technologies

Article

Improved Parallel Legalization Schemes for Standard Cell Placement with Obstacles †

Panagiotis Oikonomou [1,*], Antonios N. Dadaliaris [1], Kostas Kolomvatsos [2], Thanasis Loukopoulos [3], Athanasios Kakarountas [3] and Georgios I. Stamoulis [4]

1 Computer Science, University of Thessaly, 35131 Lamia, Greece; dadaliaris@uth.gr
2 Informatics and Telecommunications, University of Athens, 106 79 Athens, Greece; kostasks@di.uoa.gr
3 Computer Science and Biomedical Informatics, University of Thessaly, 35100 Lamia, Greece; luke@dib.uth.gr (T.L.); kakarountas@dib.uth.gr (A.K.)
4 Electrical and Computer Engineering, University of Thessaly, 382 21 Volos, Greece; georges@uth.gr
* Correspondence: paikonom@uth.gr; Tel.: +30-223-106-6713

† This paper is an extended version of our paper published in A Tetris-based Legalization Heuristic for Standard Cell Placement with Obstacles. In Proceedings of the 7th International Conference on Modern Circuits and Systems Technologies (MOCAST 2018), Thessaloniki, Greece, 7–9 May 2018.

Received: 9 November 2018; Accepted: 19 December 2018; Published: 22 December 2018

Abstract: In standard cell placement, a circuit is given consisting of cells with a standard height, (different widths) and the problem is to place the cells in the standard rows of a chip area so that no overlaps occur and some target function is optimized. The process is usually split into at least two phases. In a first pass, a global placement algorithm distributes the cells across the circuit area, while in the second step, a legalization algorithm aligns the cells to the standard rows of the power grid and alleviates any overlaps. While a few legalization schemes have been proposed in the past for the basic problem formulation, few obstacle-aware extensions exist. Furthermore, they usually provide extreme trade-offs between time performance and optimization efficiency. In this paper, we focus on the legalization step, in the presence of pre-allocated modules acting as obstacles. We extend two known algorithmic approaches, namely Tetris and Abacus, so that they become obstacle-aware. Furthermore, we propose a parallelization scheme to tackle the computational complexity. The experiments illustrate that the proposed parallelization method achieves a good scalability, while it also efficiently prunes the search space resulting in a superlinear speedup. Furthermore, this time performance comes at only a small cost (sometimes even improvement) concerning the typical optimization metrics.

Keywords: standard cell placement; cell legalization; obstacles; Abacus; Tetris; parallelization.

1. Introduction

In the cell placement problem, an input circuit must be placed over a chip area so that the circuit's cells do not overlap, and one or more target functions are optimized. Typical optimization targets considered include the following: total wirelength, routability, cell congestion, and so on. A common problem statement involves cells of a standard height (different width) that must be placed on a chip area that is split into standard height rows (capturing power grid lines). The problem is usually tackled in a step-wise fashion. At a first iteration, a global placement algorithm spreads the cells over the chip area, so that the area coverage and targeted optimization goals are achieved. The resulting cell positions might be unaligned to the chip standard rows. Thus, in a second step, a legalization algorithm is responsible for achieving cell–row alignment and alleviating all cell overlaps. Assuming an efficient global placement, the legalization step must be performed with as few changes to the original assignment as possible. Thus, the aggregated cell distance between the global and final placement (cell displacement) is usually considered as the performance metric for legalization algorithms.

