How Preserving Circuit Design Hierarchy During FPGA Packing Leads to Better Performance

Dries Vercruyce, *Graduate Student Member, IEEE*, Elias Vansteenkiste, *Graduate Student Member, IEEE*, and Dirk Stroobandt, *Member, IEEE*

***Abstract*—Generating a configuration for a field-programmable gate array (FPGA) starting from a high level description of a design is a time consuming task. The resulting configuration should have a high quality so that the FPGA resources are used in an efficient way while being able to run at high clock frequencies and having a low power consumption. In this paper, we present MULTIPART, a new hierarchical packing algorithm that obtains better quality and faster runtimes when compared to the frequently used AAPack packer in VPR. MULTIPART combines the benefits of partitioning-based and seed-based packing approaches. It tries to preserve the design hierarchy during packing. This results in a gain of 32% in total wirelength and a gain of 10% in critical path delay. The partitioning-based methodology allows us to exploit multithreading, leading to 9.3× faster packing runtimes on a CPU with 10 cores. We also gain in the total routing runtime because MULTIPART reduces congestion problems on a higher level. The subcircuits in the partitioned circuit are clustered with a seed-based packer. This allows MULTIPART to deal with the constraints of complex heterogeneous architectures. In short, MULTIPART targets heterogeneous commercial FPGAs with a lower runtime while increasing the quality of the configuration. The source code of MULTIPART is available in our FPGA CAD framework on Github.**

***Index Terms*—Design hierarchy, field-programmable gate array (FPGA), packing, partitioning.**

I. INTRODUCTION

GENERATING a configuration for an field-programmable gate array (FPGA) takes a lot of runtime, especially, for large designs that are common in commercial applications [1]. This configuration should have a high quality in terms of the three important metrics: 1) cost; 2) speed performance; and 3) power consumption. The required amount of FPGA resources determines the cost of the design. The maximum clock frequency is an indicator of the speed performance. Each of these quality requirements makes it difficult and hence time-consuming to find a configuration with good quality. Several techniques are proposed in the literature to improve the runtime of the FPGA CAD flow without degrading the quality of results. Most of them rely on multithreaded implementations. They leverage the multicore processors that have become a commodity in the last decade [2], [3].

Modern FPGA architectures have a hierarchical structure to improve area and delay [4]. On each hierarchical level a number of equivalent blocks are available. The blocks are connected by a local programmable interconnection network on that level. The routing pathways on the lowest hierarchical level are short, and hence fast, while the top level routing infrastructure is slower. Due to this hierarchical structure a packing step is introduced in the FPGA CAD tool flow. During packing, all the low level primitives in the circuit are clustered into the high level functional block types available in the FPGA architecture. These clusters are then placed and routed on the highest hierarchical level.

Several optimization criteria are used during packing such as minimum channel width (MCW), total wirelength (TWL), area, critical path delay (CPD), and power consumption. They are evaluated after the design is routed. The MCW is the maximum number of tracks in a channel needed to route the design. Finding this value takes a lot of runtime because many routing iterations are required, each time with a smaller channel width (CW) until congestion problems prevent a routable solution. The MCW of a design is closely related to the TWL [20], [21], [23]. Therefore, TWL is a good measure for the number of required routing resources. It is also closely related to the power consumption. The area is mainly determined by the number of high level clusters in the packing solution. However, if a design is too tightly packed, it can lead to congestion problems [1]. Congestion prone designs require extra wire tracks in the routing channels. This leads to more area and metal layers and may force the FPGA designer to move toward a larger more expensive chip. Connections at the lower hierarchical levels are short and have a lower capacitance, which makes them faster. Consequently, the CPD can be minimized by using these fast connections for the critical connections in the design. They should also be favored for connections with a high switching activity if reducing power consumption is an objective. Packing can greatly influence the end result of the entire tool flow. If the fast interconnection network on the low hierarchical levels is used efficiently, gains in TWL, CPD, and power consumption are obtained in the routed design.

Manuscript received December 23, 2016; revised March 30, 2017; accepted May 16, 2017. Date of publication June 20, 2017; date of current version February 16, 2018. This work was supported by the European Commission in the Context of the H2020-FETHPC EXTRA Project under Grant 671653. The work of D. Vercruyce was supported by the Ph.D. Grant of the Research Foundation Flanders (FWO). A portion of this work was published in the 26th International Conference on Field Programmable Logic and Applications (FPL), 2016. This paper was recommended by Associate Editor Y. Chen. *(Corresponding author: Dries Vercruyce.)*

The authors are with Ghent University, 9000 Ghent, Belgium (e-mail: dries.vercruyce@ugent.be; elias.vansteenkiste@ugent.be; dirk.stroobandt@ugent.be).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2717786

In this paper, we propose a fast multithreaded packing algorithm, MULTIPART, that improves the quality of the routed design by preserving the design hierarchy during packing. MULTIPART reduces the TWL with 32% and the CPD with 10% when compared to the conventional seed-based packing (SBP) approach in VPR for the heterogeneous benchmark designs in the Titan23 design suite targeted to Altera's Stratix IV FPGA [1]. This is an important result because it closes a major part of the gap between commercial and academic results in terms of TWL. Murray *et al.* [1] reported a TWL gap of 2.19× comparing VPR with Altera Quartus II. In this paper, we use the same architecture, benchmark designs, placement and routing algorithms as in the work of Murray *et al.*, only the packing algorithm is replaced with MULTIPART. The achieved wirelength gain greatly reduces the gap between academic and commercial results by just replacing the packing algorithm. An additional advantage of partitioning-based packers is the opportunity to implement a multithreaded version of the algorithm as the partitioned parts can be treated independently. Our implementation has an average runtime speedup factor of 9.3× on a CPU with 10 cores when compared to AAPack.

The remainder of this paper is organized as follows. In Section II, background concepts are explained. A global overview and contributions are given in Section III. The principles and improvements to the partitioning and SBP step are described in more detail in Sections IV–VI. In Section VII, the additional functionality in the timing-driven version of MULTIPART is explained. Experiments are described in Section VIII and this paper concludes with future work and a conclusion in Section IX.

II. RELATED WORK

Existing packing approaches can be divided into three main classes: 1) seed-based; 2) depth-optimal; and 3) partitioning-based packers. Seed-based packers are fast because they pack the circuit in a single pass. They produce tight packings but are unable to escape from local minima because they use a bottom-up approach and lack a global overview. For each new functional block a seed block is selected with a certain optimization criterium in mind. Then an affinity metric between the seed block and its surrounding blocks is calculated. The block that scores the highest on this affinity metric is packed into the cluster. This is repeated until the cluster is full. Several cost functions for the affinity metric are proposed to improve the quality for a specific optimization criterium. A well-known seed-based packer is T-VPack [5]. Its latest version AAPack [6] is used in the VTR tool flow [7]. This packer is used as a baseline in our experiments. The main objective of this packer is minimizing the CPD. A routability driven affinity metric is proposed in iRAC [8] and T-RPack [9]. Un/DoPack [10] and T-NDPack [11] try to increase routability with depopulation-based clustering methods. These packers resolve congestion problems in the routing network by preventing that functional blocks are completely filled. The total area of the functional blocks is thereby increased because the functional logic is spread across the FPGA. P-T-VPack [12] and W-P-T-VPack [13] incorporate switching activities of the nets in the affinity metric. This reduces power consumption at the cost of an increased MCW and CPD. The multiobjective packers MO-Pack [14] and YAMO-Pack [15] incorporate several criteria in the affinity metric to obtain good quality for all optimization criteria.

Due to synchronization problems, it is hard to apply multithreading to the seed-based approach. Each thread would have its own seed. Picking blocks from the neighborhood of that seed could conflict with the neighborhood of the seed from other threads that run concurrently. To our knowledge, no multithreaded seed-based packer is proposed yet.

Depth-optimal methods are proposed in TLC [16], RCP [17], and MLC [18]. These packers try to optimize the critical path by duplicating timing-critical netlist primitives. Although these methods reduce the CPD, they lead to an increase in total area.

In partitioning-based packers [19]–[23] the clusters are determined by performing a hierarchical partitioning of the circuit. Hierarchical partitioning is a top down-approach. First the circuit is bipartitioned. The partitioning algorithm minimizes the number of connections cut while splitting the circuit in two. The two parts of the circuit are then further bipartitioned independently. The resulting parts are again bipartitioned and this repeats itself recursively until the parts contain less primitives than a predefined limit.

Fully partitioning-based packers are proposed in Marrakchi *et al.* [19], PPack [20], and PPack2 [21]. These partitioning-based methods obtain good quality results but still have some fundamental problems that we solved in our packing approach. The first problem is that it is hard to impose constraints during partitioning. It is not possible to restrict the number of pins on each subcircuit, while functional blocks on physical chips have a limited number of input pins. Partitioning tends to minimize this number, but there is no guarantee that every subcircuit will have less than the allowable number of pins. Furthermore, it is hard to control the number of blocks in each subcircuit. It is likely that there are too many or too few blocks in a subcircuit compared to the number of available positions in a functional block. Due to these problems a constraints enforcing post processing step is required. This results into a loss of quality because the natural hierarchy of the circuit is disturbed. In Marrakchi *et al.* [19] blocks are swapped between the clusters until all constraints are met. In PPack and PPack2, a theoretical architecture without an input bandwidth constraint for the functional blocks is used. This is far removed from the commercial FPGA devices. Even with this unrealistic architecture it is required to swap blocks between the clusters because a limited number of blocks can be packed into a functional block.

Another problem for partitioning-based packers is the large packing runtimes because many subcircuits have to be partitioned. Each time a subcircuit on a certain hierarchy level is cut in half, it leads to two new subcircuits on the next hierarchical level. In total the number of required cuts to fully partition a circuit is approximately equal to the number of blocks in that circuit. In PPack2, a runtime overhead of 10× is reported for the MCNC-20 benchmark circuits when compared to VPR4.3.

