of the PE). By doing so, we achieve PEs faster and with less hardware resources.

The use of MINs as depicted in Fig. 8(a) gives to the CGRA a rich interconnect structure by allowing connections between PEs through the MINs. Specifically, non-local routing (i.e., routing between non-adjacent PEs) is performed through the MINs. Note, however, that the use of MINs does not prevent CGRAs to use e.g., 1-hop connections (this may minimize the use of MINs).

To decrease routing conflicts and thus making possible to P&R as many as possible kernels with a simple P&R algorithm, we propose to use more than one MIN, each one with extra stages. Fig. 8(b) illustrates the use of two MINs and their connections to the PEs of a CGRA. Although MINs only allow a subset of all the possible connections given N inputs and Noutputs, they do not prevent us to map complex kernels as we demonstrate in the experimental results section of this paper. This is explained by the fact that most DFG edges represent local connections and the MINs are only used for routing a small number of DFG connections. The experiments done so far show us that usually the number of terminals used for each MIN in the architecture is less than 30% of the total number of its terminals.

As referred, Omega is a non-rearrangeable MIN with simple and self-routing properties well suited for our purposes. In addition, the use of rearrangeable MINs would require at least $2 \times \log_2(N) - 1$ stages instead of the $\log_2(N) + K$ stages used by Omega MINs with K extra stages. This is very important, especially given the small values of K needed (see Fig. 7), because of the lower delay and hardware cost of the Omega Network compared to a rearrangeable one. Based on the reasons already presented, we use Omega MINs and the results presented consider Omega MINs using 2×2 switches. For simplicity we use a CGRA architecture without torus interconnections.

The use of MINs in the context of CGRAs has also another positive impact. It is well known that the routing on a MIN requires a polynomial solution (see the routing algorithms presented in [24–26,28]), while the routing on a 2-D Grid is NP-complete. This is very important, especially if the addition of MINs to a CGRA allows the reduction of the complexity of the P&R algorithm. Specifically, the architecture proposed permits to use a polynomial P&R algorithm as the one proposed and detailed in the next section.

3.2. Placement and Routing

As mentioned before, the P&R on a 2-D array is an NP-complete problem [37]. To simplify, P&R is usually solved in a two step approach: first the placement is performed, and then a routing algorithm is applied. Placement has been traditionally based on meta-heuristics [37], such as genetic and simulated annealing algorithms. The routing step is usually based on pathfinder [38] and similar algorithms [14,37]. Although such approaches achieve a trade-off between P&R results and execution time, they continue to be time consuming tasks, a fact that hampers their use when runtime P&R is required.

We consider that each part of the application to be mapped to the CGRA is represented as a dataflow graph (DFG). The DFG, $G = (V, E)$, consists of a set of nodes V representing operations, and a set of edges E connecting operations. We assume that each operation in the DFG requires a distinct PE of the CGRA. The CGRA is also represented as a directed graph with nodes representing PEs and edges representing the interconnect resources.

Based on the interconnect topology of the CGRA previously explained, our approach performs P&R simultaneously by using a one step graph traversal. Given a DFG, our approach tries to assign local routing resources in the CGRA for the edges of the DFG. Naturally, after this step, there might be unrouted DFG edges. Those unrouted remaining edges are then considered for routing using MIN interconnect resources.

The first stage of the P&R algorithm, illustrated in Fig. 9, is based on the depth-first search, over the input DFG, previously presented in [39]. This first stage is applied for each node v in V. A depth-first traversal starts in a node v and traverses the DFG until reaches an end node. Then the nodes in the path from v to the end node are sequentially placed on the CGRA. At the same time, neighborhood PE connections are assigned to the edges between nodes in the path. Edges without neighborhood PE connections are maintained in a set of unrouted edges to be considered by the second phase of the algorithm. The depth-first continues until all nodes and edges are visited. As referred before, this P&R approach has polynomial complexity and thus can be an interesting solution for runtime P&R.

Algorithm 1. First stage of P&R

```
DepthFirstPR(Node Current, NodeList Path, PE FirstPE)
1.  Add Current DFG node to Path
2.  Add Sink unplaced nodes of Current node to List
3.  if List is empty then // end node
4.     MapToGrid(Path, FirstPE); // depth first in Grid
5.     return from algorithm;
6.  end if

7.  X ← Get and remove first node from the List
8.  DepthFirstPR(X, Path, FirstPE); //  traversal in depth first order

    // get the PE where Current Node has been mapped
9.  PE_i ← Get the PE where Current DFG node has been mapped

10. while List is not empty do
       // start a new Path for other descendents of current node
11.    X ← Get and remove first node from the List
       //  get the first free neighbor PE to restart the depth first
12.    FirstPE ← Get the first free neighbor PE from PE_i
      // start new path
13.    Path ← {};
       // traversal in depth first order
14.    DepthFirstPR(X, Path, FirstPE);
15. end while
16. return from algorithm

Procedure: MapToGrid(NodeList L, PE P) // Function map to grid
17.  v ← Get and remove first DFG node from the list L
18.  Grid[P] ← v; // Place
19.  next ← Get a free neighbor of P or if not possible a free PE near P
20.  MapToGrid(L, next);
21. return
```

Fig. 9. Placement and local (neighborhood connections) routing algorithm.

Suppose the DFG and the 2-D CGRA both depicted in Fig. 10(a and b). A trace table for the P&R algorithm is shown in Fig. 11. Let us consider the node *n1* as the DFG start node and *pe1* the first PE from the CGRA. Several paths are created during a depth-first traversal. A path is represented by a list of connected DFG nodes. Lines 1–2 and 8 of the P&R algorithm (shown in Fig. 9) perform a recursive depth first search on one descendent of each source node to build a path. Let us consider the path built for the nodes *n1* to *n8* shown in the trace table (lines 1–5). When an end node is found, as is the case with node *n8*, the *MapToGrid* function is called to place all nodes in the path (i.e., *n1*, *n3*, *n5*, *n7*, and *n8*), as shown in Fig. 10(c). The function *MapToGrid* performs a depth-first on the graph representing the CGRA, while the *DepthFirstPR* performs a depth-first on the DFG to be mapped. When the function returns from the recursive call, lines 9–15 in the P&R algorithm (Fig. 9), a depth first traversal is performed for the remaining unplaced DFG nodes. For the current path, only the node *n1* has more than one descendent, and the depth first traversal will continue on node *n4*, as shown in the trace table. The depth first on CGRA graph should be synchronized to the DFG, so the *firstPE* will be the *pe2* which is adjacent to *pe1*, where the node *n1* has been placed (line 12). A new path will be created (line 13), to place node in the *pe2*, selected in line 12. The placement is shown in Fig. 10(d). Finally, all nodes from *n1* have been placed, and the depth-first continues on nodes *n2* to *n6*, as shown in Fig. 10(e).

As this example illustrates, all DFG edges are visited just once. For each visited edge of the DFG, the algorithm tries to associate an interconnect resource in the CGRA. Considering $|E|$ the number of edges in the DFG, and $|V|$ the number of nodes, the computational complexity of the proposed algorithm is $O(|E| + |V|)$, or $O(|V|^2)$ considering $|E| \leqslant |V|^2$.

However, as the proposed architecture has limited routing resources, it is unlikely that the routing step is completed using this simple P&R. In the example illustrated in Fig. 10, the depth-first P&R algorithm completes with three unrouted DFG edges (see the edges $n2 \rightarrow n4$, $n6 \rightarrow n7$, and $n4 \rightarrow n8$ in Fig. 10(a) and Fig. 10(e), respectively). Here comes the next step that tries to route those remaining DFG edges using the interconnect resources given by the MIN(s) included in the CGRA, i.e., after the depth-first traversal previously explained, a second routing phase is performed. This latter routing stage is responsible to route DFG edges using the routing resources provided by the MIN(s). Fig. 12 shows the algorithm for this second routing phase. The routing algorithm scans the MIN network lines from the input to the output to verify if all lines are free and the routing is feasible. The algorithm is called for each unrouted connection in the previous P&R phase.