While extnsive literature exists on legalization schemes for standard cell placement (discussed in Section 2), the case where obstacles exist in the chip area has received less attention. Such obstacles might be the result of preplaced modules in the chip area at fixed positions, and may introduce additional constraints whereby cells cannot overlap with the obstacle areas. In this paper, we turn our attention to the legalization step in standard cell placement with obstacles. In particular, we propose and evaluate extensions for two well-known legalization algorithms, namely Tetris [1] and Abacus [2], which were originally designed to work for the case where no obstacles exist. The aforementioned algorithms account for different trade-offs between running time and optimization efficiency, with Abacus producing a better solution quality, but at a significantly higher computational overhead. The targeted extensions aim at making the algorithms obstacle-aware. Furthermore, in order to tackle the high running time of Abacus, we propose and evaluate a parallelization approach based on multi-threaded execution, whereby each thread handles a non-overlapping chip area partition. It turns out that the proposed parallelization method not only manages to reduce the running time of Abacus (Tetris too), but also does so without affecting the quality of the final placement (even improves it in some scenarios, particularly in the Tetris case). The initial results for the Tetris algorithm were presented in the literature [3]. Here, we extend and consolidate our previous work so as to account for the obstacle awareness and parallelization of the Abacus algorithm, which consists our primary contribution. Through experiments based on ibm circuits, the merits of our contributions are illustrated. Specifically, compared to the baseline Abacus algorithm, the parallel obstacle aware algorithm we propose (poAbacus) achieves a similar quality, but at a speedup that can reach 66 x with 12 cores.

The rest of the paper is organized as follows. Section 2 provides an overview of the research on the cell placement problem, as well as the legalization methods. Section 3 describes the proposed algorithms for obstacle aware legalization, while Section 4 presents the experimental results. Finally, Section 5 discusses the findings from the experiments and concludes the paper.

2. Related Work

Placement, routing, and the posterior layout generation of an integrated circuit, whether the design at hand is purely analog, strictly digital, or mixed, has been at the forefront of physical design research. This effort focuses on digital standard cell designs and their placement modelled in a 2D plain. It should be noted that complex gates have been proposed as a valuable alternative in terms of area and delay [4]. From a transistor-layer point of view, multiple design frameworks were proposed in order to achieve area efficient layouts of radiation hardened devices [5], or highly dense integrated circuits (ICs) [6].

Concerning standard cell placement legalization, the Tetris algorithm [1] first orders cells along their x-axis. Then, starting with the cell of the minimum x-axis coordinate, it places each cell to the first closest, leftmost available position. To do this, all of the possible candidate row positions are checked and the final decision is taken in a greedy manner. Tetris is a particularly fast method, albeit the final placement quality that is achieved is not on par with its counterparts. Efforts to improve the basic scheme include, for instance [7], where various heuristics were evaluated aiming at restricting the allowable displacement across x, y, or both axes. The authors also evaluated various heuristic combinations involving leftward and rightward cell movement. Extensions of the basic Tetris scheme to account for obstacles are presented in the literature [3]. The core idea is to split a standard row into sub-rows, defined by obstacle boundaries. The algorithm then scans all of the sub-row candidate positions in order to identify the final cell placement. In the literature [8], another obstacle-aware alteration was introduced for Tetris, whereby the cell selection order depended on the cell width instead of the x-axis position.

The Abacus legalizer proposed in the literature [2] works in a different manner compared with Tetris. Abacus places each cell into its optimal row position, starting from the cell of the minimum x-axis coordinate. If during the process a cell overlap occurs, a cluster of cells is formed and the whole cluster's best position is calculated through quadratic optimization. Both Tetris and Abacus were adopted by cell placement suites. Kraftwerk2 [9] uses a Tetris-like procedure for its legalization phase, while NTUplace3 [10] utilizes a similar scheme both as a look-ahead legalization approach during the global placement step, and as a final legalizer. Abacus is used in the density-aware detailed placement flow in the literature [11] in order to legalize the placement instances produced after each cell swap is performed by the detailed placer. In the literature [12], a parallel version of the Abacus algorithm is implemented and evaluated. Parallelization was achieved by spawning multiple threads in order to evaluate candidate row positions. The achieved speedup with four cores was shown to be roughly 2.5. Here, we follow an alternative approach, inspired by the authors of [13], for the case where no obstacles exist. The approach was based on vertically dividing the chip area into partitions to be treated by different threads. In this paper, we evaluate a variety of partitioning options. At the same time, we factor the presence of obstacles in the partitioning formation process.