Luckily, partitioning-based packing offers an opportunity for multithreaded parallelism. Once a subcircuit is split in half, the two resulting subcircuits can be partitioned concurrently by spawning a separate thread for each subcircuit.

HDPack [22] and PartSA [23] solve the fundamental problems while preserving the reduction in TWL. HDPack uses a seed-based clustering method but incorporates physical information from a global placement in the affinity metric. This global placement is obtained by partitioning the circuit to a certain hierarchical level. All blocks in the subcircuits on this level have a physical location on the architecture assigned. The physical information leads to a gain of 16% in TWL and a gain of 8% in CPD when compared to T-VPack for the MCNC-20 benchmark designs. However, these gains come at the cost of additional runtime to partition the circuit. PartSA proposes a fully partitioning-based packer with a simulated annealing-based clustering step instead of the constraints enforcing post processing step. It uses the distance in the partition tree as a cost metric in the annealing method. This way it is possible to impose a bandwidth constraint on each functional block by adding an additional cost, relative to the number of input pins. The problem with the size of the subcircuits is solved because the annealing-based packer fills all clusters until no more empty positions are left. This packer achieves a gain of 26% in TWL when compared to AAPack. As it is possible to use multithreading, up to 2.3× faster runtimes are obtained for the largest VTR benchmark circuits on a CPU with four cores. A problem occurs when complex architectures with sparse local interconnect crossbars are used. Each time two blocks are swapped, a detailed routing would be required to check if a legal solution is obtained because not all connections are possible in the sparse local interconnection network. This results in large runtimes because many swaps are required in the simulated annealing algorithm to obtain a result with a low wiring cost and it takes too much time to perform a detailed routing every time.

III. Contributions

In this paper, we further elaborate on a preliminary version of MultiPart [23]. MultiPart combines the advantages of partitioning-based and SBP in one algorithm. It consists of two main parts (Fig. 1): 1) a partitioning step to partition the circuit hierarchically into a set of subcircuits and 2) an SBP step that clusters all these subcircuits concurrently. This approach has a number of advantages over classical packing methods.

1) A large reduction in TWL is achieved because the natural hierarchy of the circuit is retained in the partitioning step.
2) No partitioning is required on the deep hierarchical levels which saves runtime. When a certain hierarchy level is reached, the fast SBP approach is used to cluster the subcircuits concurrently. A multithreaded implementation can be used because the subcircuits are weakly interconnected.
3) Heterogeneous designs can be packed on commercial architectures with sparse local interconnect crossbars. The required detailed routing is taken care of by the seed-based packer.

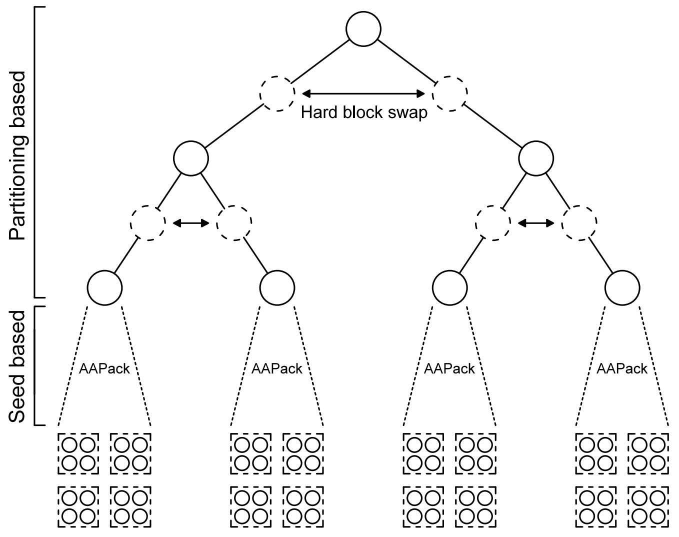


Fig. 1. Overview of the MultiPart algorithm. The tool consists of two steps: a partitioning step to split the circuit into a set of subcircuits and an SBP step to cluster these subcircuits concurrently.

To allow a hierarchical packing of heterogeneous designs we extend the functionality of the partitioning and SBP step in the preliminary version of MultiPart. The main contribution in the partitioning step is a hard block swap. The swap redistributes the hard block primitives between both subcircuits after each cut (Fig. 1). This way the available hard block types in the architecture are used efficiently while maximally preserving the natural hierarchy in the circuit. A prepartitioning step is added to improve packing quality. We apply a global hard block assignment to ensure that the memory hard block types (M9K and M144K) are used efficiently. Furthermore, specific netlist primitives are clustered into molecules prior to the partitioning step to prevent that they are scattered over the subcircuits, similar to the prepacking step in AAPack [6]. We also generate a design specific architecture to reduce the SBP runtime overhead. The design specific architecture file only contains the modes used in the design. Timing-driven MultiPart now focusses on critical edges instead of critical paths, leading to a larger gain in CPD.

The new functionality allows us to cluster large heterogeneous designs. We use large designs to analyze the opportunities of MultiPart on commercial applications. Most academic packers are evaluated with small nonheterogeneous designs. In the experiments section of this paper, we analyze the scalability of our tool with increasing design size and increasing number of available threads.

Next to the additional functionality, we also explored the parameter space using HPC. Long placement and routing runtimes are required to generate a configuration for the large designs in the Titan23 suite. Therefore, this exploration would not be possible on commodity hardware. In this paper, we propose a parameter combination for the best runtime-quality tradeoff. With the presented results, other parameters can be chosen if MultiPart is used for high quality or fast packing purposes. In the parameter sweeps, we use the 13 smallest Titan23 designs in order to keep the total packing, placement, and routing runtime acceptable. In the experiments section, MULTIPART is analyzed with all routable Titan23 designs.

Our main contributions are as follows.

1) MULTIPART: A multithreaded partitioning-based packing tool that is able to generate high quality configurations in less runtime (based on [23]).
2) An enhanced version of the partitioning and SBP step to allow the clustering of large and highly heterogeneous designs.
3) A better parameter exploration of MULTIPART and an extraction of the best parameter values for the runtime-quality tradeoff.
4) A scalability analysis of our multithreaded approach in function of the number of available threads and the size of the designs.

IV. HIERARCHICAL CIRCUIT BIPARTITIONING

Digital designs are built up hierarchically to cope with the increasing vastness and complexity of applications. The application is divided into several subproblems, which are in turn divided into smaller subproblems. Typically, the complexity within these subproblems is high while there are a small number of connections between them. This hierarchical structure is exploited in our partitioning-based packing tool to reduce the total required wirelength in the routed design.

In the partitioning phase, the circuit is split hierarchically into a set of weakly interconnected subcircuits. MULTIPART recursively bipartitions the circuit while minimizing the number of connections to be cut. This way an optimal Rent characteristic [24] is obtained and the natural hierarchy of the circuit is preserved [21]. Rent's rule states that the relation between the number of terminals, n_t (cut nets), and the number of internal blocks, n_b, will be a power law (1) on each hierarchical level of a recursively bipartitioned circuit in case the circuit is hierarchically partitioned while minimizing the number of cut nets. Rent's rule emerges because of the natural hierarchy present in the circuit, which is introduced by the human hardware designers

$$n_t = t * (n_b)^{\text{rent\_exponent}}. \tag{1}$$

Hierarchical circuit bipartitioning works as follows. First, the circuit is split into two parts. The amount of cut edges is minimized and the difference in size between the parts is limited by an unbalance factor. This leads to two subcircuits on the first hierarchy level of the partition tree. Then the two resulting subcircuits are further bipartitioned leading to four subcircuits on the next hierarchy level. The number of hierarchy levels is approximately equal to log2(N), with N the number of blocks in the circuit.

V. CLUSTERING THE SUBCIRCUITS CONCURRENTLY WITH THE SEED-BASED PACKER

In the second phase, we cluster all subcircuits that result from the partitioning phase with a seed-based packer. We use the seed-based packer AAPack 7.0 [6] because it is able to cluster highly heterogeneous designs on complex architectures. The result of the partitioning step is a set

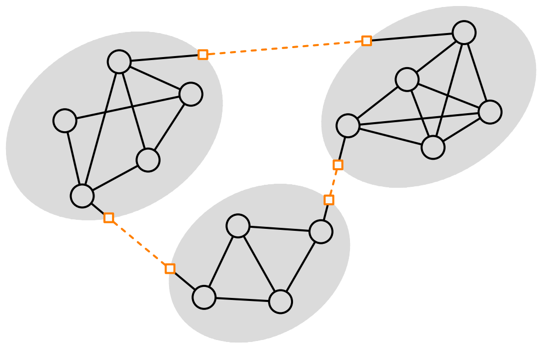


Fig. 2. Result of the partitioning step: the circuit is split into a set of weakly interconnected subcircuits. All edges that interconnect two subcircuits are replaced by an input/output pin combination.

of weakly interconnected subcircuits (Fig. 2). Due to the low interconnection complexity between the subcircuits they can be treated independently by the seed-based packer. This leads to the opportunity to cluster all independent subcircuits concurrently on a separate thread without a synchronization overhead between the threads. Prior to SBP all edges that interconnect two subcircuits are replaced by an input/output pin combination (Fig. 2). After the independent clustering all the packed blocks have to be merged into a single netlist. The methodology has a number of drawbacks though.

1) *Resource Quantization:* The resources in the FPGA architecture (functional blocks) can not be shared between the subcircuits because they are all packed independently.
2) *Difficult Selection of RAM Types:* As all subcircuits are packed concurrently, no information is available on the FPGA resources used by the other subcircuits. This leads to a problem for RAM primitives because they can be packed into multiple RAM block types.
3) *Runtime Overhead:* Each subcircuit is packed with AAPack on a separate thread. Launching an AAPack thread introduces a runtime overhead. Initializing the structure of the complex architecture causes the largest overhead.
4) *Lack of Correct Timing Analysis:* All combinational paths that span over more than one subcircuit are cut. Consequently, the seed-based packer lacks correct timing information for these cut paths.