Before continuing with the example illustrated in Fig. 10, we explain through two examples how the routing is performed for the Omega MINs. For simplicity, we assume here a 4×4 Omega MIN without extra levels. First, the binary representations of the input

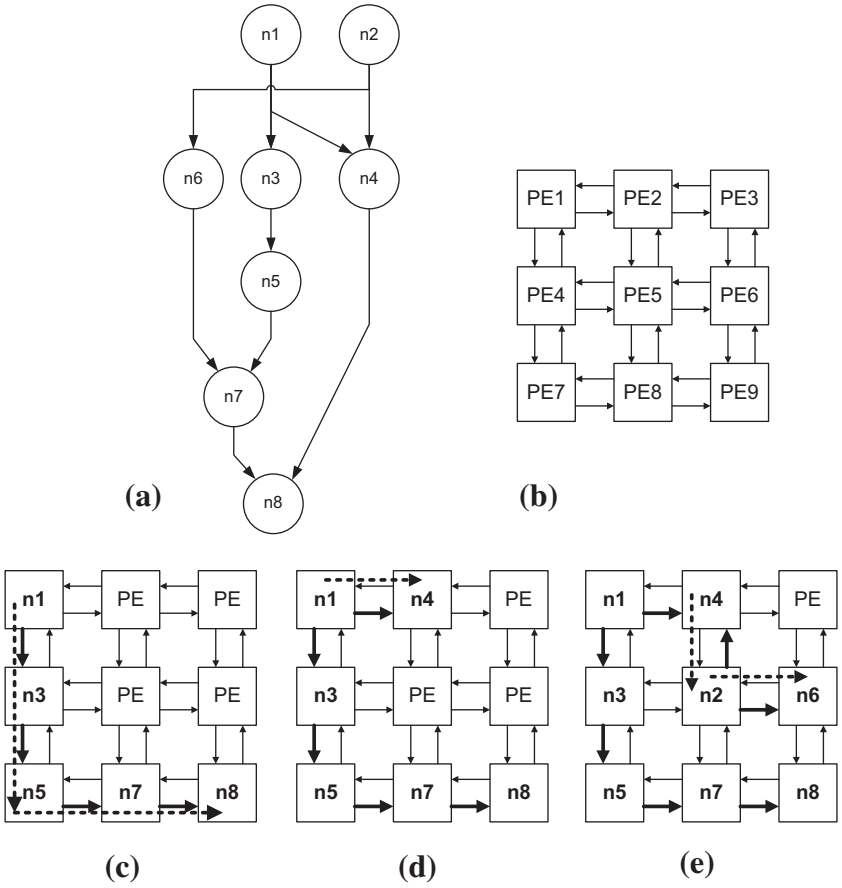

Fig. 10. First placement and routing stage: (a) input dataflow graph (DFG); (b) target CGRA example; (c) 1st depth-first path; (d) 2nd depth-first path; and (e) last depth-first path.

	Current	List	Path	FirstPE
1	n1	**n3**,n4	n1	pe1
2	n3	**n5**	n1,n3	pe1
3	n5	**n7**	n1,n3,n5	pe1
4	n7	**n8**	n1,n3,n5,n7	pe1
5	n8	**-**	**n1,n3,n5,n7,n8**	**pe1**
6	n7	-	-	pe1
7	n5	-	-	pe1
8	n3	-	-	pe1
9	n1	**n4**		pe1
10	n4	-	n4	pe2
11	n2	n6	n2	pe5
12	n6	-	n2,n6	pe5

P	Node list	next	MAP
pe1	n1,n3,n5,n7,n8	pe4,pe2	n1
pe4	n3,n5,n7,n8	pe7,pe5	n3
pe7	n5,n7,n8	pe8	n5
pe8	n7,n8	pe9	n7
pe9	n8	pe6	n8
pe2	n4	pe3,pe5	n4
pe5	n2,n6	pe6	n2
pe6	n6	pe8	n6

Fig. 11. Trace of the P&R algorithm for a simple example.

and output ports of the MIN to perform a specific connection are concatenated in a single word (w). If we intend to create a $3 \rightarrow 1$ connection – input 3 (1 1) and output 1 (0 1) – w equals 1 1 0 1 as shown in Fig. 13. The routing path is unique and is formed by shifting step by step the concatenation word (w) by one bit each step and by taking z, the $\text{Log}_2(N)$ (2 in this example) most significant bits for an $N \times N$ Omega MIN, represented underlined and in bold in Fig. 13. After each shift (step), z represents the line address of each intermediate stage. The shifting process is shown in Fig. 13(a and b) for the connections $3 \rightarrow 1$ and $0 \rightarrow 2$. If after these two connections one intends to add the connection $2 \rightarrow 3$ a collision occurs, as shown in Fig. 13(c). After the first step z equals 0 1 (1 **0 1** 1), which corresponds to a line address already in use by the connection $0 \rightarrow 2$.

Adding an extra level to the Omega MIN increases the length of w, and a connection has more than one option to route the input to the output. In the case of extra levels, the path is determined by the input followed by the extra levels and by the output. Let us consider again the connections $3 \rightarrow 1$, $0 \rightarrow 2$, and $2 \rightarrow 3$. Let us suppose a 4×4 Omega MIN again, but now with one extra level as shown in Fig. 14. The concatenation word w is 1 1 X 0 1 for the connection $3 \rightarrow 1$, where X is the extra level. There are two options, $X = 0$ or $X = 1$. For the case where $X = 0$, the path is shown in Fig. 14(a). Fig. 14(b) displays the path for the connection $0 \rightarrow 2$ when consid-

Algorithm 2. Second stage of P&R

Parameters:

- N inputs and N outputs; M = Number of Stages = $Log_2(N) + K$
- K = number of extra stages; L = 2^K
- Bitmask = length $Log_2(N)$, e.g., if N=8 then Bitmask = 111

```
RoutingOmega(DFGnode source, DFGnode target)
1.  F1: for Inter ← 0 to L-1 do // test all possible values for extra stages
2.      path ← (source << M) OR (Inter << Log2(N)) OR target
3.      found ← true
4.      for j ← 0 to Log2(N)+K-1 do // from stage 0 to stage Log2(N)+K-1
5.          i ← path >> (Log2(N)+K+1-j) AND bitmask
6.          found ← found AND freeline[i][j] // verify if line connection is free
7.          If found then
8.              line[j] ← i
9.          else continue F1
10.         end if
11.     end for
12.     if found then
13.       freeline[line[0:Log2(N)+K-1]][j] ← false  // set line as occupied
14.       return true
15.     end if
16. end for
17. return false // path conflict
```

Fig. 12. Algorithm to perform routing between PEs using the Omega Network (global connections routing algorithm).

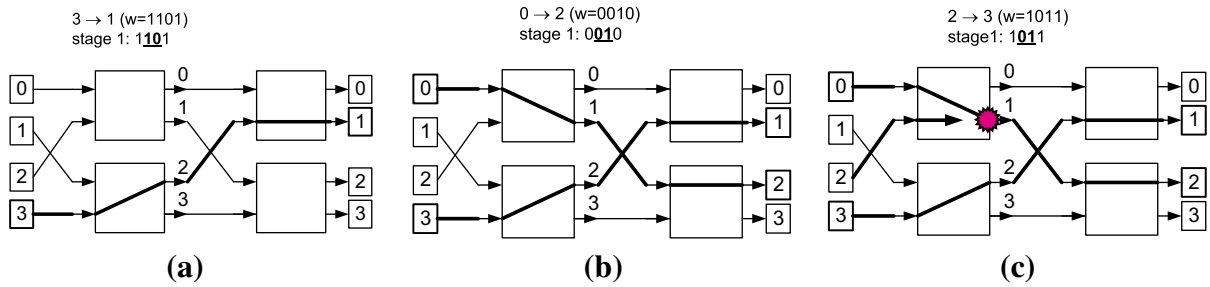

Fig. 13. Example showing the steps of the routing algorithm when considering a 2 × 2, two stages Omega MIN: (a) routing 3 → 1; (b) routing 0 → 2; and (c) trying to route 2 → 3.

ering $X = 0$, and therefore w equals 0 0 0 1 0. The connection 2 → 3 generates $w = 1\ 0\ X\ 1\ 1$. If $X = 0$, a collision occurs at stage 1, line 0, where the concatenation words for 0 → 2 ($w = 0\ \mathbf{0\ 0}\ 1\ 0$) and 2 → 3 ($w = 1\ \mathbf{0\ 0}\ 1\ 1$) need the same line address at stage 1. However, if we select $X = 1$ for the connection 2 → 3, no collision occurs and this connection is routed, as shown in Fig. 14(c).