The authors of [14] pinpoint the deficiencies of Abacus, and present modifications and extensions of its functionality in order to handle mixed-height standard cell designs. In the literature [15], the authors tackle placement induced problems, such as pin shorts and pin access, by adopting a look-ahead legalization procedure that ensures the existence of sufficient white space among cells. Abax [16] is another legalizer contrived from Abacus. While it retains the main functionality of Abacus, Abax adds hard-macro/blockage handling capabilities and look-ahead legalization during the global placement step, tailored to suit the minimization of the mean displacement function.

Legalization in FastPlace 3.0 [17] has two distinct steps, one concerning macro-blocks and another concerning standard cells. In the first step, the overlaps among the macro-blocks are resolved by repositioning them to their nearest legal position. In the second step, the remaining standard cells are assigned to a legal position in specific bins, based on the wirelength reduction caused by their relocation and on the density target of the bin. A Tetris-like legalization scheme was used by ePlace [18] and its subsequent extensions for mixed-size designs (e.g., [19,20]).

Dragon2005 [21] performs min-cut multi-way partitioning using hMetis [22] to spread the cells in the chip area. In the case of a macro-free design, the cells are placed one after the other inside a row, starting from the left edge, and proceeding to the right. When the design contains macro-blocks, placement is governed by a mutated cost function that takes into account the legality of each movement and the solution quality deterioration. SimPL [23] performs the approximate legalization during its global placement phase. A uniform grid is used in order to identify the locations that present the highest amount of overlaps. Subsequently, the cells associated with the overlaps are re-positioned while preserving the relative order.

The legalizer presented in the literature [24] is an integral part of BonnPlace [25]. As a first step, cell assignment to a set of predefined bins is performed, which might lead to overflows concerning cell number/density. In order to eradicate this effect and achieve balance, a cell flow is computed between bins. The main characteristic of the aforementioned procedure is that the flow augmentations that prevail and are subsequently realized, are only those that lead to feasible solutions.

In the literature [26], the legalization procedure is comprised of three stages. In the first stage, cells are aligned within sites, following a width descending order. Subsequently, an optimal position for each illegally placed cell (in a pin amount descending order) is identified within a specific search window. Finally, the cells are ordered based on their center coordinate, and the white space of each row is distributed accordingly in order to remove the remaining overlaps. The detailed placer in the literature [27] incorporates a collection of steps established in previous legalizers. More specifically, a cyclic flow is presented comprised of cell swapping, cell re-ordering, and cell bloating and refinement. The approach in the literature [28] targets designs containing multi-row height cells subject to additional hard constraints in the form of fences. The cells are legalized sequentially by checking for an optimal position on a predefined window around their global placement generated positions. Detailed placement is also performed by two separate network-flow-based optimizations concerning total displacement and cell ordering. Eh?Legalizer [29] approaches the legalization procedure as a network flow problem as well, but also abides by the layout-related technology constraints such as fence regions and cell edge spacing rules. This method leads to minimized maximum and average cell perturbation in the competitive runtime. The first is achieved by incorporating an additional maximum movement constraint during the search for feasible paths and the cell movement along them, while the second goal is achieved by pinpointing the candidate paths where moving cells deflate overflowed bins. In the literature [30], throughout the iterations of the algorithm, the legalization problem is dynamically formulated in order to encompass an additional constraint in the form of a history file, keeping track of the cell movements that are highly probable to cause illegal instances.

In the literature [31], a mechanism for legalization utilizing *k*-d tree data structures is proposed. A modified k-d tree construction algorithm is applied, which leads to the formulation of data independent (and thus algorithmic agnostic) partitions. Subsequently, the overall legalization procedure can be accelerated, because of the reduced problem size and the parallel execution of any legalization algorithm in each of the partitions. Recent advancements in the design and implementation of standard cells were depicted in the development of an open source cell library, which contains several versions of different routing tracks [32] and an effective through silicon via (TSV) planning and repair framework [33]. These provide additional insight into the posing challenges of performing routing-aware placement in 3D ICs by modifying existing 2D algorithms. Effective variations of the established routing algorithms, such as the maze router, can be found in the literature [34]. Finally, the authors of [35] and [36] describe design techniques that can be extended to post-placement designs, and ensure their robustness to PVT (process voltage temperature) variations.