To mitigate the first three drawbacks, functionality is added before partitioning (prepartitioning step) and during partitioning (hard block swap). First we discuss the prepartitioning step, then the hard block swap is explained. In the timing-driven section (Section VII), we explain how the timing problem is solved.

A. Prepartitioning

Prepartitioning consists of two steps. In the global hard block assignment an RAM type is assigned to all RAM primitives in the design. Then we cluster netlist primitives into molecules to prevent that they are scattered over the subcircuits during partitioning. This way we reduce the resource quantization problem. Further improvement to this problem is achieved in the hard block swap. Besides this, we also generate a design specific architecture to solve the runtime overhead problem.

1) Global Hard Block Assignment: Commercial architectures provide multiple hard block types for some of the hard block primitives in a design. In the Stratix IV architecture two RAM block types are available: 1) 9-Kbit M9K blocks and 2) 144-Kbit M144K blocks. AAPack uses the following policy for the RAM blocks: if a memory slice can be packed into an M9K block then this block should be used. If the memory slice is not supported by the M9K blocks or no more M9K blocks are available then the primitive is packed into an M144K block. As MULTIPART clusters all subcircuits independently, no information is available about the M9K and M144K usage by the other subcircuits. Consequently it is not possible to know if free M9K blocks are available. To solve this problem, an RAM block type is assigned to all memory slices before partitioning because then a global overview of the design is available. The following methodology is used: first all slices get the best suitable RAM block type assigned. An M9K block is preferred over an M144K block because it consumes less area and there are more available positions on the chip. Then the RAM block usage is analyzed. If too many M9K blocks are required then slices that are currently assigned to an M9K block are changed to M144K until enough M9K blocks are available. Hereby, we use the RAM blocks efficiently by minimizing the sum of the required M9K and M144K blocks.

2) Molecule Generation: In the molecule generation step, groups of netlist primitives which should stay together during partitioning are clustered into molecules. This step is adopted from the prepacking step in AAPack [6]. We generate molecules for primitives that are part of a chain and DSP and RAM primitives. Many carry and share chains are part of a long combinational path in the circuit, leading to a large path delay in the routed design. Modern architectures have short and hence fast dedicated connections built-in for these chains. If a connection is cut during partitioning, then this connection is routed with the slow interconnection network. For this reason connections between blocks that are part of a chain should remain uncut. To avoid that such connections are cut, we generate molecule blocks that contain all atom blocks in a chain. For some of the DSP and RAM block primitives we know that they should be clustered into the same hard block prior to the partitioning phase. Each DSP element in modern architectures consists a few multiplier primitives connected to an accumulator. The multiplier and accumulator primitives should be packed into the same DSP block. Similarly, if a set of RAM primitives with the same address and control signals can be packed in a single RAM block, then the primitives should be clustered before partitioning.

3) Design Specific Architecture Generation: When AAPack is launched, it builds the structure of the architecture in the memory. Modern architectures lead to a large initialization overhead because they consist of many different functional block types which support multiple operation modes. This large overhead nullifies the runtime speed-up obtained with the multithreaded SBP approach because each subcircuit is clustered on a separate thread with AAPack. To enable fast packing on complex architectures, we automatically generate a design specific architecture in which only the required functional blocks and modes are described. This architecture does not only diminish the initialization overhead, it also reduces the total SBP runtime because all redundant modes are removed from the architecture.

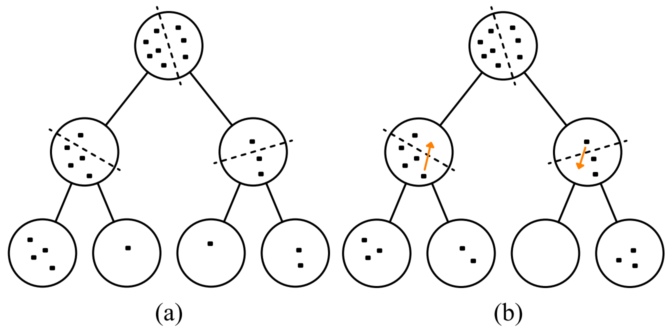


Fig. 3. Influence of hard block redistribution in the partitioning step on the total number of required hard blocks. (a) Without hard block swap. (b) With hard block swap.

B. Hard Block Swap

The hard blocks in an FPGA architecture are typically able to implement a number of netlist primitives [25]. All memory slices that share the same address and control signals can be packed together into the same RAM block as long as the depth of the block is large enough. Similarly, a DSP block is able to implement two DSP molecules. The hard blocks can not be shared between the subcircuits because they are clustered independently. If a set of M memory slices with the same address and control signals are scattered over all the subcircuits then each subcircuit requires at least one RAM block to implement the slices. In the worst case, this leads to M required RAM blocks. To address the problem we add a hard block swap during recursive bipartitioning. The aim is to redistribute the hard block primitives between both partitions after each cut. An example is shown in Fig. 3 for eight memory slices that share the same address and control signals. Consider an architecture in which the RAM blocks are able to implement three of these slices. The minimum number of RAM blocks to implement these slices is thus equal to three. When no hard block swap is used during partitioning, five RAM blocks are required [Fig. 3(a)]. The first subcircuit requires two blocks, while the others require one. When hard block redistribution is added, memory slices are swapped between both parts after each cut if this reduces the total number of required RAM blocks [Fig. 3(b)]. In this case, only three blocks are required to pack all slices.

The difference with the molecule generation step is that the netlist primitives in this step are clustered to a molecule prior to partitioning. We do this because there is little or no choice for the blocks we cluster. These blocks have to be packed into the same functional block. During partitioning, we add a hard block swap because there are many candidate primitives that fit into a few hard blocks. Choosing a hard block instance for each of these primitives before partitioning could disturb the natural hierarchy of the circuit. The decision can not be based on the hierarchy that is only obtained during partitioning.

The algorithm is defined as follows. Before partitioning we group all hard block slices that can be implemented by the same hard block type. Let us assume that one of these groups consists of P hard block primitives and that each hard block of this type can implement N slices. In total, we thus require $H = \lceil (P/N) \rceil$ hard blocks of this type. When a circuit is partitioned in two parts, C_1 and C_2, the parts have P_{C_1} and P_{C_2} hard block slices, respectively. Hereby, the sum of P_{C_1} and P_{C_2} is equal to P. Now a total of $H_1 + H_2$ hard blocks are required, with H_1 equal to $\lceil (P_{C_1}/N) \rceil$ and H_2 equal to $\lceil (P_{C_2}/N) \rceil$. If $H_1 + H_2$ is larger than the original number of required hard blocks (H), then a hard block swap is introduced after partitioning. The aim is to redistribute the hard block primitives in such a way that $H_1 + H_2 = H$. In this swap blocks are moved from C_1 to C_2 or from C_2 to C_1. The direction in which primitives are moved is determined by calculating the number of moves M_{12} and M_{21} that are required to ensure that $H = H_1 + H_2$ when blocks are moved in the $C_{1\rightarrow 2}$ and $C_{2\rightarrow 1}$ direction, respectively. The best direction is determined by the minimum of M_{12} and M_{21}. Once the direction is known, we move blocks between the subcircuits until $H = H_1 + H_2$. Hereby, the moved blocks are chosen in such a way that the total number of cut edges is minimized. Each time a block is moved, the minimum cut increase is chosen by comparing all possible moves. This way, the design hierarchy is maximally preserved.

Once the number of hard block primitives in a subcircuit is smaller than the number of available positions in a hard block instance then all these primitives should be packed into a single hard block. In this case, the primitives are clustered into a molecule to prevent that they are further partitioned.

C. Resource Quantization of the LABs

Each time a subcircuit is split into two parts, connections between LUTs and FFs are cut. If an edge between two primitives that can be packed into the same LAB is cut then the number of LABs increases. The primitives are now located in different subcircuits which are packed independently, each with its own resources. In the experiments section, we observe that this problem leads to a small area increase for some of the designs. A possible solution is assigning a larger weight to connections between LUTs and FFs that should be packed into the same LAB. Unfortunately, this solution leads to a loss in quality because imposing additional constraints on the cut connections disturbs the natural hierarchy of the design. An example is an LUT-FF pair with a single connection between them. Other examples are LUT-LUT connections in long combinational paths and LUT/FF connections with a high criticality.

VI. Runtime-Quality Tradeoff in Recursive Bipartitioning

The problem of computing an optimal partition is NP-complete. However, because many application areas benefit from partitioning-based approaches, many heuristic algorithms have been developed. In the hMetis tool [26] a multilevel partitioning technique is used (Fig. 4). This technique consists of three phases: a coarsening phase, an initial partitioning phase, and an uncoarsening and refinement phase. In the coarsening phase, vertices and edges are collapsed to reduce the size of the graph. Then this smaller graph is cut in the initial partitioning phase. The uncoarsening and refinement phase constructs a partition of the original graph. hMetis is used in the partitioning step because it is fast and able to generate bipartitions with high quality.

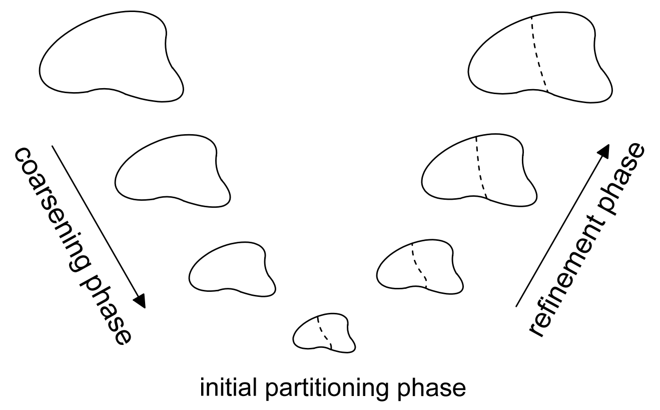


Fig. 4. Multilevel partitioning algorithm that is used in the hMetis partitioning tool. The algorithm consists of three phases: a coarsening phase, an initial partitioning phase, and an uncoarsening and refinement phase.