Fig. 15 presents the same example of Fig. 10 but now including the routing through an Omega MIN (for simplicity we do not show all the connections from/to the PEs to/from the MIN). The first stage of P&R based on depth-first traversal is applied using the input DFG. The bold lines in Fig. 15(a) show the DFG edges mapped onto adjacent PEs during the local P&R. Fig. 15(b) shows a 3 × 3 Grid CGRA as well as the local connections (also in bold) assigned to the edges of the DFG in bold (Fig. 15(a)). The DFG edges *n6* → *n7* and *n4* → *n8* have not been assigned to CGRA interconnections during the first P&R step (see dash lines in Fig. 15(a)). The second P&R stage previously explained consists on the MIN routing. This second stage is able to successfully route the two edges, not routed using the local interconnect resources of the CGRA, through the Omega Network interconnect resources.

Given E the set of DFG edges to route through the MIN, the number of terminals, N, of the MIN, and the number of extra stages, K, the worst case computational complexity of the MIN routing stage performed using the algorithm illustrated in Fig. 12 is $O(|E| \times 2^K \times \log_2(N + K))$ and since $|E| \leqslant N$, we have $O(N \times 2^K \times \log_2(N + K))$. As we use a small number of extra stages (e.g., $K = 2$ for the benchmarks we use in the experimental results presented in this paper) we have $O(N \times \log_2(N))$ as the worst case computational complexity for the routing algorithm presented in Fig. 12, which is the same as the one presented in the literature for the routing of MINs [33,34].

When considering the two stages of our P&R we have as overall worst case computational complexity $O(|V| + |E| + N \times \text{Log}_2(N))$, or $O(|V|^2 + N \times \text{Log}_2(N))$ considering the worst case where $|E| \leqslant |V|^2$. Since N equals the number of PEs of the CGRA and is greater or equal $|V|$, the worst case complexity of the two stages of our P&R is $O(N^2)$.

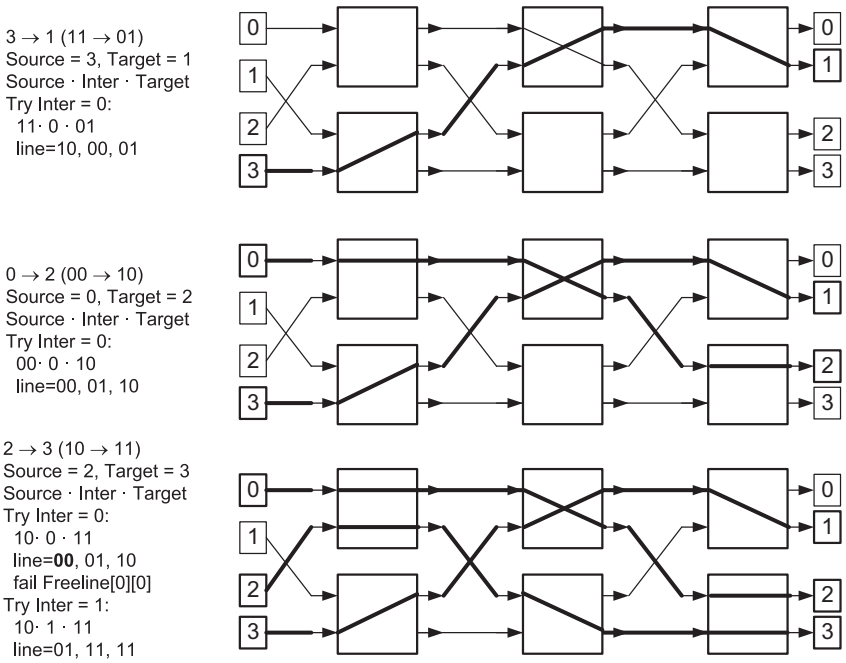

Fig. 14. Example showing the steps of the routing algorithm when considering a 2 × 2 Omega MIN with three stages (i.e., with one extra stage) and the connections 3 → 1, 0 → 1, and 2 → 3.

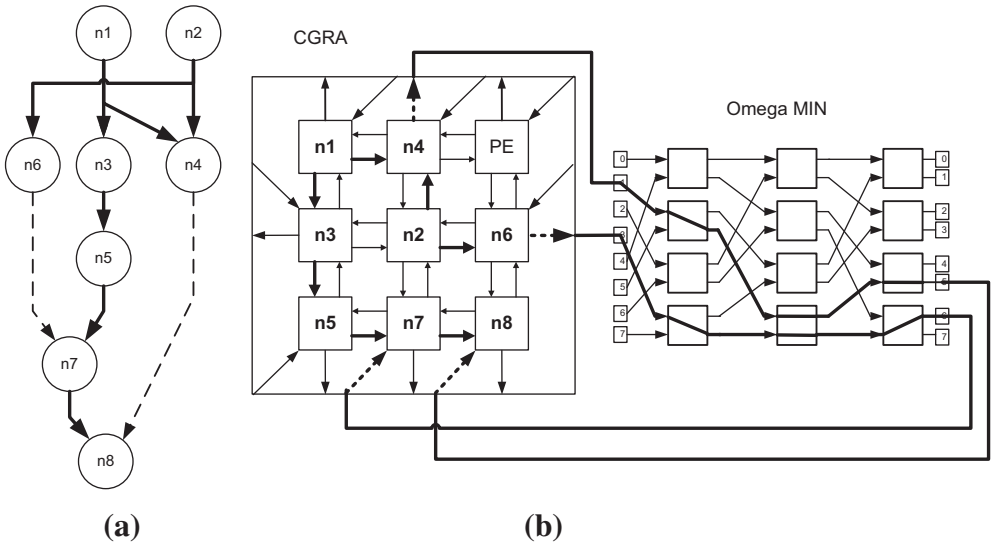

Fig. 15. Connections routed through the Omega MIN: (a) DFG with edges not routed in the grid represented as dotted lines; (b) grid with the Omega MIN and the mapping of the DFG (the two dotted lines edges are now routed through the Omega MIN).

4. Experimental results

In order to evaluate the approach presented in this paper, we have used two benchmark sets consisting in a total of 27 data-flow graphs (DFGs). The first set is composed by DFGs representing the benchmarks previously used in [39]: treefir64, Cplx8, filterRGB, DCT, fir64, Fir16, filterRGBpaeth, and snn. The second set is composed by 19 DFGs from [40], a repository with DFGs carefully selected from over 1400 DFGs obtained from applications in the MediaBench benchmark suite. The DFGs used range in complexity from 11 to 359 nodes, and from 8 to 380 edges (see Table 1).

We evaluated the success of P&R when targeting 2-D Grid CGRAs, with Torus topology and 8/8 input/outputs per PE, vs. CGRAs with Omega Networks and with 4/4 input/outputs per PE with the output of the FU directly connected to every PE output and to the Omega Networks (referred here as ***simple Grid***). For this study, we considered CGRAs with size from 16 (4 × 4) to 361 (19 × 19) PEs.

For the P&R to 2-D Grid-based CGRAs we employed the placement algorithm proposed in [39] enhanced with a version of the pathfinder routing algorithm for the routing stage (referred herein as "Depth-First + Pathfinder", DFP). For the CGRA with Omega Networks, we used the One-Step P&R (OPR) algorithm presented in this paper.

The DFP is able to successfully place and route all the benchmarks considered in this paper when targeting a 2-D Grid CGRA, with Torus topology and at least 8/8 input/outputs per PE. These CGRA properties, however, lead to a high-cost PE. Using the same

Table 1
OPR P&R results for grid plus MIN architectures for the set of benchmarks used.