Overall, although much work exists with problem statements that do not contain obstacles, few of the aforementioned works deal with obstacles. Furthermore, time performance is typically not in the cornerstone of the proposed schemes. Its importance, however, can not be diminished, as practical problem instances can easily scale to the order of hundrends of millions of cells. Such complexity can only be tackled through efficient parallelism. As Abacus was shown in the relevant literature to achieve a very good performance at the expense of high computational time for the case where no obstacles exist, we based the contributions of this paper on proposing a parallelization approach together with the extensions necessary to handle obstacles. We term the resulting algorithm poAbacus (parallel obstacle-aware Abacus). For comparison reasons, we also present poTetris, which follows a similar design logic, but is based on the Tetris algorithm.

3. Obstacle-Aware Parallel Legalization Algorithms

In this section, we illustrate poAbacus and poTetris. Pseudocode 1 describes the basic Abacus algorithm that operates without considering obstacles. The cells are first ordered according to the *x*-coordinate (increasing fashion) (line 1). The algorithm then places the cells in an iterative manner, starting with the one of the minimum *x*-coordinate value (lines 2–13). In doing so, the best position at each candidate row is calculated (lines 3–11), and the best overall position (displacement wise, Manhattan distance) is selected among all of the possible candidates (line 9). The previous steps described for Abacus also hold true for Tetris. However, the algorithms differ in the manner they treat the case where an overlap might occur with a previously placed cell. Tetris simply places the overlapping cell at the first leftmost eligible position. On the other hand, Abacus calls for a function presented in Pseudocode 2. The function merges the overlapping cells into a cell cluster (line 10). It then computes the best cluster position (line 11). Obviously, the process might involve moving the already placed cells, thus, increasing their relevant (previously optimal) displacement. Thus, in order to identify the best possible total displacement change (in the whole cell cluster), the algorithm formulates the problem in a quadratic optimization fashion, and solves it using a dynamic programming method. This is done by the collapseClusters function (line 11), the details of which are described in the literature [2].

Pseudocode 1: Abacus algorithm

input: circuit cells C, circuit rows R
output: C cells aligned in R rows without overlaps

```
1  sort $C$ based on x-coordinate
2  foreach cell $c_i \in C$ do
3          bestCost := INF
4          bestRow := $-1$
5          foreach row $r_j \in R$ do
6              cost := $insertCell\left(c_i, r_j, TRIAL\right)$
7              if cost < bestCost then
8                  bestCost := cost
9                  bestRow := $r_j$
10             end if
11         end for
12         $insertCell(c_i, bestRow, NOTRIAL)$
13 end for
```

Pseudocode 2: Function *insertCell()*

input: cell c_i, row r_j, mode
output: Manhattan distance of c_i's displacement

```
1  oldPlacement: = existing placement before $C_i$ is inserted
2  if $area(c_i) + occupiedArea\left(r_j\right) > area(r_j)$ then
3        cost: = INF
4  end if
5  vertically align $c_i$ into $r_j$ //x-coordinate does not change
6  if $c_i$ does not overlap with any cluster $cl_u$ then
7        create new cluster $cl_{new}$ containing $c_i$
8        cost: = displacement of $c_i$
9  else
10       add $c_i$ into $cl_u$ with which it overlaps
12       cost: = $collapseClusters(cl_u)$
12 end if
13 if mode = TRIAL then
14       restore oldPlacement
15 end if
16 return cost
```

Both Tetris and Abacus are extended to account for obstacles in the following manner. We consider that each obstacle effectively splits all of the intersecting rows into sub-rows (before and after obstacle boundaries). Consider, for instance, the example of Figure 1, which shows an example placement scenario whereby the chip area is split into six standard rows; nine obstacles exist shown as grey areas; and the final position of nine cells A, B, . . . , and I must be defined. poTetris and poAbacus will operate by considering the induced sub-rows, whereby the initial rows are split because of the presence of the obstacles. In the example, three sub-rows exist in rows 1 and 5, while each of the rows 2, 3, 4, and 6 are split into two sub-rows. The total sub-rows induced by the obstaces in the example is 14, and they consist candidates for cell placement.