A. Multithreaded Runtime Speed-Up

Besides the fact that partitioning is an NP-complete problem, the partitioning step also introduces a runtime overhead. In a purely SBP approach the circuit is immediately clustered. In our approach, we first partition the circuit into a set of weakly interconnected subcircuits which are then clustered independently with a seed-based packer. To speed up the partitioning step we use a multithreaded implementation. Once a circuit is split in half then both subcircuits can be further bipartitioned independently on two separate threads. The maximum number of threads is parameterized and is best adapted to the number of cores on the used CPU.

B. Optimizing the hMetis Parameters

Several parameter values are used to optimize the functionality of the hMetis partitioning tool [27].

1) The values of *CType*, *RType*, and *Vcycle* select different algorithms in the multilevel partitioning technique.
2) hMetis computes *Nruns* different bisections and selects the best as the final solution.
3) The maximum unbalance in the number of blocks between both parts of the partition is determined by an unbalance factor (UB).

The value of the parameters is important because they determine the quality of the cut and the total required partitioning runtime. They should be optimized to the problem at hand because each application has a different type of graphs.

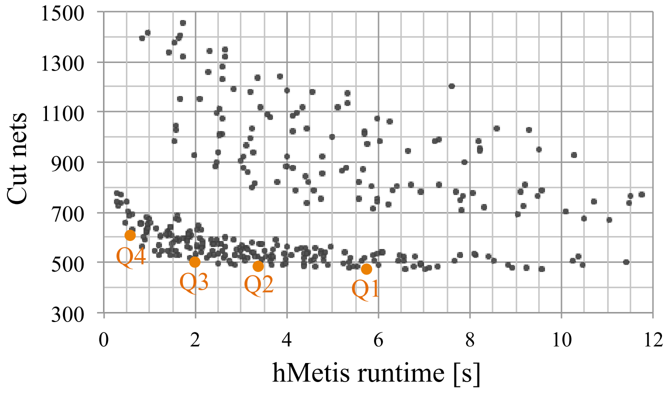


Fig. 5. Influence of the hMetis parameters on the number of cut nets and the partitioning runtime. Four parameter combinations are chosen as four standard quality settings ($Q1 \rightarrow Q4$).

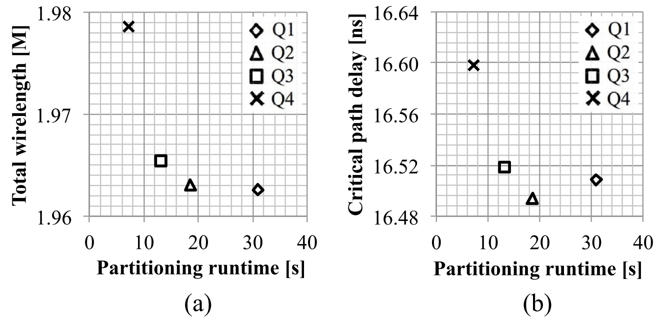


Fig. 6. Influence of the hMetis quality on partitioning runtime with (a) TWL and (b) CPD.

In MULTIPART, the graphs are a representation of digital circuits. To find the optimal hMetis parameters, we bipartition large heterogeneous circuits for a range of parameter combinations. This range consists of all *CType*, *RType*, and *Vcycle* combinations. For the *Nruns* parameter, values between one and ten are analyzed. The unbalance factor is not considered in this grid search, instead it is separately analyzed below. Fig. 5 shows the number of cut nets in function of the required hMetis runtime.

The number of cut nets varies from 471 up to 2425 whereas the partitioning runtime varies from 0.27 to 14.5 s (not all parameter combinations are shown in Fig. 5). The important parameter combinations are those that lie on the Pareto front. In MULTIPART four of the Pareto optimal parameter combinations are used as four quality standards in the partitioning step. These are highlighted as orange dots in Fig. 5. The fastest quality setting ($Q4$) requires a geomean runtime of 0.58 s to partition a graph and results in a cut of 606. This is reduced to 471 at the cost of a runtime increase (5.75 s) in the best quality setting ($Q1$).

The influence of the partitioning quality on the post-routing TWL and CPD is shown in function of the total required partitioning runtime in Fig. 6. As expected, less wires are required when partitions with a better quality are used. This comes at the cost of an increase in total runtime. For the CPD a similar behavior is observed when the quality increases from $Q4$ to $Q2$. The parameter combination that results in the smallest

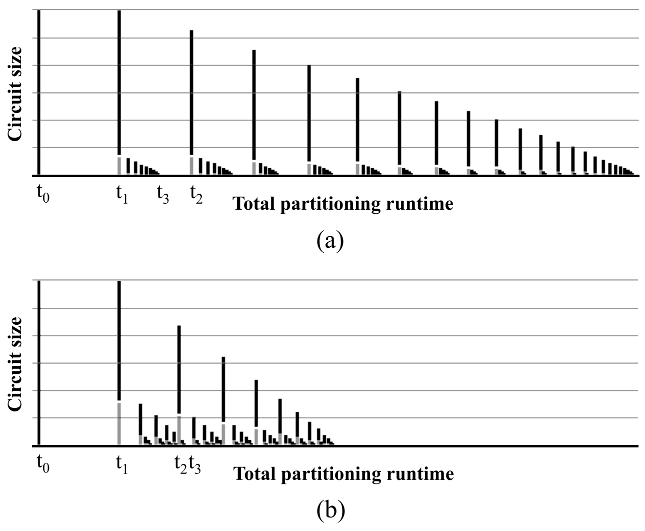


Fig. 7. Influence of the unbalance factor on the total required partitioning runtime. (a) UB 40. (b) UB 25.

cut ($Q1$) leads to an increase in CPD when compared to $Q2$. The difference between the parameters in $Q1$ and $Q2$ is the type of V-cycle refinement. The V-cycle refinement in $Q2$ thus leads to a lower CPD in the routed design. It is important to note that although a clear trend is observed, the differences in TWL and CPD are small. The quality setting $Q2$ is used as the optimal runtime-quality tradeoff. If fast packing is the main objective to use MULTIPART then it is useful to consider the fastest parameter combination $Q4$.

C. Unbalance Factor

The UB determines the maximum unbalance in number of blocks between both parts during the partitioning of a circuit. For an unbalance equal to UB, the boundaries of each subcircuit with size N_S are set by

$$N_O \frac{50 - \mathrm{UB}}{100} < N_S < N_O \frac{50 + \mathrm{UB}}{100} \quad (2)$$

with N_O the number of blocks in the original circuit. Thus the larger the value of UB, the larger the maximum allowed unbalance will be.

The best unbalance factor is a tradeoff between runtime and quality. First we explain why a large UB leads to an increase in the total partitioning runtime. Then we search for the best UB value by analyzing the runtime-quality tradeoff.

1) Partitioning Runtime: The larger the value of UB, the larger the total required partitioning runtime will be. This is due to the effect shown in Fig. 7. In this figure, the size of the partitioned subcircuits is shown in function of time for two values of the unbalance factor. The partitioning of the original circuit with a size equal to N starts on time instance t_0 and is finished at time t_1. In the example with UB equal to 40 [Fig. 7(a)], the partition leads to two subcircuits with a size between $0.1N$ and $0.9N$. The main objective of hMetis is minimizing the number of cut edges. It is highly probable that cutting through the center of the circuit will lead to a larger amount of cut edges. Consequently, it is likely that the

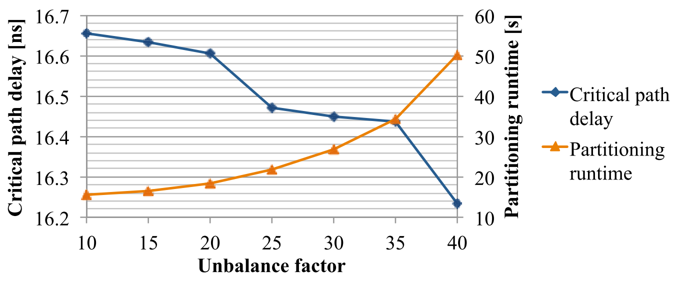


Fig. 8. Total partitioning runtime and CPD in function of the unbalance factor.

size of the subcircuits will be approximately equal to $0.1N$ and $0.9N$. Then two hMetis threads are launched to partition both subcircuits concurrently. The small subcircuit is quickly recursively bipartitioned to the threshold hierarchy level (t_3). For the large subcircuit, the process is repeated. Now the size of the circuit is equal to $0.9N$. At time t_2 the large subcircuit is partitioned into a small subcircuit, which is quickly partitioned to the threshold hierarchy level, and a large subcircuit, which is again partitioned into a small and a large subcircuit. This recursive process repeats itself until the threshold circuit size is reached for all subcircuits.

The partitioning runtime increases with an increasing unbalance factor because the partitioning runtime is dominated by the large part in each cut. In Fig. 7(b) the partitioning of the same circuit is shown for an unbalance factor of 25. In this case, less time is required to fully partition the circuit.

2) Post-Routing Total Wirelength and Critical Path Delay: Using a small unbalance in the partitioning step leads to fast runtimes but worse quality because the partitioning tool has less freedom. When the value of UB is large then the size boundaries of both subcircuits are large. Consequently, hMetis has a lot of freedom to cut the circuit into two parts. Mainly, the CPD is influenced by changing the unbalance factor. In timing-driven MULTIPART, timing edges with an increased weight are introduced to reduce the CPD (Section VII). hMetis tries to avoid cutting a critical edge because it leads to a large cut increase. When a small unbalance factor is used then it will be difficult to avoid cutting a critical edge due to the reduced freedom, leading to an increase in CPD (Fig. 8). The default unbalance factor in MULTIPART is set to 25. This value is a good tradeoff between the partitioning runtime and CPD.