Benchmark	#Nodes	#Edges	#Unrouted edges (proposed architecture)							
			Grid	% Grid	Grid + one MIN			Grid + two MINs		
					$K = 0$	$K = 2$	$K = 4$	$K = 0$	$K = 2$	$K = 4$
treefir64	193	255	97	38.0	39	17	7	5	0	0
Cplx8	46	60	20	33.3	6	1	0	0	0	0
filterRGB	57	70	16	22.9	1	0	1	0	0	0
DCT	92	124	46	37.1	16	6	4	1	0	0
fir64	193	255	91	35.7	42	17	8	9	0	0
fir16	49	63	22	34.9	7	2	2	0	0	0
filterRGBpaeth	84	106	31	29.2	9	2	1	0	0	0
snn	253	299	80	26.8	31	13	1	8	0	0
fir1	44	43	21	48.8	6	0	0	0	0	0
arf	28	30	10	33.3	2	0	0	0	0	0
jpeg_idct_ifast	170	210	71	33.8	27	13	6	6	0	0
smooth_color_z	197	196	67	34.2	23	6	2	4	0	0
collapse_pyr	72	89	31	34.8	11	4	3	1	0	0
horner_bezier_s	18	16	2	12.5	1	0	0	0	0	0
jpeg_fdct_islow	175	210	69	32.9	22	7	1	6	0	0
matmul	117	124	39	31.5	11	4	1	3	0	0
fir2	40	39	14	35.9	2	0	0	0	0	0
motion_vectors	32	29	10	34.5	2	0	0	0	0	0
cosine1	66	76	32	42.1	7	3	3	0	0	0
idctcol	186	236	78	33.1	31	16	6	2	0	0
interpolate_aux	108	104	44	42.3	13	3	1	3	0	0
feedback_points	54	51	18	35.3	4	0	0	1	0	0
ewf	42	55	16	29.1	3	0	0	0	0	0
write_bmp_header	111	93	25	26.9	6	0	0	0	0	0
h2v2_smooth	54	55	18	32.7	8	1	0	2	0	0
invert_matrix_gen	359	380	109	28.7	39	10	5	6	0	0
hal	11	8	1	12.5	0	0	0	0	0	0
Average	105.6	121.3	39.9	32.3	13.7	4.6	1.9	2.1	0.0	0.0
Total	2851	3276	1078		369	125	52	57	0	0
% Unrouted edges			32.9		11.3	3.8	1.6	1.7	0.0	0.0

algorithm to target the simple Grid CGRA many connections (edges in the DFGs) are not routed. This is due to the low connectivity of the simple Grid and the simplicity of the simple greedy placement approach used by DFP.

Table 1 shows the P&R results considering the P&R with the "One-Step P&R" (OPR) algorithm. When targeting the simple Grid, 33% of the total DFG edges are not routed by the neighborhood PE interconnections (see "Grid" column in Table 1). These results are similar to the ones achieved if we use the DFP approach. However, when we add an Omega MIN to the 2-D Grid CGRA, the percentage of unrouted edges decreases from 32.9% to 11%. With extra MIN stages, the percentage of unrouted DFG edges reduces to 4% and to 2%, when using two ($K = 2$) and four ($K = 4$) extra stages, respectively. Using the Omega MIN with four extra stages, 15 benchmarks (out of 27) were successfully P&R as shown in Table 1. Each routing congestion in a MIN has two possible causes. One happens when at least one switch is busy for all possible routing paths for an edge. In this case, extra switch levels can be added to solve the conflict. The second one happens when a node is the sink for at least two unrouted edges, and therefore it is not possible to route them because each node receives only one line from the MIN. When using additional MINs, we are diminishing such treats. The results for two MINs presented in column "Grid + Two MINs" in Table 1 show that all edges are routed when two MINs each one with two extra stages ($K = 2$) are added, and less than 2% of total edges are not routed when no extra stage is added.

It is important to note that even with MINs allowing rearrangeable routing, with OPR we still need two MINs to successfully route all the connections for the DFGs considered in this paper.

We now present execution time results for the P&R approaches considered in this paper. All the results represent average CPU time needed after running 100 times the P&R with each benchmark in a PC with the Ubuntu 9.04 Linux version, an Intel® Core™2 Duo CPU T7250 @ 2 GHz, with 2048 KB of cache, and 3 GB of system memory.

Table 2 presents the P&R execution time when using DFP and OPR and shows the speedups obtained by OPR vs. DFP. The CGRA considered for DFP is the one previously referred as ***simple Grid***. The CGRA considered for the OPR is the CGRA with two MINs. Using the CGRA architecture and the P&R approach proposed in this paper (OPR), we achieved speedups from about 10.83× to 353.85× (about 88.95× in average) over DFP, an already fast P&R algorithm. Those speedups are mainly consequence of the fact that with the CGRA extended with two MINs we are substituting a computationally intensive global routing using pathfinder with a much simpler routing in the MINs. The execution times achieved and the simplicity of the P&R algorithm are a strong evidence that the novel CGRA structure proposed in this paper can be an important support for runtime compilation.

The next experiment compares execution times of OPR and VPR [41,42] to P&R the benchmarks used in this paper. Note, however, that these comparisons are just to make strong evidence of the speed of the solution presented in this paper over well-known P&R approaches such as the one represented by VPR. It is not our intention to compare the efficiency of the results achieved by the two distinct approaches. VPR is not a P&R tool targeting CGRAs, but mainly island-style FPGAs. Comparisons to P&R tools used for CGRAs would have been a more interesting approach if tools were publicly available.

The OPR targeted a CGRA with two Omega MINs and VPR target an island-style CGRA. The execution times for VPR were measured for the fastest option (*-fast-timing_analysis off*). Fig. 16 shows results that indicate the execution time speedup of OPR over VPR. On average, VPR was 14.3× slower. The highest speedup was achieved with *jpeg_fdct_islow* (32.1×) and the lowest with *fir16* (3.2×). Note, however, that VPR was run as an executable while

Table 2
Execution time comparison when performing P&R with the OPR vs. DFP.

Benchmark	CPU execution time (ms)				Speedup (OPR vs. DFP)
	DFP	OPR			
		MIN routing	Local P&R	Total	
treefir64	1663	15.00	11.00	26.00	63.96
Cplx8	176	0.20	3.65	3.85	45.71
filterRGB	76	0.40	2.65	3.05	24.92
DCT	261	2.40	8.50	10.90	23.94
fir64	1377	4.40	19.30	23.70	58.10
fir16	164	1.15	5.10	6.25	26.24
filterRGBpaeth	126	2.45	2.60	5.05	24.95
snn	1143	6.25	14.40	20.65	55.35
fir1	68	0.10	2.20	2.30	29.57
ari	26	1.90	0.50	2.40	10.83
jpeg_idct_ifast	612	1.55	11.60	13.15	46.54
smooth_color_z	802	13.10	9.75	22.85	35.10
collapse_pyr	77	1.50	2.40	3.90	19.74
horner_bezier_surf	78	0.05	0.25	0.30	260.00
jpeg_fdct_islow	655	2.50	2.80	5.30	123.58
matmul	228	0.45	3.80	4.25	53.65
fir2	34	2.25	0.25	2.50	13.60
motion_vectors	71	0.30	0.35	0.65	109.23
cosine1	153	0.50	1.05	1.55	98.71
idctcol	909	2.20	8.65	10.85	83.78
interpolate_aux	920	0.85	1.75	2.60	353.85
feedback_points	96	0.40	0.95	1.35	71.11
ewf	96	0.70	1.40	2.10	45.71
write_bmp_header	198	1.00	2.75	3.75	52.80
h2v2_smooth	72	0.35	1.50	1.85	38.92
invert_matrix_general	4965	3.05	13.40	16.45	301.82
hal	33	0.05	0.05	0.10	330.00
Average	558.48	2.41	4.91	7.32	88.95

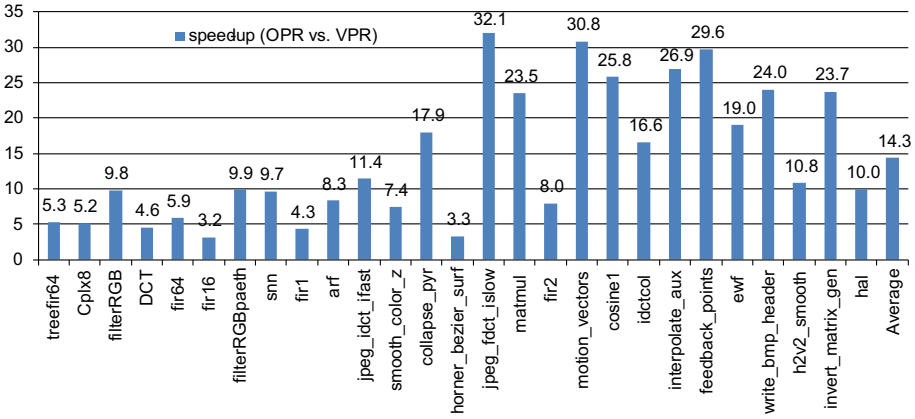

Fig. 16. Execution time comparison when performing P&R with the OPR vs. the use of VPR.

