An Evolutionary Algorithm for Netlist Partitioning Targeting 3-D FPGAs

Konstantinos Maragos, Student Member and Kostas Siozios, Member and Dimitrios Soudris, Member, IEEE

Abstract—Three-dimensional (3-D) chip stacking is considered as the silver bullet technology to preserve Moore’s momentum and fuel the next wave of consumer electronics. However, the benefits of such an integration technology have not yet been explored due to limitations posed mostly by the lack of efficient tools to support application mapping onto these devices. This letter introduces a framework based on a genetic algorithm for netlist partitioning targeting 3-D reconfigurable platforms. Experimental results prove the efficiency of our solution, as we achieve average reduction of the number of utilized TSVs up to 17% for comparable performance metrics against relevant state-of-the-art algorithms.

Index Terms—Partitioning algorithms, Genetic algorithms, Three-dimensional integrated circuits

I. INTRODUCTION

THREE-dimensional (3-D) chip stacking is a well-established approach to overcome the performance bottleneck and simultaneously shrink the form factor of computational platforms. The shift from horizontal scaling to volumetric stacking has the potential to mitigate many limitations of integrated circuits. Three-dimensional architectures contain multiple physical layers and offer considerable improvement in circuit performance, such as lower power/energy consumption, shorter wire-length, higher integration density, and greater speed, compared to two dimensional (2-D) circuits.

The benefits of using 3-D integration in logic chips are especially great for designing FPGAs, since these architectures suffer from data communication problems. Specifically, the delay and power consumption of the interconnection network are the main bottlenecks compared to alternative ASIC implementations [1]. The reconfigurable industry has already expressed its interest for designing 3-D FPGAs. Typical examples are the 3-D FPGAs provided by Tezzaron, as well as the 2.5-D Xilinx Virtex-7 and UltraScale devices.

Although the opportunities offered by 3-D integration are essentially limitless, efficient application implementation is constrained by the lack of design tools. Physical design in the 3-D realm requires fresh approaches (e.g., new algorithms and cost functions), that can benefit from the architectural features provided by the 3-D platforms. A critical task in this procedure is the netlist partitioning, which deals with the partition and mapping of the design’s netlist into the device layers. During this task, the signal transfers between adjacent layers, implemented through Through-Silicon Vias (TSVs), should be minimized with respect to the desired system’s performance metrics. This requirement is posed mainly by the technology perspective, as the number of TSVs affects the device yield and fabrication cost [1].

Netlist partitioning has been studied extensively in the VLSI (Very Large Scale Integration) context and several heuristic algorithms providing high-quality solutions with polynomial time have been proposed [2]. In particular, Kernighan-Lin (KL) [3] and Fiduccia-Mattheyses (FM) [4] perform netlist partitioning through iterative improvement steps. These algorithms are applicable to designs with up to several hundreds vertices; thus, they can solve sufficiently the macroblock-based partitioning problem. However, in order to benefit from the recent architectural advantages found in 3-D architectures, algorithms that perform netlist partitioning in finer granularity level are absolutely necessary. To address this challenge, the FL algorithm is usually embedded into a multi-level framework, such as the widely-accepted hMetis algorithm [5], which consisted of several distinct steps. Although, the last years various enhancements to hMetis were proposed [6] [7] [8], the efficiency of this algorithm to provide qualitative partitions with acceptably low execution latency makes hMetis to be though as a state-of-the-art approach in VLSI domain.

This letter introduces a framework based on Genetic Algorithm (G.A.), for netlist partitioning targeting 3-D FPGAs. The evolutionary heuristic nature of this algorithm guarantees optimum solutions under multiple objectives without taking into account only local information from the solution search space [9]. Additionally, the inherent parallelism found in our algorithm can be exploited by multi-core architectures to reduce the execution run-time. The introduced algorithm provides high quality solutions for benchmarks of various sizes. Specifically, we achieve to perform application implementation with significant fewer TSVs ranging between 11% and 17%, on average, compared to existing state-of-the-art hMetis algorithm, without sacrificing performance in terms of critical path delay and total routing wire-length.

The rest of the paper is organized as follows: Section II formulates the problem of netlist partitioning for 3-D FPGAs. The proposed algorithm, as well as the experimental results that prove the efficiency of our solution are discussed in Sections III and IV, respectively. Finally, conclusions are summarized in Section V.