D. Partition Depth: Optimizing the Number of Subcircuits

The recursive bipartitioning step ends when a certain hierarchy level is reached. To define this hierarchy level, a maximum cluster size (N_{max}) is set for all subcircuits. If the size of a subcircuit is smaller than this value then it is clustered with the seed-based packer. The smaller the value of N_{max}, the larger the number of subcircuits (N_{sub}) will be (Table I). There are a few conflicting motivations to determine the optimal number of subcircuits.

To improve quality, the number of subcircuits should be maximized. The deeper we go in the partition tree, the more information we have about the design hierarchy. This leads to better results for the TWL and CPD as shown in Table I.

TABLE I
INFLUENCE OF THE PARTITION DEPTH ON THE NUMBER OF SUBCIRCUITS, AREA (LAB COUNT), QUALITY, AND RUNTIME. ANALYZED FOR ALL ROUTABLE TITAN23 BENCHMARK DESIGNS

		Relative to baseline			Runtime [s]	
N_{max}	N_{sub}	LAB	TWL	CPD	Part	SBP
30000	11	1.001	0.77	0.93	27.8	84.6
17500	19	1.003	0.74	0.92	29.9	56.6
10000	34	1.005	0.71	0.92	30.6	43.7
7500	45	1.006	0.71	0.92	31.3	40.5
5000	68	1.008	0.69	0.91	31.7	37.4
3500	98	1.012	0.68	0.90	31.9	34.8
2500	134	1.016	0.68	0.90	31.9	33.4
1750	193	1.021	0.66	0.90	32.5	32.2
1250	275	1.027	0.66	0.90	32.2	32.6
1000	354	1.035	0.65	0.89	32.2	33.0
750	484	1.045	0.65	0.88	32.8	34.1
500	725	1.063	0.64	0.87	33.3	36.9

On the other hand, going deeper in the partition tree has two important drawbacks: 1) higher LAB usage and 2) an increased partitioning runtime. First, when a connection between two netlist primitives is cut then these primitives can not be packed into the same functional block. This leads to an increase in the number of required LABs (and area) as we go deeper in the design hierarchy because the deeper we go, the more connections between primitives are cut. Notice that going deeper in the hierarchy only results in an increase in the number of required LABs. The DSP and RAM hard block types are used efficiently because the hard block primitives are redistributed between both parts after each cut (Section V-B). Second, the partitioning runtime increases with decreasing N_{max} because more partitions are required until all subcircuits have a size smaller than N_{max}.

In addition to the two opposing forces discussed above, we should also consider the SBP runtime. Each thread that packs a subcircuit with the seed-based packer introduces an overhead. The main overhead is caused by launching and initializing AAPack. Thus the more subcircuits we have, the larger the overhead will be. However, this is not reflected in the SBP runtime when N_{max} ranges from 30 000 to 1750 (Table I). The runtime decreases as N_{sub} increases for two reasons: the usage of the available threads and the size of the subcircuits. First, if a small number of large subcircuits are available then it is not likely that all threads will be used efficiently because it is difficult to balance them over the available threads on a 10-core CPU. The impact of thread balancing is best noticed when N_{max} decreases from 30 000 to 10 000. Second, the runtime per block (i.e., the time required to pack one block in the circuit) decreases when smaller subcircuits are packed with AAPack [Fig. 9(b)]. The AAPack runtime thus scales badly with an increased size of the subcircuits. This results in a reduction of the total SBP runtime if the circuit is partitioned into a set of smaller subcircuits [Fig. 9(a)]. Notice in Fig. 9(a) that the large gain in total clustering runtime is partly nullified by the runtime overhead for each clustered subcircuit.

We conclude that N_{max} should be as small as possible. This way a maximum gain in TWL and CPD is achieved.

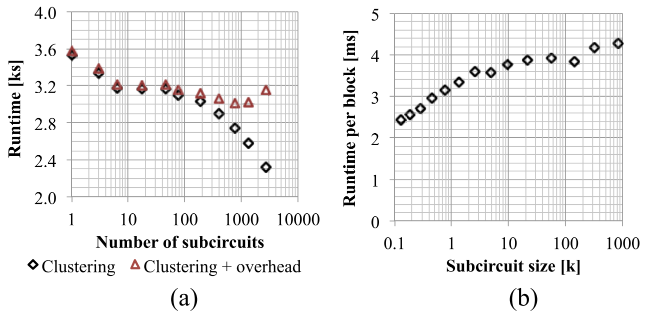


Fig. 9. Influence of the partition depth on the SBP runtime for *sparcT1_chip2*. In this simulation only one thread is used to avoid the thread balancing effect. (a) Total SBP runtime to cluster all subcircuits in function of the number of subcircuits. (b) Average runtime per block in function of the subcircuit size.

We choose as the minimal $N_{\max}$, the value where the gain in quality is justified by the increase in required area (LABs). The runtime is not much influenced by $N_{\max}$. The additional runtime overhead of the partitioning step is nullified by a gain in SBP runtime when $N_{\max}$ is larger than 1750. An $N_{\max}$ value of 2500 is used as the default value.

VII. Timing-Driven MultiPart

During static timing analysis the timing constraints are checked. A typical constraint is a lower limit for the clock frequency. To analyze when violations will occur, a timing graph is constructed. In this graph, we add information about the delays in the circuit. The arrival time of a signal is the time elapsed for a signal to arrive at a certain point, T_{arr}. The required time, T_{req}, is the latest time at which a signal can arrive at a node without timing violations. The slack of an edge with source node A and sink node B, is the difference between the required and the arrival time of those nodes, subtracted with the delay of that edge

$$\text{Slack}(A, B) = T_{\text{req}}(B) - T_{\text{arr}}(A) - \text{delay}(A, B). \tag{3}$$

A positive slack at a node implies that the arrival time at that node may be increased without affecting the maximum delay in the circuit. Conversely, a negative slack implies that a path is too slow, and the path must be sped up (or the reference signal delayed) if the timing constraints have to be met. The worst negative slack (WNS) indicates the critical path of the circuit. The timing violations will not improve unless this path is taken care of. To indicate how critical an edge is, a criticality measure is introduced

$$\text{Crit}_{\text{edge}} = 1 - \frac{\text{Slack}_{\text{edge}}}{\text{WNS}}. \tag{4}$$

MultiPart adds timing information to the partitioning and SBP step. The partitioning tool is unable to decide which edges are critical and which are not because the cost of cutting a connection is the same for all connections. The seed-based packer clusters all netlist primitives in the subcircuits, while minimizing the CPD of each subcircuit. It is not possible to minimize the delay of timing paths that span multiple subcircuits because they are cut on the edges between the subcircuits (Fig. 2). Table II shows the ratio of the CPD for the Titan23

TABLE II
CPD of the Titan23 Benchmark Designs With and Without Added Timing Information

	No timing			Timing-driven		
Name	CPD [ns]	rel	std	CPD [ns]	rel	std
neuron	10.0	1.02	0.5	9.9	1.02	0.4
sparcT1_core	9.2	0.96	0.6	9.3	0.98	1.0
stereo_vision	8.7	0.97	0.4	8.6	0.96	0.4
cholesky_mc	7.9	0.91	0.5	8.1	0.93	0.5
des90	12.6	0.88	1.2	11.6	0.81	0.8
SLAM_spheric	81.4	0.98	1.5	79.8	0.96	1.0
segmentation	828	0.92	10	784	0.87	3.4
bitonic_mesh	14.7	0.83	1.1	13.5	0.77	0.9
dart	19.0	1.16	1.4	15.6	0.95	0.8
openCV	11.2	0.97	1.1	11.0	0.96	0.7
stap_qrd	7.8	0.81	0.4	7.8	0.81	0.4
minres	8.9	0.98	0.7	8.6	0.94	0.6
cholesky_bdti	9.0	0.84	0.4	9.0	0.84	0.5
sparcT2_core	12.0	1.05	1.0	12.0	1.05	1.1
denoise	852	0.92	20	792	0.85	3.4
gsm_switch	11.5	1.00	1.5	11.5	1.00	1.4
mes_noc	13.5	0.66	1.1	13.9	0.67	1.3
sparcT1_chip2	25.2	1.08	2.6	23.5	1.01	1.5
Geomean		**0.93**	**1.2**		**0.90**	**0.9**

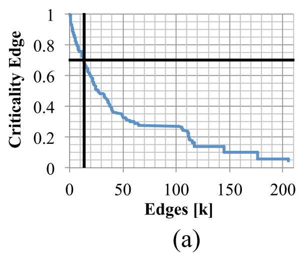
benchmark designs when compared to AAPack. We notice that the lack of timing information results in a large CPD for some of the designs. In this section, we explain the added timing functionality. The functionality does not only reduce the CPD for the problematic designs, on average a gain of 3% is obtained. The results shown in Table II are averaged over 20 iterations for each design. Timing-driven MultiPart also reduces the variance over those iterations.

A. Timing Information in the Partitioning Phase

During placement the clusters are assigned to a physical location on the architecture. The placer tries to minimize the TWL, so blocks with a higher interconnectivity are placed closer together. This results in a smaller distance between clusters that are close to each other in the partition tree. Consequently, edges on the critical or a near critical path should not be cut on high hierarchical levels because this leads to long and slow connections in the interconnection network.

To prevent cutting critical connections, we introduce timing edges to the circuit graph before it is passed to the partitioning tool. The timing edges are added in parallel with the critical connections in the design. They avoid that a critical or a near critical path is cut when a partition is possible without cutting this path. The amount of timing edges added and the weight of these edges are important parameters. Adding too many timing edges with too large weights results in partitions that violate the natural hierarchy of the design.