OPR is a Java implementation run with the JVM. These speedups make strong evidence of the importance of the proposed CGR plus the P&R solution.

We now analyze the increase in the critical path of the benchmarks after P&R to the CGRA + MINs. The analysis considers different ratios of PE:MIN latencies (1:1 and 1:2) and compares the increase in the critical path latencies obtained for each example using a PE:MIN latency ratio of 1:0, which is equivalent to the critical paths obtained for the DFGs when assuming one clock cycle of latency for each operation and negligible interconnect delays. In all of the comparisons we assume the delays of the local interconnections are part of the PE latency.

We consider for these experiments three versions of the OPR P&R algorithm described in this paper. The first version is the base OPR, which does not take into account the critical path of the DFG when performing P&R. The second version is an extension to OPR that, during the depth first based approach, prioritizes the placement of the nodes in the critical path and we refer to it as partial critical path aware P&R (OPR-P). The third option referred to as full critical path aware P&R (OPR-F) performs placement by considering first the nodes in the critical path instead of using a depth first approach.

Fig. 17 illustrates the results obtained considering the previous conditions and for the three P&R algorithms. For a ratio 1:1 (i.e., same latencies for PE and MIN) we obtained on average increase of the critical path latency by 30% (maximum of 71% and a minimum of 10%), 27% (maximum of 57% and a minimum of 2%) and 16% (maximum of 44% and a minimum of 0%), for the OPR, OPR-P, and OPR-F, respectively. When considering the ratio 1:2 (i.e., MIN with twice the latency of the PE), we obtained on average in-

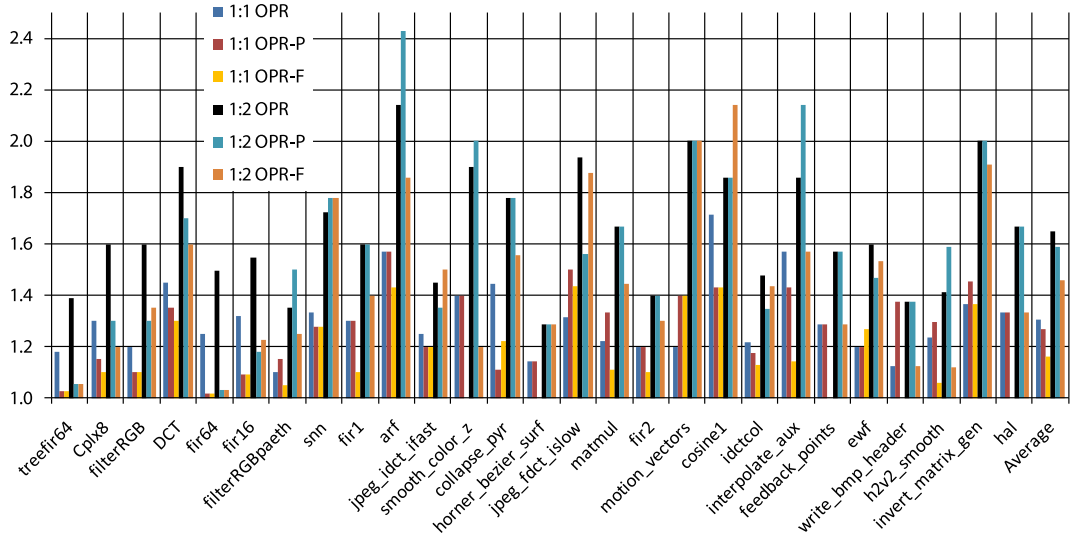

Fig. 17. Increase of the critical path latency over the critical path of the DFGs considering one clock cycle for each PE and zero-delay neighborhood interconnections for different MIN latencies (one and two clock cycles).

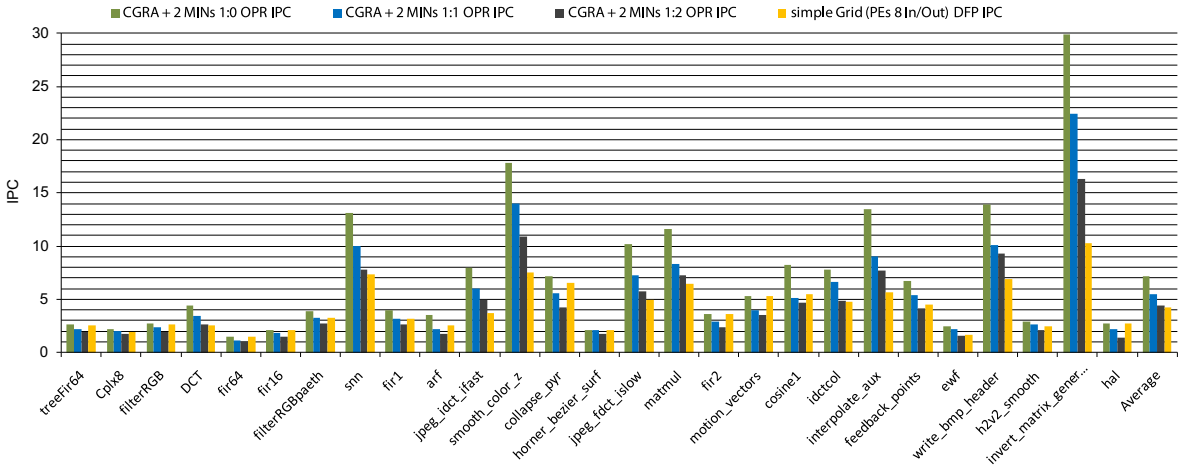

Fig. 18. IPC achieved when considering *simple Grid* (and DFP) and *Grid* + 2 *MINs* (and OPR) for three options of latency for the MINs (0, 1, and 2).

crease of the critical path latency by 65% (maximum of 114% and a minimum of 28.6%), 59% (maximum of 143% and a minimum of 3%) and 45.8% (maximum of 114% and a minimum of 3%), for the OPR, OPR-P, and OPR-F, respectively. Note that the reductions in the critical path latencies achieved by OPR-P and OPR-F over OPR were on average 2.7 and 12% for 1:1, respectively, and 3.8 and 13.3% for 1:2.

We now consider the IPC (instructions per cycle) obtained for each CGRA considered in this paper. The main objective is to evaluate if the use of CGRA + two MINs and its fast P&R approach conducts to lower IPC than when using *simple Grid* and a more computationally intensive P&R. Fig. 18 illustrates the IPCs obtained for the 27 benchmarks. The maximum IPC was 29.92 (1:0), 22.44 (1:1), 16.32 (1:2) for the CGRA + two MINs (with OPR version), and 10.26 for the *simple Grid*. The minimum IPC was 2.13 (1:0), 1.81 (1:1), 1.38 (1:2) for the CGRA + two MINs (with OPR version), and 1.68 for the *simple Grid*. On average the IPC was 7.19 (1:0), 5.45 (1:1), 4.38 (1:2) for the CGRA + two MINs (with OPR version), and 4.23 for the *simple Grid*. These results show that the CGRA + two MINs achieves on average similar or better performance than the use of *simple Grid*. The resultant IPCs are dependent on the ratio used between PE latency and MIN (PE latency:MIN latency). The results of the CGRA + two MINs are, however, improved if we use the OPR-F P&R. In this case, as the nodes in the critical path are placed firstly and their connections may use neighbor PE interconnect resources, improvements of 10.79% and 15.36% were measured for 1:1 and 1:2, being the case with 1:0 with similar results as expected.

A question that may arise at the moment is related to the area and delay overhead when using Omega Networks. As we are interested in CGRAs to define a computational layer implemented in fine-grained reconfigurable fabrics, we show here implementations of CGRAs in a Xilinx Virtex-5 FPGA (xc5vlx330ff1760-2). We consider in these experiments PEs with 32-bit FUs implementing arithmetic operations (addition, multiplication, subtraction, negation, comparisons), shifts, and bitwise operations (AND, OR, XOR, and NOT). We also consider that all outputs of the PEs are registered.