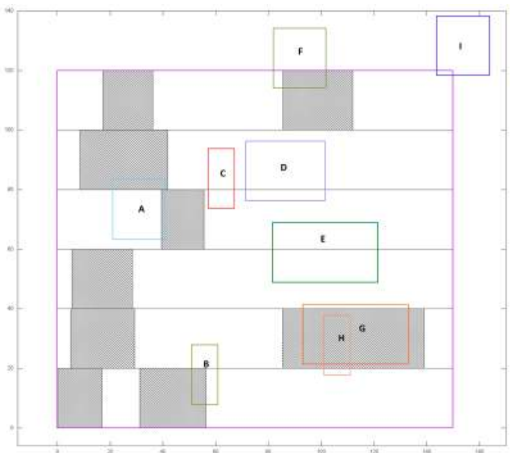
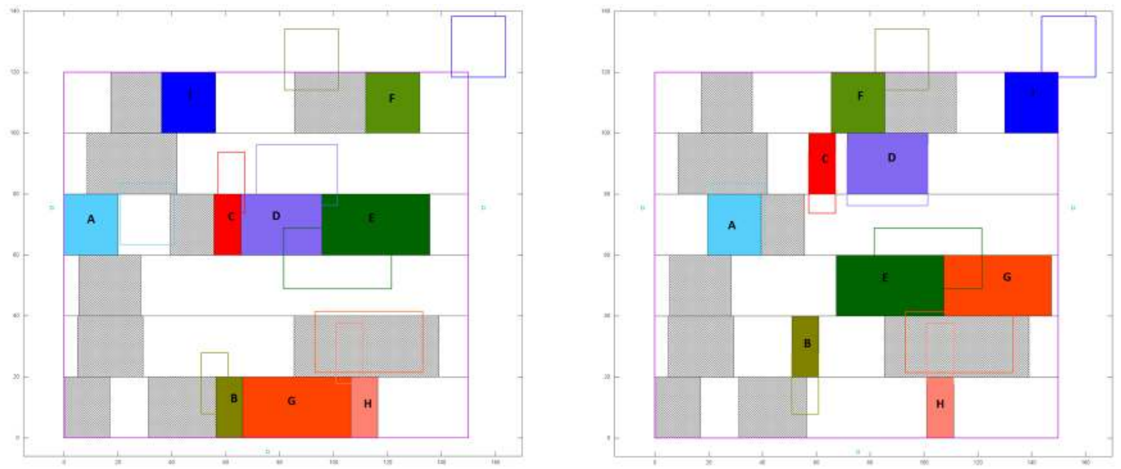


Figure 1. Example global placement

Continuing the example of Figure 1, in Figure 2a, the final placement achieved by poTetris is shown, and in Figure 2b, the one by poAbacus is shown. As it can be observed, the two algorithms lead to almost completely different results, with a closer look revealing that poAbacus results in a much smaller displacement compared with poTetris. To better understand why, consider, for instance, the case of E and G's placement. In poTetris, these cells will be placed on different rows to the one they overlap most (row four). In poAbacus, on the other hand, after E is first placed at row four, G's placement will result in forming a cluster with E and G. The algorithm then identifies the best position of the cluster as a whole.



(**a**) poTetris placement (**b**) poAbacus placement

Figure 2. Obstacle-aware placement examples

Depending on the number of obstacles and their height, the number of candidate positions both algorithms should check might drastically increase as multiple sub-rows are introduced. This might further hinder the algorithmic performance time wise, with the effects being more prominent in the case of poAbacus. For this reason, we tackle algorithmic parallelization not from the rather straightforward standpoint of spawning multiple threads to calculate each sub-row cell candidate position, but from the perspective of reducing the effective search space that is used for each cell placement decision. In Pseudocode 3, the proposed poAbacus is described. The algorithm is based on splitting the chip area into independent tile partitions (lines 2–14), and restricting the search space for each cell to the sub-rows contained in the tile it belongs to (line 18). On top, the cells of each tile are assigned to a separate thread in order to further the time gains with parallelization (line 15).