A static timing analysis determines where the timing edges should be added. For all connections we calculate the slack and its corresponding criticality. If a net has multiple sink nodes then all source-sink connections in the net are analyzed separately. No detailed timing information of the connections is



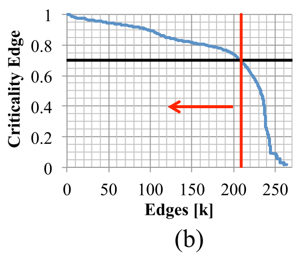
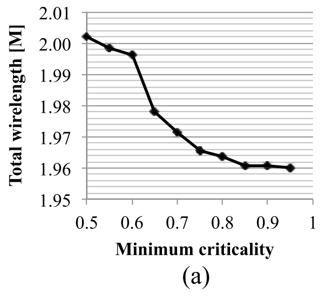


Fig. 10. Criticality of all connections in (a) *MCML* and (b) *EKF_SLAM*. The minimum criticality of *EKF_SLAM* is increased to ensure that maximum 20% of the connections are assigned as critical.



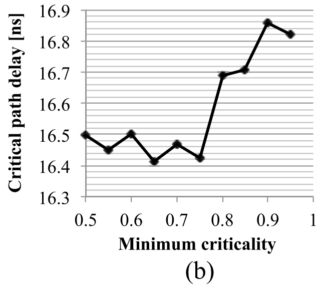


Fig. 11. Influence of the minimum criticality on (a) TWL and (b) CPD.

available after synthesis. Therefore, we optimistically assume that the fastest possible route is used between the primitive blocks in the netlist. A timing edge is added in parallel to each connection with a criticality larger than a predefined threshold value, the minimum criticality (C_{min}).

We distinguish two types of circuits to determine the minimum criticality: 1) circuits with only a few long paths [Fig. 10(a)] and 2) circuits with a more gradual path delay distribution [Fig. 10(b)]. For the circuits with only a few long paths the process is straightforward. All edges with a criticality larger than C_{min} are added. An example of such a circuit is *MCML* [Fig. 10(a)]. To find the best value for the minimum criticality, we analyzed the post-routing results for a range of minimum criticalities. Enough critical edges are added to the circuit to control the CPD as long as C_{min} is smaller than 0.8 [Fig. 11(b)]. As expected, a higher C_{min} reduces the TWL of the designs [Fig. 11(a)]. Less timing edges are added to the circuit graph if C_{min} increases, resulting in less distortion in the design hierarchy during partitioning. As the best value we choose 0.7, this way some safety margin is build in at the cost of a small wirelength increase. In Fig. 11(b), a C_{min} of 0.7 leads to a larger CPD when compared to a C_{min} of 0.75. This is due to the variance in results. The difference between the CPD for C_{min} equal to 0.7 and 0.75 is smaller than the noise margin.

However, some circuits have a more gradual path delay distribution. Adding a timing edge for every connection with a criticality larger than C_{min} would lead to a large amount of timing edges. For these circuits we avoid adding too many edges by considering only the 20% most critical edges. The minimum criticality for these circuits is thus increased in such a way that maximum 20% of the edges in the design have a criticality larger than this value [Fig. 10(b)].

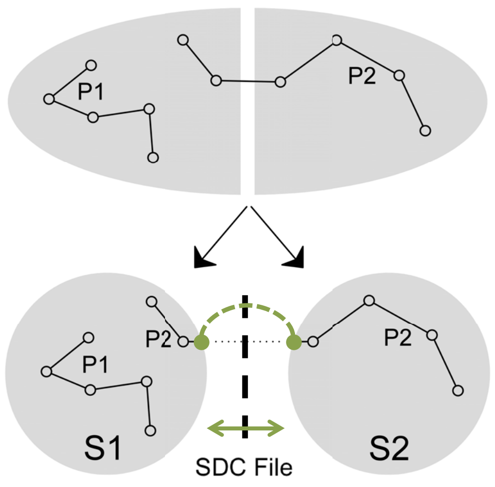


Fig. 12. Partitioning of a circuit with cut and uncut critical path. The cut critical path results in shorter paths in both subcircuits. Delay information is passed to the seed-based packer by making use of the SDC format.

All edges with a criticality larger than C_{min} are defined as critical. For each critical edge a timing edge is added to the circuit in parallel with the critical connection. A weight is assigned to each timing edge, in order to differentiate between the critical and near critical edges. The weight of a timing edge (W_{edge}) is proportional to the criticality of that edge (5). The larger the criticality of an edge, the larger its corresponding weight will be. The multiplication factor M in (5) is used for two reasons. First, hMetis only allows integer values for the weights [27]. The criticality of the edge is multiplied with M and rounded to the nearest integer value. Second, the timing edges should have a larger weight than the nets in the circuit. This way cutting a critical edge is delayed as much a possible. By analyzing post-routing results, we found that M should be equal to 10. This value of M results in the best CPD without deteriorating the natural hierarchy of the circuit

$$W_{\text{edge}} = \lfloor M * \text{Crit}_{\text{edge}} \rceil. \tag{5}$$

B. Timing Edge Weight Update

If a critical path is cut during recursive bipartitioning then this path should have special attention. Otherwise, it could be cut several times and lead to multiple slow connections in the interconnection network. To prevent this, all timing edges on a critical path have a very large weight assigned once the path is cut. This ensures that other uncut critical paths are cut first if no partition is possible without cutting critical edges.

C. Timing Information in the Seed-Based Packing Phase

For some circuits it is impossible to avoid that critical edges are cut. This introduces a problem because a cut critical path results in smaller paths in both subcircuits. In Fig. 12, the critical path P2 is cut during partitioning. This leads to two subcircuits where the cut critical edge is replaced with an input/output pin. When these subcircuits are clustered, the path

TABLE III
PROBLEMS WITH THE TITAN23 BENCHMARK DESIGNS

Name	Problem
LU230	hMetis partitioning fails during coarsening phase
LU_Network	This benchmark can not be packed with VPR [30]
directrf	Minimum required channel width is larger than 600
bitcoin_miner	Minimum required channel width is larger than 600
gaussianblur	This benchmark can not be placed with VPR [31]

P2 will not be considered as critical by the seed-based packer because now it is shorter than in the original circuit.

To solve this problem, additional information is passed on to the seed-based packer. We use the synopsis design constraints format in our implementation [28]. In this format, it is possible to assign a delay to any input and output pin of the circuit. In MULTIPART, these delays are added to all input/output pins that result from a cut critical edge. This way, the seed-based packer knows the total delay of all critical paths in the circuit and is able to minimize the maximum delay.

VIII. EXPERIMENTS

A. Methodology

The Titan23 suite is used for benchmarking. The target device for the Titan23 benchmark designs is Altera's Stratix IV FPGA. The experiments are performed on a workstation with an Intel E5-2660v3@2.6 GHz (10 cores) and with 128 GB work memory. Post-routing TWL and CPD are obtained by placing and routing the packed circuits with VPR 7.0.7 r75b47d3. MULTIPART and AAPack are compared on the same architecture. The CW and dimensions of the FPGA are optimized for each design separately. The CW is chosen in such a way that medium stress routing is required for the designs. This way a fair comparison is possible between AAPack and MULTIPART.

All experiments are run 20 times, each time with a different seed for placement. The results shown below are averaged out over these iterations. We were not able to use all designs in the Titan23 suite. From the 23 available designs, 18 are successfully placed and routed. The problems with the five remaining designs are shown in Table III. The design *sparcT2_core* is used, but it has a problem with the clustering of M9K blocks during the SBP step with AAPack [29]. This leads to an increase in the number of required M9K blocks (Table IV).

The area requirements, quality metrics and runtime of MULTIPART are compared to the state-of-the-art academic packer AAPack. Area results are shown in Table IV. The quality metrics and runtime are given in Table V.

B. Area

MULTIPART leads to a minor average increase in the total number of required LABs because the weakly interconnected subcircuits are clustered independently. Over all designs, this leads to an LAB increase of 1.6%. The LAB increase is relative to the number of LABs after packing with AAPack, which is efficient at densely packing the designs with the seed-based bottom-up approach. MULTIPART outperforms AAPack for some designs. Better results are obtained because MULTIPART has a top-down approach with a global overview of the design. This is an advantage, for example, for circuits with long combinational paths (*segmentation* and *denoise*) because these paths span over a large amount of LABs. The greedy clustering approach of AAPack lacks a global overview of these long paths. On the other hand, we notice that a large LAB increase is introduced for all *sparc* designs, which are μP cores. The LAB increase depends on the type of application, its corresponding connectivity, and design hierarchy.

In contrast to the LAB increase, less area is required for the hard blocks due to the high quality of the prepartitioning step and the hard block swap. The number of required DSP blocks is exactly the same in MULTIPART and AAPack. The total area of the RAM blocks is reduced with 2.6%. This partly nullifies the area increase of the LABs. In total 1.1% more area is required to implement all netlist primitives.

TABLE IV
AREA: NUMBER OF LAB AND RAM BLOCKS

	Baseline			MultiPart		
Name	LAB	RAM		LAB	RAM	
		9K	144K		9K	144K
neuron	3166	185	0	3219	185	0
sparcT1_core	3747	131	0	3886	131	0
stereo_vision	2759	135	0	2891	135	0
cholesky_mc	4697	444	16	4738	444	16
des90	3955	860	0	4018	860	0
SLAM_spheric	6439	87	0	6474	87	0
segmentation	8431	440	28	8253	424	4
bitonic_mesh	6841	1664	0	6938	1664	0
dart	6605	530	0	6625	530	0
openCV	6501	787	40	6746	787	40
stap_qrd	15715	553	0	15662	553	0
minres	7231	1208	66	7391	1201	32
cholesky_bdti	9621	600	0	9545	600	0
sparcT2_core	13041	266	0	13762	290	0
denoise	17820	367	0	17512	366	0
gsm_switch	19084	1848	0	19028	1848	0
mes_noc	24870	800	0	25457	800	0
sparcT1_chip2	30917	530	0	32643	530	0

C. Quality

The main goal of MULTIPART is reducing the TWL by preserving the natural hierarchy in the circuit during packing. Over all designs, a gain of 32% in TWL is obtained. The gain ranges from 10% up to 48%. The large gain is achieved because the connections in the interconnection network are shorter and is not due to a lower number of point-to-point connections in the clustered design. There is a slight increase in the number of routed nets (1.2%), but the nets have a smaller average fanout. The reduction in TWL has some major advantages.