K. Maragos, is with the School of ECE, National Technical University of Athens, Greece. Email: komaragos@microlab.ntua.gr
K. Siozios is with the School of ECE, National Technical University of Athens, Greece. E-mail: ksiop@microlab.ntua.gr
D. Soudris is with the School of ECE, National Technical University of Athens, Greece. E-mail: dsoudris@microlab.ntua.gr.
II. PROBLEM FORMULATION

This section formulates the netlist partitioning problem targeting 3-D FPGA devices. A netlist is represented by a hypergraph \( H(V,E) \), where the set of vertices \( V = \{v_i | i = 1, 2, \ldots, n\} \) and hyperedges \( E = \{e_j | j = 1, 2, \ldots, m\} \) denote the application’s modules and nets, respectively. Each net \( e_j \) is a subset of \( V \) with cardinality \( |e_j| \geq 2 \). Also, the partition set \( P = \{p_l | l = 1, 2, \ldots, k\} \) denotes disjoint subsets of \( V \), such as \( p_1 \cup p_2 \cup \ldots \cup p_k = V \). Each module \( v_i \) has a positive real weight \( w(v_i) \), while the total weight of a partition \( p_l \) is defined as \( W(p_l) = \sum_{v_i \in p_l} w(v_i) \). We say that a hyperedge \( e_j = \{v_1, v_2, \ldots, v_d\} \) is cut, if its vertices \( v_1, v_2, \ldots, v_d \) are spread to more than one partitions. Given a hypergraph \( H(V,E) \), the introduced evolutionary algorithm seeks to find a \( k \)-way partition that assigns the vertices of \( H \) into \( k \) disjoint partitions, such that:

- **Objective1**: Minimize the hyperedge-cut, which indicates the number of hyperedges that span multiple partitions.
- **Objective2**: Minimize the source-cut. This objective aims at reducing the number of incident connections, between the source vertex and the rest vertices of a hyperedge, that span more than one partitions.
- **Constraint**: Keep a well-balanced partition scheme. A partition \( p_l \) with a balance constraint must follow the \( W_{min} \leq W(p_l) \leq W_{max} \), where \( W_{min} \) and \( W_{max} \) correspond to the minimum and maximum constraint weights, respectively.

The aforementioned objectives affect the number of utilized TSVs, resulting after the physical implementation phase. More thoroughly, the reduction of the hyperedge-cut leads to solutions where hyperedges reside inside a single partition, whereas the source-cut reduction aims at addressing the layer ordering problem for those hyperedges that span more than one partitions (layers).

III. PROPOSED ALGORITHM

This section describes the proposed solution for netlist partitioning, based on Genetic Algorithm (G.A.). The G.A. is an iterative procedure applied to a population of chromosomes for a number of generations. Each chromosome represents a candidate solution and it is encoded by a finite string of symbols from a given alphabet, known as genes. Regarding the formulation of our problem, a gene defines the partition where a module (e.g. logic block or I/O block) is assigned to.

Starting with the initialization phase, a population of chromosomes is created by defining a set of genes in a random manner (per chromosome). Such a variety guarantees the expansion of the solution space, as well as minimize the possibility to be trap at local minima. Then, a number of genetic operations, namely selection, crossover and mutation, are applied to this population in order to produce the next generation. Finally, during the algorithm’s execution, each chromosome receives a score (referred as fitness) which quantifies its efficiency.

The rest subsections describe in detail the employed operations for netlist partitioning.

1: cross = rand(1, |V|);
2: offspring1 = parent1;
3: offspring2 = parent2;
4: \( \alpha = \text{rand}(0,1) \);
5: if \( (\alpha \leq P_{\text{crossover}}) \) then
6: \( \text{for (each } i \in [1, \text{cross}] \text{ do} \)
7: \( v_i = v_i' | \ v_i' \in \text{offspring1}, \ v_i \in \text{parent2}; \)
8: \( v_i'' = v_i'' | \ v_i'' \in \text{offspring2}, \ v_i'' \in \text{parent1}; \)
9: \( \text{end for} \)
10: end if

Algorithm 1: Crossover Algorithm

1: for (each \( v_i \in V | (1 \leq i \leq |V|) \) do
2: \( \beta = \text{rand}(0,1) \);
3: \( \text{for (each } j \in \{e \mid v_i \text{ is the source node of } e \} | (1 \leq j \leq |V|) \) do
4: \( \text{if } (pu' | \ (v_j \in p_u, \ 1 \leq l \leq |P|) \neq (pu' | \ (v_j \in p_u, \ 1 \leq l' \leq |P|) \) then
5: \( \text{if } (\beta \leq P_{\text{mutation}}) \) then
6: \( p_u = p_u - \{v_i\}; \)
7: \( l'' = \text{rand}(1, |P|) \mid (l'' \neq l); \)
8: \( p_{u''} = p_{u'} \cup \{v_i\}; \)
9: break;
10: end if
11: end if
12: end for
13: end for

Algorithm 2: Mutation Algorithm.