Fig. 19 presents the number of FPGA resources (#LUTs) and maximum delays obtained for different sizes of Omega Networks dealing with 32-bit input/outputs. The area cost in LUTs is a function of $N \times \mathrm{Log}_2(N)$. The delay increases by about 0.5 ns for each

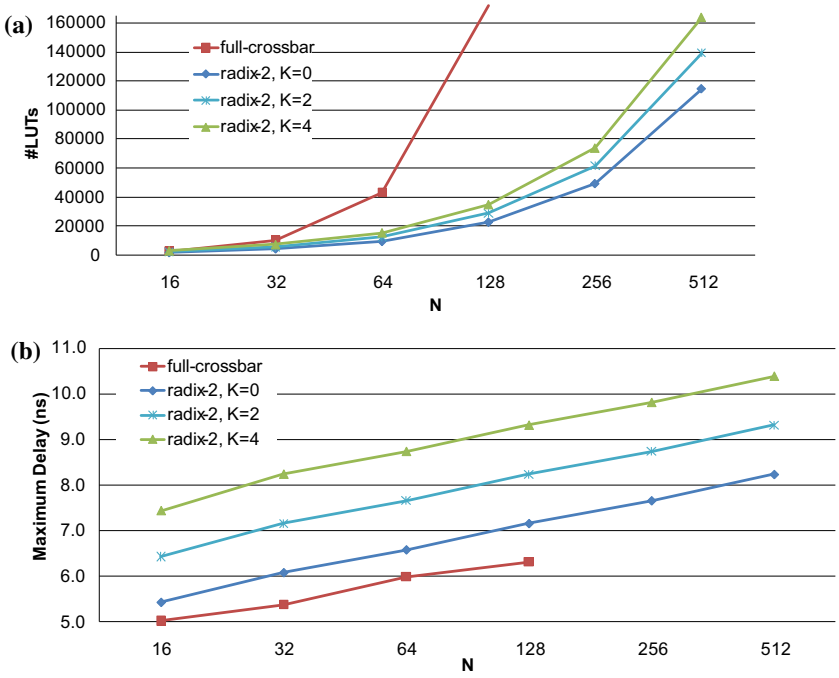

Fig. 19. Results for 32-bit, radix-2, Omega Networks considering different number of terminals (N): (a) area in terms of #LUTs and (b) maximum delay.

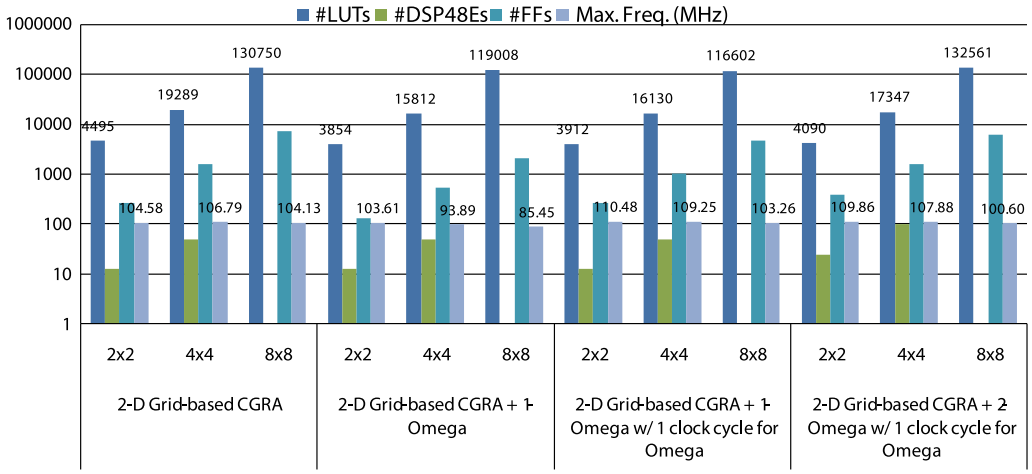

Fig. 20. FPGA resources used and maximum frequencies for a number of CGRAs (percentages of used resources are given; for the 8 × 8 arrays the DSP48 resources were not used).

additional input/output of the network. Note that even an Omega Network with 128 input/outputs only uses about 11% of the total LUTs available in the FPGA considered.

We show in Fig. 20 the number of #LUTs used and the maximum clock frequencies achieved for simple 2-D Grid-based CGRAs and for CGRAs with Omega Networks proposed in this paper. The Grid with simple routing resources (i.e., with PEs with a small number of neighborhood connections) has a very low cost, but suffers from congestions and may need long P&R execution times. The Grid plus one MIN has an area cost below the Grid with 4/4 inputs/outputs PEs with routing resources. With respect to maximum clock frequencies, they decrease for the CGRA with MIN when the MIN is not registered (MIN delay adds to the maximum delay of the PE). In this case, the clock frequency decreases about 12% and 18% for the 4 × 4 and 8 × 8 CGRA with 1-MIN, respectively, when compared to the ones obtained for the original CGRA (these frequencies correspond also to the performance decrease as the latencies of the applications in the CGRA do not change). When using MINs with registered outputs, we achieve maximum clock frequencies similar to the baseline 2-D Grid-based CGRA without MINs.

In this case, we are including interconnect resources that route in a clock cycle and thus the latencies may change, especially with P&R algorithms not aware of the critical path of the DFGs (i.e., without assigning local interconnects to edges of the critical path). In the improbably worst case assumption, this would increase 2× the latencies of the applications in the CGRA. However, the previous study presented in Fig. 17 shows that an OPR P&R algorithm aware of the critical path achieves an average performance decrease of about 13%.

The results presented in this paper show that CGRAs extended with Omega Networks are promising solutions, especially in systems where one needs to avoid long P&R runtimes, e.g., in systems needing runtime P&R.

5. Related work

There have been many CGRAs proposed [8–17]. Most CGRAs use a 2D mesh-based interconnect topology with nearest neighbor connectivity and include other layers of connectivity as 1-hop or 2-hop. To handle more effectively with irregular communication patterns, some CGRAs use buses that allow to connect one PE from a set of possible PEs to one or more PEs (called express lanes in the MorphoSys [13]). The MorphoSys [13] uses horizontal and vertical buses besides the local connections between PEs. The XPP [15] uses horizontal buses between rows of PEs and each bus allows the connection between a source PE and multiple destinations, all in adjacent rows. Each PE has specific forward and backward registers to allow connections over the rows of PEs. The ADRES [14] uses a 2D mesh-based interconnect topology extended with non-neighborhood interconnect resources (e.g., 1-hop interconnections and buses). Bouwens et al. [43] presented a study about the impact on performance, energy, and power for different options for the ADRES CGRA. They concluded that buses do not give substantial advantages and they suggested the use of local and 1-hop interconnect resources. All those approaches use interconnect resources and topologies based on requirements that include the enhancement of routability, energy savings, power dissipation reductions, and better performance, but without bearing in mind the speedup of the placement and routing phase. All of them need complex placement and routing steps that may prevent their acceptance in the context of dynamic compilation. Note, however, that the use of small CGRA (e.g., 2×2, 4×4) diminishes the placement and routing efforts and in the context of dynamic compilation may make viable such CGRAs.

The mapping approaches proposed for ADRES [14] and SPR [44] CGRAs consider loop pipelining when mapping loops. Loop pipelining using modulo scheduling is performed in a stage tightly coupled to the P&R stages. For instance, in SPR [44] each unsuccessful routing conducts to another iteration of the mapping. In each mapping iteration, the II (initiation interval) cycle is increased by one. Iteration over the mapping stages repeats a computationally demanding process and the use of MINs in these CGRAs may diminish the number of iterations and may allow lower IIs conducting to a more efficient mapping process.

The approach presented in this paper complements the local interconnect resources in a 2D mesh-based CGRA with global interconnect resources based on multistage interconnection networks. During the 90's, multistage interconnection networks have been proposed in the context of system emulation platforms consisting of multiple FPGAs [45–47]. In those approaches, one or more FPGAs were used as interconnection devices. They used for interconnection networks: Clos networks [45], Folded-Clos [46], and partial cross-bars [47].