1) Minimum Channel Width: The area of the routing network in an FPGA is proportional to the CW and requires a large amount of the total FPGA area. Because of the improved packing algorithm less wires are required to route the designs, leading to a smaller minimum required channel width (MCW).

TABLE V
QUALITY AND RUNTIME OF AAPACK AND MULTIPART FOR THE TITAN23 BENCHMARK DESIGNS

	Baseline					**MultiPart**				
	Quality		**Runtime**			**Quality**		**Runtime**		
Name	**TWL [M]**	**CPD [ns]**	**Pack [s]**	**Place [s]**	**Route [s]**	**TWL [M] (rel)**	**CPD [ns] (rel)**	**Pack [s] (rel)**	**Place [s] (rel)**	**Route [s] (rel)**
neuron	1.24	9.7	156	206	230	0.82 (0.66)	9.9 (1.02)	27 (0.17)	200 (0.97)	189 (0.82)
sparcT1_core	1.64	9.5	385	282	483	1.14 (0.69)	9.3 (0.98)	54 (0.14)	248 (0.88)	155 (0.32)
stereo_vision	0.99	9.0	112	240	169	0.70 (0.71)	8.6 (0.96)	18 (0.16)	219 (0.91)	149 (0.88)
cholesky_mc	1.51	8.7	235	331	303	1.18 (0.78)	8.1 (0.93)	37 (0.16)	303 (0.92)	288 (0.95)
des90	3.54	14.3	463	526	570	2.50 (0.71)	11.6 (0.81)	51 (0.11)	508 (0.97)	482 (0.84)
SLAM_spheric	2.77	83.4	488	735	510	2.03 (0.73)	79.8 (0.96)	62 (0.13)	733 (1.00)	362 (0.71)
segmentation	2.86	900	721	1390	610	2.37 (0.83)	784 (0.87)	69 (0.10)	1455 (1.05)	352 (0.58)
bitonic_mesh	7.48	17.7	949	1158	2161	4.89 (0.65)	13.5 (0.77)	96 (0.10)	1012 (0.87)	922 (0.43)
dart	4.28	16.4	691	789	2901	2.24 (0.52)	15.6 (0.95)	79 (0.11)	701 (0.89)	508 (0.17)
openCV	5.46	11.5	716	951	1587	3.59 (0.66)	11.0 (0.96)	67 (0.09)	878 (0.92)	1067 (0.67)
stap_qrd	4.11	9.6	650	1671	644	2.41 (0.59)	7.8 (0.81)	57 (0.09)	1691 (1.01)	488 (0.76)
minres	4.95	9.1	695	974	1063	2.90 (0.59)	8.6 (0.94)	65 (0.09)	873 (0.90)	706 (0.66)
cholesky_bdti	3.55	10.7	634	1048	677	2.61 (0.73)	9.0 (0.84)	75 (0.12)	951 (0.91)	581 (0.86)
sparcT2_core	5.64	11.5	1553	1849	908	4.17 (0.74)	12.0 (1.05)	156 (0.10)	1768 (0.96)	594 (0.65)
denoise	5.34	927	1788	4141	900	4.79 (0.90)	792 (0.85)	153 (0.09)	3984 (0.96)	722 (0.80)
gsm_switch	10.1	11.5	2296	2985	1794	5.54 (0.55)	11.5 (1.00)	177 (0.08)	2555 (0.86)	1436 (0.80)
mes_noc	9.06	20.6	2776	5079	2758	6.12 (0.68)	13.9 (0.67)	235 (0.08)	4506 (0.89)	2446 (0.89)
sparcT1_chip2	12.1	23.3	4553	5245	4423	7.00 (0.58)	23.5 (1.01)	352 (0.08)	4625 (0.88)	1460 (0.33)
Geomean	**3.86**	**22.0**	**721**	**1046**	**871**	**2.61 (0.68)**	**19.9 (0.90)**	**78 (0.11)**	**971 (0.93)**	**543 (0.62)**

TABLE VI
MCW OF FIVE SMALL TITAN23 DESIGNS

	Baseline		**MultiPart**	
Name	**TWL [M]**	**MCW**	**TWL [M] (rel)**	**MCW (rel)**
neuron	1.26	205	0.81 (0.65)	109 (0.53)
sparcT1_core	1.65	290	1.13 (0.69)	198 (0.68)
stereo_vision	0.99	191	0.70 (0.71)	109 (0.57)
cholesky_mc	1.50	185	1.19 (0.79)	127 (0.69)
SLAM_spheric	2.77	328	2.04 (0.73)	214 (0.65)
Geomean	**1.54**	**233**	**1.09 (0.71)**	**145 (0.62)**

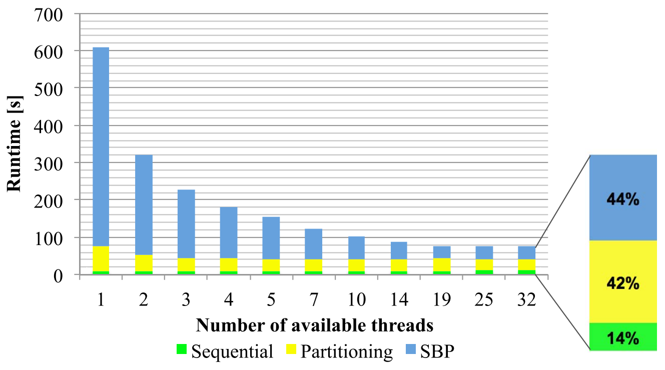


Fig. 13. Geomean runtime of MULTIPART in function of the number of available threads.

In Table VI the gain is shown for five small Titan23 designs. Small designs are used because finding the MCW for the large designs leads to extremely long runtimes due to the large number of required routing iterations. The gain in MCW is larger than the gain in TWL for these five designs. On average a gain of 38% is obtained. This result is important, because it means that smaller and thus cheaper FPGAs can be used to implement the applications. It also completely justifies the small area increase of the required functional blocks.

Additionally, this also increases the usability of the Titan23 suite. With AAPack five of the used Titan23 designs fail to route with the default Stratix IV architecture targeted. In this architecture, the CW is equal to 300. If MULTIPART is used then all designs can easily be routed without congestion problems on the default architecture.

2) Routing Runtime: The gain in TWL results in less congestion. It is easier to find a routable routing solution. Consequently, a gain of 38% in the total routing runtime is achieved. MULTIPART thus not only has faster packing runtimes, it also greatly reduces the routing runtime.

3) Maximum Clock Frequency: There is a gain of 10% for the CPD due to the improved design hierarchy after packing.

D. Packing Runtime

The packing runtime for all designs is shown in Table V. An average runtime speed-up of 9.3× is achieved on a CPU with 10 cores. It is interesting to note that the geomean of the AAPack runtime is equal to 721 s, while the single-threaded version of MULTIPART only needs 611 s (Fig. 13). Even though the partition step introduces a runtime overhead, single-threaded MULTIPART is faster than AAPack for two reasons. First, less runtime is required to cluster the subcircuits because a design specific architecture is used (Section V-A3). Second, the total SBP runtime is lower because a set of small subcircuits is clustered, leading to a smaller runtime per block [Fig. 9(a)].

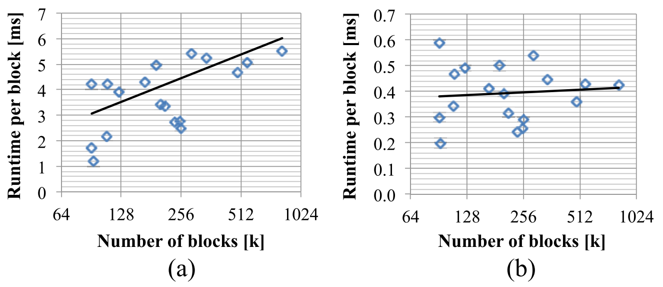


Fig. 14. Scalability in size of the designs for (a) AAPack and (b) MULTIPART: absolute runtime per block and linear interpolation of the values.

1) Scalability in Number of Threads: The partitioning-based methodology offers an opportunity for multithreaded parallelism in the partitioning and SBP step. In Fig. 13 the total runtime of MULTIPART is shown in function of the number of available threads. The tool consists of three main parts: 1) a sequential; 2) a partitioning; and 3) an SBP part. The sequential part consists of the static timing analysis and the prepartitioning steps. Its relative importance increases when more threads are used. The SBP step requires no synchronization between the concurrently packed subcircuits. This leads to a large runtime gain if more threads are used. If the number of threads increases from 1 to 19 then 15.6× faster runtimes are obtained. Further increasing the number of threads does not influence the runtime on the used 10-core CPU. The partitioning step is parallelized by splitting the subcircuits independently. Partitioning does not scale well with an increasing number of available threads due to hierarchy dependencies and post-partitioning runtime overheads. A maximum runtime speed-up of 2.1× is reached when four threads are used. Hierarchy dependencies result from the fact that new subcircuits are only available when the subcircuits on the previous hierarchy level are split and processed. The post-partitioning runtime overhead is introduced by the hard block swap (Section V-B) and the timing edge weight update (Section VII-B). MULTIPART thus achieves large runtime speed-ups when more threads are used, but is not able to fully exploit CPUs with a large number of threads. Increasing the number of threads from one to five leads to four times faster runtimes. Further increasing the number of threads to 19 leads to a maximum gain of 8× (Fig. 13).