A. Selection

This operation deals with the selection of chromosomes from the current population in order to breed new offspring. The selected chromosomes are referred to as parents of the upcoming generation. Our implementation relies on a tournament selection algorithm, where two candidate parents are randomly selected and the winner is obtained in a deterministic way by evaluating their fitness [10]. For additional flexibility, a chromosome may participate in tournament selection multiple times per generation.

B. Crossover

The crossover operation deals with the generation of new offspring. Our algorithm relies on the one-point crossover approach, as depicted in Algorithm 1. Initially, a randomly selected cross-point is computed; then, two new offsprings are generated by combining the parent’s genes \( (v_i’’ \text{ and } v_i’’) \) in reference to the previously mentioned cross-point. The inherent randomness of the employed crossover scheme minimizes the probability of the G.A. to be trapped to local minima. After exhaustive exploration, we found that optimum partitioning results are retrieved when the probability of applying crossover operation, mentioned as \( \alpha \), is less than \( P_{\text{crossover}} = 80\% \).

C. Mutation

The mutation operation maintains the diversity of chromosomes through the generations, by altering the offspring’s
genes. The employed mutation, as depicted in Algorithm 2, traverses all the source vertices and checks whether the attached hyperedges span more than one partitions. If so, there is a probability \( P_{\text{mutation}} \) to move this source vertex to a different partition. Based on our exploration we found that a good compromise between the quality of solutions and the avoidance of entrapment to local minima is achieved when the probability \( \beta \) is less than \( P_{\text{mutation}} = 1\% \). Note that the source vertices of hyperedges that are interleaved in a partition are not taken into account during this operation, because both their hyperedge-cut and source-cut are equal to 0.

### IV. Evaluation Results

This section provides a number of quantitative results that highlight the effectiveness of the introduced solution in comparison to the state-of-the-art hMetis algorithm [5]. For this purpose, we employ the 20 biggest MCNC benchmarks [11] and 3-D FPGAs with up to five layers. Each of these layers follows the homogeneous island-style architecture. This resource homogeneity is taken into consideration by assigning identical weights \( w(v_i) \) to the netlist’s logic modules. Regarding the connectivity between adjacent layers, it is provided through vertically aligned Switch Boxes (SBs). A SB consists of four TSVs, since this is the narrowest common vertical channel width for successful routing among benchmarks. The electrical equivalent parameters (i.e. resistance and capacitance) for the TSVs are extracted based on state-of-the-art solutions published in relevant literature [12]. Table I summarizes the properties of the employed benchmark suite, as well as the size of the minimum FPGA array, where each of the benchmarks is mapped to. The selection of a minimum array per benchmark, instead of a common 3-D FPGA for the entire benchmark suite, is a widely adopted approach in order to stress the CAD algorithms.

As we have already mentioned, the proposed G.A. does not solve the netlist partitioning problem in general. On the contrary, we optimize our algorithm to maximize the efficiency of designs mapped onto 3-D FPGA platforms. For this purpose, our experimentation relies on the TPR tool [13] (the only publicly available toolset for application placement and routing onto 3-D FPGAs), which employs the well-established hMetis partitioning algorithm [5]. For sake of completeness, we also integrated the proposed G.A. partitioning to the TPR tool (by replacing hMetis), as depicted at Fig. 1. Consequently, this comparison quantifies the efficiency of our introduced G.A. against the hMetis algorithm. Unfortunately, we cannot provide direct comparison against other partitioning algorithms found in literature, since they are not publicly available and/or they are not integrated as part of a physical design flow for 3-D FPGAs.