Recently, a multiprocessors system on a chip implemented in an FPGA and using an Omega Network has been presented in [48]. The authors considered up to eight mini-MIPS 32 bits processors, local instruction caches and shared data caches. The processors and the shared data caches used an Omega Network as packet-switching network to communicate data. The Omega Network has been shown as an interesting approach as it requires a smaller area compared to the area needed for crossbar networks.

Our work differs from those approaches as we are focusing on CGRA extensions based on multistage interconnection networks in order to enrich the interconnect possibilities of typical CGRAs.

Most P&R algorithms are based on simulated annealing (SA) for placement and pathfinder for routing [5]. Albeit the inherent lower number of resources to manage by P&R when targeting CGRAs, the P&R execution times for CGRAs are still quite high, e.g., a modulo scheduling algorithm based on SA [14] spent around 100 s for DFGs with about 200 nodes. Recently, [49] presented a CGRA based on a 2-D stripe model. However, even using a greedy P&R, the reported execution times are also in the order of seconds.

Our previous work addressed a polynomial placement algorithm [39] for 2-D Grid-based CGRAs. However, it needs that the target architecture uses rich local interconnection resources and 1-hop interconnections to achieve successful P&R.

Recently, some authors focused on runtime P&R in the context of just-in-time (JIT) compilation. The most notorious work is the one associated with the Warp processors [50]. They simplify P&R by using a greedy placement algorithm and by reducing the interconnect resources of their fine-grained reconfigurable logic array. However, the P&R execution times reported are in the order of seconds when considering examples with similar complexity as the ones presented in this paper.

Recognizing that we need both new P&R algorithms and CGRA specific hardware support to achieve fast P&R (envisioning dynamic compilation), we propose the use of global multistage interconnect networks (MINs) in 2-D Grid-based CGRAs. Our approach is different from the previous ones in a number of aspects. The support given by MINs allows a P&R algorithm with polynomial complexity, flexible, and achieving execution times in the order of milliseconds, even for large DFGs (around 300 nodes), which give a speedup about a order of magnitude compared to the execution times reported in literature. Note, however, that the approach presented in this paper can be used to extend other CGRAs (e.g., ADRES [14] and SPR [44]), especially when large CGRAs are needed.

6. Conclusions

This paper proposed a coarse-grained reconfigurable array (CGRA) architecture extended with global multistage interconnection networks. The proposed solution aims at simplifying and making faster the placement and routing stage, an important aspect when envisioning the runtime mapping of parts of applications to CGRAs acting as hardware accelerators.

The main idea is to augment local routing resources between neighborhood processing elements with global routing allowed by multistage interconnection networks. The multistage interconnection networks easier the routing by giving a richer interconnect topology which permits to connect directly processing elements independently of their position in the CGRA. By doing so, a simple and fast placement and routing algorithm can be used as was fully demonstrated in this paper. One interesting result of this research is the fact that the CGRA proposed takes advantage of the local connections between PEs and only delegates to the global multistage interconnect networks the connections not successfully routed locally (i.e., between neighborhood CGRA processing elements).

The proposed CGRA architecture allows a polynomial placement and routing. This papers presented extensive experiments to fully evaluate the CGRA and the placement and routing algorithms. The results achieved are very promising. The new architecture leads to a very fast placement and routing (on average in 16× less time than when using a similar architecture, but without global interconnection networks), with acceptable area overhead, and low performance degradation (less than 18% for an 8×8 array).

We showed in this paper that even a placement and routing algorithm unaware to the critical path may achieve good results. This is an important aspect, as it allows P&R to be done partially,

e.g., with a frame of instructions and without building its dataflow graph.

There are a number of opportunities to continue this work. One may consider the use of multistage interconnect networks by clusters of PEs and will study how that could affect the complexity of the P&R and the area and performance of clustered CGRAs. Another research avenue may be to extend the CGRA with hardware support for runtime placement and routing. Short term research work will address retiming and pipelining while performing P&R, an important aspect when addressing some CGRAs.

Acknowledgements

This work was partially funded by Grices/CNPq, within a bilateral Portugal/Brazil project. João Cardoso acknowledges the support of FCT, Portugal, under Grant PTDC/EEA-ELC/70272/2006.