2) Scalability in Size of the Designs: The runtime per block in function of the design size is shown for the Titan23 designs in Fig. 14. For AAPack there is a clear increase in the runtime per block when the size of the designs increases. This behavior is also noticed in Fig. 9(b) where the runtime per block is analyzed for different sizes of the subcircuits after partitioning. MULTIPART has a much better scalability with increasing design size because the circuit is hierarchically partitioned into a set of small subcircuits. The number of subcircuits increases linearly with the size of a design. Partitioning also scales well with increasing design size. Only one additional partitioning step is required to split the circuit in two subcircuits if the size of the design doubles. Both resulting subcircuits are then further partitioned independently.

The designs in Table V are organized from small to large. Due to the better scaling the gain in packing runtime is larger if the design size increases. The smallest gain is equal to 5.8× (*neuron*) and increases to 12.9× (*sparcT1_chip2*).

IX. CONCLUSION

In this paper, we presented an extended version of MULTIPART. MULTIPART is a partitioning-based packing tool that consists of two phases: 1) a partitioning phase and 2) an SBP phase. The hierarchical packing approach obtains better quality results in less runtime at the cost of a small area increase because more logic blocks are required. MULTIPART is able to obtain faster runtimes (9.3×) because we can easily implement multithreading without consequences for the quality. The quality of the results improves because the design hierarchy is preserved during packing. The quality of packing has a great influence on the post-routing TWL and CPD. On average MULTIPART leads to a gain of 32% in TWL and 10% for the CPD. This closes a major part of the gap between commercial and academic results in terms of TWL. Murray *et al.* [1] reported a TWL gap of 2.19× comparing VPR with Altera Quartus II. In this paper, where the AAPack packer is replaced with MULTIPART, we greatly reduced this gap by just replacing the packing algorithm. As a large reduction in TWL is obtained, it is possible to route the designs on an architecture with a smaller channel width. Such an architecture requires less routing resources and is thus smaller and cheaper. The source code of MULTIPART is released on Github in our FPGA CAD framework [32].

In the future we would like to investigate if we could reuse the information that is obtained in the partitioning step. Currently, this information is not used during placement and routing. In the future, we want to use the hierarchical information to provide a better initial placement for the functional blocks.

ACKNOWLEDGMENT

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government - department EWI.

REFERENCES

[1] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial CAD," *ACM Trans. Reconfig. Technol. Syst.*, vol. 8, no. 2, 2015, Art. no. 10.

[2] A. Ludwin, V. Betz, and K. Padalia, "High-quality, deterministic parallel placement for FPGAs on commodity hardware," in *Proc. 16th Int. ACM/SIGDA Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 2008, pp. 14–23.

[3] G. Jain *et al.*, "Multi-threaded deterministic router," U.S. Patent 8 671 379, Mar. 11, 2014. [Online]. Available: https://www.google.com/patents/US8671379

[4] A. Marquardt, V. Betz, and J. Rose, "Speed and area tradeoffs in cluster-based FPGA architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 1, pp. 84–93, Feb. 2000.

[5] A. S. Marquardt, V. Betz, and J. Rose, “Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density,” in *Proc. ACM/SIGDA 7th Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 1999, pp. 37–46.

[6] J. Luu, J. Rose, and J. Anderson, “Towards interconnect-adaptive packing for FPGAs,” in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Monterey, CA, USA, 2014, pp. 21–30.

[7] J. Luu *et al.*, “VTR 7.0: Next generation architecture and CAD system for FPGAs,” *ACM Trans. Reconfig. Technol. Syst.*, vol. 7, no. 2, 2014, Art. no. 6.

[8] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, “Efficient circuit clustering for area and power reduction in FPGAs,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 7, no. 4, pp. 643–663, 2002.

[9] E. Bozorgzadeh, S. O. Memik, X. Yang, and M. Sarrafzadeh, “Routability-driven packing: Metrics and algorithms for cluster-based FPGAs,” *J. Circuits Syst. Comput.*, vol. 13, no. 1, pp. 77–100, 2004.

[10] M. Tom, D. Leong, and G. Lemieux, “Un/DoPack: Re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2006, pp. 680–687.

[11] H. Liu and A. Akoglu, “Timing-driven nonuniform depopulation-based clustering,” *Int. J. Reconfig. Comput.*, vol. 2010, Jan. 2010, Art. no. 3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863644

[12] J. Lamoureux and S. J. E. Wilton, “On the interaction between power-aware FPGA CAD algorithms,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2003, pp. 701–708.

[13] L. Easwaran and A. Akoglu, “Net-length-based routability-driven power-aware clustering,” *ACM Trans. Reconfig. Technol. Syst.*, vol. 4, no. 4, 2011, Art. no. 38.

[14] S. T. Rajavel and A. Akoglu, “MO-Pack: Many-objective clustering for FPGA CAD,” in *Proc. 48th Design Autom. Conf.*, San Diego, CA, USA, 2011, pp. 818–823.

[15] M. Yang, J. Lai, and J. Tong, “Yet another many-objective clustering (YAMO-Pack) for FPGA CAD,” in *Proc. 23rd Int. Conf. Field Program. Logic Appl.*, Porto, Portugal, 2013, pp. 1–4.

[16] J. Cong and M. Romesis, “Performance-driven multi-level clustering with application to hierarchical FPGA mapping,” in *Proc. Design Autom. Conf.*, Las Vegas, NV, USA, 2001, pp. 389–394.

[17] M. E. Dehkordi and S. D. Brown, “Performance-driven recursive multilevel clustering,” in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, Tokyo, Japan, 2003, pp. 262–269.

[18] C. Sze, T.-C. Wang, and L.-C. Wang, “Multilevel circuit clustering for delay minimization,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 7, pp. 1073–1085, Jul. 2004.

[19] Z. Marrakchi, H. Mrabet, and H. Mehrez, “Hierarchical FPGA clustering based on multilevel partitioning approach to improve routability and reduce power dissipation,” in *Proc. ReConfig*, Puebla, Mexico, 2005, p. 25.

[20] W. Feng, “K-way partitioning based packing for FPGA logic blocks without input bandwidth constraint,” in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Seoul, South Korea, 2012, pp. 8–15.

[21] W. Feng, J. Greene, K. Vorwerk, V. Pevzner, and A. Kundu, “Rent’s rule based FPGA packing for routability optimization,” in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Monterey, CA, USA, 2014, pp. 31–34.

[22] D. T. Chen, K. Vorwerk, and A. Kennings, “Improving timing-driven FPGA packing with physical information,” in *Proc. Int. Conf. Field Program. Logic Appl.*, Amsterdam, The Netherlands, 2007, pp. 117–123.

[23] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, “Runtime-quality tradeoff in partitioning based multithreaded packing,” in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, Lausanne, Switzerland, 2016, pp. 1–9.

[24] B. S. Landman and R. L. Russo, “On a pin versus block relationship for partitions of logic graphs,” *IEEE Trans. Comput.*, vol. C-20, no. 12, pp. 1469–1479, Dec. 1971.

[25] *Stratix IV Device Handbook*, Altera Corporat., San Jose, CA, USA, Jan. 2016. [Online]. Available: https://www.altera.com

[26] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, “Multilevel hypergraph partitioning: Applications in VLSI domain,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 69–79, Mar. 1999.

[27] G. Karypis and V. Kumar, “hMETIS 1.5: A hypergraph partitioning package,” Dept. Comput. Sci. Eng. Army HPC Res. Center, Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep., Nov. 1998. [Online]. Available: http://www.cs.umn.edu/metis

[28] J. Luu *et al.*, *VPR User’s Manual (Version 7.0)*, Dept. Elect. Comput. Eng., Univ. at Toronto, Toronto, ON, Canada, Tech. Rep., 2013.

[29] *Sparct2_Core GitHub Issue*. Accessed on Oct. 2016. [Online]. Available: https://github.com/ verilog-to-routing/vtr-verilog-to-routing/issues/160

[30] *LU_Network GitHub Issue*. Accessed on Aug. 2016. [Online]. Available: https://github.com/ verilog-to-routing/vtr-verilog-to-routing/issues/111

[31] *Gaussian Blur GitHub Issue*. Accessed on Aug. 2016. [Online]. Available: https://github.com/verilog-to-routing/vtr-verilog-to-routing/issues/115

[32] E. Vansteenkiste, D. Vercruyce, and S. Lenders. (2016). *FPGA CAD Framework: MultiPart and Liquid*. [Online]. Available: https://github.com/EliasVansteenkiste/FPGA-CAD-Framework

Dries Vercruyce (GS’16) received the Master of Applied Science degree in electronics from Ghent University, Ghent, Belgium, in 2015, where he is currently pursuing the Ph.D. degree from the Department of Electronics and Information Systems, Computer Systems Laboratory.

He has a special interest in the back end of the FPGA CAD tool flow. His current research interest includes improving the quality of the configuration by providing hierarchy information of the design.

Elias Vansteenkiste (GS’13) received the Master of Applied Science degree in electronics from Ghent University, Ghent, Belgium, in 2011 and the Ph.D. degree from the Department of Electronics and Information Systems, Computer Systems Laboratory, Ghent University in 2016.

His current research interests include hardware acceleration for FPGA CAD algorithms and convolutional networks and new deep learning techniques for high dimensional data.

Dirk Stroobandt (M’98) received the Ph.D. degree from Ghent University, Ghent, Belgium, in 1998.

He is currently a Professor with the Department of Electronics and Information Systems and Computer Systems Laboratory, Ghent University. He currently leads the Hardware and Embedded Systems Research Group of about ten people, interested in semi-automatic hardware design methodologies and tools, runtime reconfiguration, and reconfigurable multiprocessor networks.

Dr. Stroobandt was a recipient of the ACM/SIGDA Outstanding Doctoral Thesis Award in Design Automation in 1999. He has also been initiated and co-organized the International Workshop on System-Level Interconnect Prediction since 1999. He has been the Associate Editor and Special Issue Guest Editor of a few international journals.