The number of generations in heuristic algorithms affects the efficiency of the derived solution. In order to study this topic more thoroughly, we evaluate the impact of this parameter in terms of TSVs reduction and run-time overhead. The results from this analysis regarding a 3-D FPGA with 3 layers are summarized at Fig. 2. To be more precise, the horizontal axis corresponds to algorithm’s generations, while the TSV reduction and run-time overhead versus the hMetis algorithm are depicted at vertical axes. Note that similar results are retrieved for devices consisting of a different number of layers. Based on this analysis, we decide for the rest of our experimentation to execute the G.A. for 200 generations, as it is a good compromise between the achieved reduction of utilized TSVs and the algorithm’s execution overhead. Although the introduced algorithm exhibits inherent parallelism, the analysis performed throughout this paper assumes single thread execution. Thus, one might expect significant reduction at run-time overhead versus hMetis by incorporating multicore processors.

Next, we study the efficiency of partitions derived using the proposed G.A. according to a number of system metrics. For comparison purposes, this analysis assumes (without affecting the generality of the introduced solution) that the number of vertices per partition ranges between \( \frac{\text{num-}b}{\text{num}} \times N \) and \( \frac{\text{num}+b}{\text{num}} \times N \), where \( L \) and \( N \) denote the number of layers and the number of logic blocks per benchmark, respectively. The parameter \( b \) defines the imbalance among partitions \( (b = 5 \) for our experimentation). Such a constraint leads to partitions with similar number of logic blocks per layer, as compared to the hMetis algorithm.

The results from this analysis are plotted in Fig. 3. For demonstration purposes, the vertical axis is normalized over the corresponding results retrieved by hMetis netlist partitioning (with the flow discussed in Fig. 1). This analysis indicates that our algorithm enables the application’s placement and routing (P&R) with significant fewer TSVs, ranging between 11% and 17%, compared to the existing state-of-the-art approach. Even though one might expect that such a reduction might introduce routing congestion problems and hence performance degradation, these penalties are almost negligible.

---

1943-0663 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
TABLE I
EMPLOYED BENCHMARKS AND TARGET 3-D FPGAS.

<table>
<thead>
<tr>
<th>3D-Metrics</th>
<th>benchmarks</th>
<th>apex4</th>
<th>apex3</th>
<th>apex2</th>
<th>spla</th>
<th>alu</th>
<th>clma</th>
<th>sue2</th>
<th>sue1</th>
<th>bigkey</th>
<th>frisc</th>
<th>des</th>
<th>pdc</th>
<th>hMetis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total wire-length</td>
<td>4,556</td>
<td>49</td>
<td>8,445</td>
<td>50</td>
<td>28</td>
<td>18</td>
<td>335</td>
<td>27</td>
<td>189</td>
<td>48</td>
<td>57</td>
<td>44</td>
<td>399</td>
<td></td>
</tr>
<tr>
<td>Total area</td>
<td>1,177</td>
<td>14</td>
<td>50</td>
<td>8,445</td>
<td>50</td>
<td>28</td>
<td>18</td>
<td>335</td>
<td>27</td>
<td>189</td>
<td>48</td>
<td>57</td>
<td>44</td>
<td>399</td>
</tr>
</tbody>
</table>

Fig. 3. Evaluation of the proposed partitioning algorithm as compared to hMetis algorithm [5].

Specifically, based on Fig. 3 the proposed G.A. achieves similar total wire-length and critical path delay against to the existing hMetis algorithm. For the sake of completeness, at this figure we also plot the average variations of source-cut and hyperedge-cut among the studied benchmarks for different number of device layers.

Finally, we evaluate the utilization of TSVs per SB, as they retrieved after netlist P&R onto the target 3-D FPGA. These results for the existing (hMetis) and the proposed partitioning algorithm are summarized at Table II. Based on this analysis we might conclude that our proposed G.A. reduces on average the percentage of 3-D SBs that utilize only 1, 2, 3 and 4 of the TSVs by 14%, 28%, 44% and 54%, respectively, compared to those derived with hMetis algorithm. As a consequence, the spare TSVs per SB can be used for enhancing other design metrics, such as fault-tolerance with techniques proposed at relevant publications [14].

V. CONCLUSIONS

A novel evolutionary algorithm for addressing the netlist partitioning problem, was introduced. In contrast to similar state-of-the-art approaches which are platform-agnostic, the introduced one takes into account inherent constraints posed by the 3-D architectures. Experimental results with various benchmarks highlight the efficiency of the introduced approach, since it reduces significantly the number of utilized TSVs, ranging between 11% and 17%, without performance degradation in terms of critical path delay and total wire-length.

REFERENCES