References

[1] Rainer Hartenstein, A decade of reconfigurable computing: a visionary retrospective, in: Conference on Design, Automation and Test in Europe (DATE), IEEE Press, Munich, Germany, 2001.
[2] Maya Gokhale, P. Graham, Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays, first ed., Springer, 2005.
[3] S. Hauck, A. DeHon, Reconfigurable Computing: The Theory and Practice of FPGA-based Computation, Morgan Kaufmann, 2007.
[4] Scott Hauck, The roles of FPGAs in reprogrammable systems, Proceedings of the IEEE 86 (4) (1998) 615–638.
[5] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, P.K.Y. Cheung, Reconfigurable computing: architectures and design methods, IEE Proceedings – Computers and Digital Techniques 152 (2) (2005) 193–207.
[6] Alex K. Jones, Raymond Hoare, Dara Kusic, Joshua Fazekas, J. Foster, An FPGA-based VLIW processor with custom hardware execution, in: ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays (FPGA'05), Monterey, California, USA, ACM, New York, NY, USA, 2005.
[7] B. Buyukkurt, W.A. Najjar, Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs, in: International Conference on Field Programmable Logic and Applications (FPL'08), 2008.
[8] S. Shukla, N.W. Bergmann, J. Becker, QUKU: a FPGA based flexible coarse grain architecture design paradigm using process networks, in: IEEE International Parallel and Distributed Processing Symposium (IPDPS'07), 2007.
[9] Christophe Wolinski, Maya Gokhale, K. McCabe, A polymorphous computing fabric, IEEE Micro 22 (5) (2002) 56–68.
[10] Mladen Berekovic, Frank Bouwens, Tom Aa, Diederik Verkest, Interconnect power analysis for a coarse-grained reconfigurable array processor, in: Eighteenth International Workshop on Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation (PATMOS'08), September 10–12, 2008 (Revised Selected Papers 2009, Springer-Verlag, Berlin, Heidelberg, Lisbon, Portugal, pp. 449–457).
[11] Reiner W. Hartenstein, Juergen Becker, Reiner Kress, H. Reinig, High-performance computing using a reconfigurable accelerator, Concurrency – Practice and Experience 8 (6) (1996) 429–443.
[12] Carl Ebeling, Darren C. Cronquist, P. Franklin, RaPiD – reconfigurable pipelined datapath, in: Sixth International Workshop on Field-Programmable Logic and Applications (FPL'95), Springer-Verlag, 1995.
[13] Hartej Singh, Ming-Hau Lee, Guangming Lu, Nader Bagherzadeh, Fadi J. Kurdahi, E.M.C. Filho, MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications, IEEE Transactions on Computers 49 (5) (2000) 465–481.
[14] Bingfeng Mei, Andy Lambrechts, Diederik Verkest, Jean-Yves Mignolet, R. Lauwereins, Architecture exploration for a reconfigurable architecture template, IEEE Design and Test of Computers Magazine 22 (2) (2005) 90–101.
[15] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, M. Weinhardt, PACT XPP: a self-reconfigurable data processing architecture, The Journal of Supercomputing 26(2) (2003) 167–184.
[16] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, R.R. Taylor, PipeRench: a reconfigurable architecture and compiler, Computer 33 (4) (2000) 70–77.
[17] F. Thoma, M. Kuhnle, P. Bonnot, E.M. Panainte, K. Bertels, S. Goller, A. Schneider, S. Guyetant, E. Schuler, K.D. Muller-Glaser, J. Becker, MORPHEUS: heterogeneous reconfigurable computing, in: International Conference on Field Programmable Logic and Applications (FPL'07), 2007.
[18] F. Hannig, H. Dutta, J. Teich, Parallelization approaches for hardware accelerators – loop unrolling versus loop partitioning, in: Proceedings of the 22nd International Conference on Architecture of Computing Systems (ARCS), Delft, The Netherlands, 2009.
[19] J.M.P. Cardoso, On estimations for compiling software to FPGA-based systems, in: Sixteenth IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05), 2005.
[20] J. Siegel Howard, Interconnection Networks for Large-scale Parallel Processing: Theory and Case Studies, Lexington Books, 1985, p. 260.
[21] D.H. Lawrie, Access and alignment of data in an array processor, IEEE Transactions on Computers 24 (12) (1975) 1145–1155.
[22] Ricardo Ferreira, Alex Damiany, J. Vendramini, Tiago Teixeira, J.M.P. Cardoso, On simplifying placement and routing by extending coarse-grained reconfigurable arrays with Omega networks, in: Proceedings of the Fifth International Workshop on Reconfigurable Computing: Architectures, Tools and Applications (ARC'09), Springer-Verlag, Karlsruhe, Germany, 2009.
[23] N. Bansal, S. Gupta, N. Dutt, A. Nicolau, R. Gupta, Network topology exploration of mesh-based coarse-grain reconfigurable architectures, in: Design, Automation and Test in Europe Conference and Exhibition (DATE'04), 2004.
[24] Y. Yao-Ming, F. Tse-yun, On a class of rearrangeable networks, IEEE Transactions on Computers 41 (11) (1992) 1361–1379.
[25] S. Andresen, The looping algorithm extended to base 2^t rearrangeable switching networks, IEEE Transactions on Communications 25 (10) (1977) 1057–1063.
[26] Hu Qing, Shen Xiaojun, L. Weifa, Optimally routing LC permutations on k-extra-stage cube-type networks, IEEE Transactions on Computers 45 (1) (1996) 97–103.
[27] V.E. Benes, Mathematical Theory of Connecting Networks and Telephone Traffic, Academic Press, New York, USA, 1965.
[28] K.Y. Lee, A new Benes network control algorithm, IEEE Transactions on Computers 36 (6) (1987) 768–772.
[29] W.J. Dally, B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann Publishers, 2004.
[30] Marrakchi Zied, Mrabet Hayder, Amouri Emna, M. Habib, Efficient tree topology for FPGA interconnect network, in: Proceedings of the 18th ACM Great Lakes Symposium on VLSI, ACM, Orlando, Florida, USA, 2008.
[31] John Kim, William J. Dally, D. Abts, Flattened butterfly: a cost-efficient topology for high-radix networks, ACM SIGARCH Computer Architecture News 35 (2) (2007) 126–137.
[32] André DeHon, Randy Huang, J. Wawrzynek, Hardware-assisted fast routing, in: Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02), IEEE Computer Society, 2002.
[33] Abraham Waksman, A permutation network, Journal of the ACM (JACM) 15 (1) (1968) 159–163.
[34] Hasan Çam, J.A.B. Fortes, Work-efficient routing algorithms for rearrangeable symmetrical networks, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 733–741.
[35] Yao-Ming Yeh, T.-Y. Feng, On a class of rearrangeable networks, IEEE Transactions on Computers 41 (11) (1992) 1361–1379.
[36] I. Gazit, M. Malek, On the number of permutations performable by extra-stage multistage interconnection networks, IEEE Transactions on Computers 38 (2) (1989) 297–302.
[37] R.G. Tessier, Fast Place and Route Approaches for FPGAs, MIT, Massachusetts Institute of Technology, 1999.
[38] Larry McMurchie, C. Ebeling, PathFinder: a negotiation-based performance-driven router for FPGAs, in: Proceedings of the ACM Third International Symposium on Field-Programmable Gate Arrays (FPGA'95), ACM, Monterey, California, USA, 1995.
[39] R. Ferreira, A. Garcia, T. Teixeira, J.M.P. Cardoso, A polynomial placement algorithm for data driven coarse-grained reconfigurable architectures, in: IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07), 2007.
[40] Electrical and Computer Engineering Department at the UCSB, U. ExPRESS Benchmarks, Available from: <http://www.express.ece.ucsb.edu/benchmark/> (accessed 03.11.08).
[41] Vaughn Betz, Jonathan Rose, A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, 1999, p. 23.
[42] VPR and T-VPack 5.0.2, Full CAD Flow for Heterogeneous FPGAs, Available from: <http://www.eecg.utoronto.ca/vpr/> (accessed 24.03.09).
[43] Frank Bouwens, Mladen Berekovic, Andreas Kanstein, G. Gaydadjiev, Architectural exploration of the ADRES coarse-grained reconfigurable array, in: Proceedings of the Third International Conference on Reconfigurable Computing: Architectures, Tools and Applications (ARC'07), Springer-Verlag, Mangaratiba, Brazil.
[44] Stephen Friedman, Allan Carroll, Brian Van Essen, Benjamin Ylvisaker, Carl Ebeling, S. Hauck, SPR: an architecture-adaptive CGRA mapping tool, in: Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09), ACM, Monterey, California, USA, 2009.
[45] J. Barbie, F. Reblewski, Emulation System having a Scalable Multi-level Multi-Stage Hybrid Programmable Interconnection Network, USA, 1999, US Patent 5907679.
[46] Abdel Ejnioui, N. Ranganathan, Multi-terminal net routing for partial crossbar-based multi-FPGA systems, in: Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays (FPGA'99), ACM, Monterey, California, USA, 1999.
[47] S. Lin, Y. Lin, T. Hwang, Net assignment for the FPGA-based logic emulation system in the folded-clos network structure, IEEE Transactions on Computer-Aided Design 16 (3) (1997) 316–320.
[48] B. Neji, Y. Aydi, R. Ben-atilallah, S. Meftaly, M. Abid, J.-L. Dykeyser, Multistage interconnection network for MPSoC: performances study and prototyping on FPGA, in: Third International IEEE Design and Test Workshop (IDT'08), Monastir, 20–22 December, 2008.

[49] Gayatri Mehta, Justin Stander, Mustafa Baz, Brady Hunsaker, A.K. Jones, Interconnect customization for a hardware fabric, ACM Transactions on Design Automation of Electronic Systems (TODAES) 14 (1) (2009) 1–32.
[50] Frank Vahid, Greg Stitt, R. Lysecky, Warp processing: dynamic translation of binaries to FPGA circuits, Computer 41 (7) (2008) 40–46.

Ricardo dos Santos Ferreira has been an Associate Professor of Computer Science at the Federal University of Viçosa, Brazil, since 1992. He received the Degree in Physics and Master Science in Computer Science from the Federal University of Minas Gerais, Brazil, in 1991 and 1995. He received his PhD in Microelectronics from the Catholic University of Louvain at Louvain-la-Neuve, Belgium, in 1999. His interests are now primarily concerned with Reconfigurable Computer Architectures, Placement and Routing Algorithms, and Interconnections.

João M. P. Cardoso received a 5-year Electronics Engineering from the University of Aveiro in 1993, and an M.Sc and a PhD degree in Electrical and Computer Engineering from the IST/UTL (Technical University of Lisbon), Lisbon, Portugal in 1997 and 2001, respectively. He is currently Associate Professor with tenure at the Department of Informatics Engineering, Faculty of Engineering of the University of Porto. Before, he was with the IST/UTL (2006–2008), a senior researcher at INESC-ID (2001–2009), and with the University of Algarve (1993–2006). In 2001/2002, he worked for PACT XPP Technologies, Inc., Munich, Germany. He has participated in the organization of a number of conferences and he serves as a Program Committee member for various international conferences. He is co-author of a Springer book and co-editor of two Springer LNCS volumes. He has (co-)authored over 80 scientific publications (including journal/conference papers and patents) on subjects related to compilers, embedded systems, and reconfigurable computing. He is a member of IEEE, IEEE Computer Society and a senior member of ACM. His research interests include compilation techniques, domain-specific languages, reconfigurable computing, and application-specific architectures.

Alex Damiany Assis received a master degree from the Federal University of Viçosa (UFV) in 2010. He was a Professor of Information System at the Federal University of Ouro Preto (UFOP) from 2007 to 2008. His main research interests are in reconfigurable architecture, algorithm development and optimization.

Julio Cesar Goldner Vendramini was born in Colatina, Brasil, on the 21th of Frebuary, 1987. He got a degree in Computer Science from the Federal University of Viçosa, Brazil, in 2010. He is doing his Masters in Computer Science at the same university. His main research interests include computer architecture, reconfigurable architecture, multistage network and FPGA.

Tiago Aparecido Teixeira, M.Sc in Computer Science from the Federal University of Viçosa (2009). Today's Analyst's T.I. Apolo Group. He has experience in computer science, with emphasis in Architecture Computer Systems, acting on the following themes: reconfigurable hardware, evolutionary algorithms, devices (including mobile), performance and open-source software.