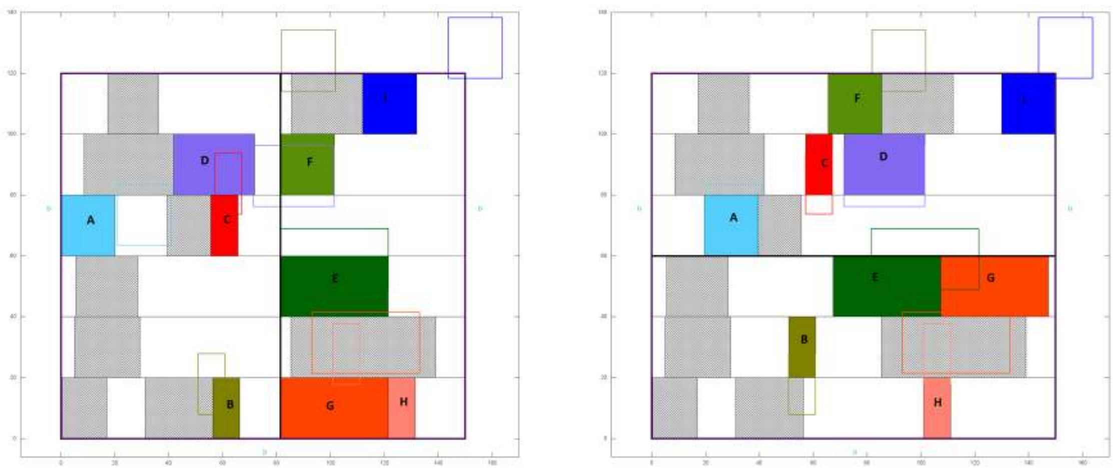
Pseudocode 3: poAbacus algorithm

input: circuit cells *C*, circuit rows *R*, number of horizontal partitions *N*, number of vertical partitions *M*
output: *C* cells aligned in *R* rows without overlaps

```
1  $C_{rem} := \varnothing$ //stores the leftover cells
2  $b = |R|/N$
3  split chip area into N-1 horizontal partitions of b rows each //the $N^{th}$ partition will have the
                                                  //remaining rows: $|R| - (N-1)b$
4  foreach horizontal partition h do
5     avgFreeSpace: = $(area(h) - obstacleArea(h))/M$
6     xoffset: = chipAreaWidth/s                  //s is a tunable parameter
7     for v: = 1 to M-1 do
8        setTileBoundary($T_{hv}$, getTileBoundary($T_{h(v-1)}$)+xoffset) // getTileBoundary($T_{h0}$) = 0
9          while freeArea($T_{hv}$) < avgFreeSpace do
10            setTileBoundary($T_{hv}$, getTileBoundary($T_{hv}$)+xoffset)
11         end while
12    end for
13    setTileBoundary($T_{hM}$, chipAreaWidth)
14 end for
15 par-foreach $T_{hv}$ do //each tile is assigned to a different thread
16    $C_{hv}$: = {all cells contained in $T_{hv}$}
17    $R_{hv}$: = {all sub-rows contained in $T_{hv}$}
18    Abacus ($C_{hv}$, $R_{hv}$)
19    foreach $(c_i, r_j) : c_i \in C_{hv} \wedge r_j \in R_{hv} \wedge c_i$ is placed at $r_j$ do
20         if $c_i$ exceeds $r_j$ then
21            down(mutex)
22             $C_{rem} \cup = \{c_i\}$
23            up(mutex)
24         end if
25    end for
26 end par-for
27 barrier
28 if $C_{rem} \neq \varnothing$ then
29    Abacus($C_{rem}$, $R_{sub}$) //$R_{sub}$ is the set of subrows existing in the whole chip area
30 end if
```

In order to achieve tile independence (necessary for efficient parallelization), it is imperative that the cells of a tile must be placed within the tile, and not be allowed to overlap with neighboring tiles. Thus, it is possible that some cells can be left unplaced, as no suitable position might exist within their assigned tile (lines 20–24). Both in poTetris and poAbacus, after all of the tile threads terminate (line 27), any remaining cells are placed during a second phase, without considering tile boundary restrictions (line 29). This necessary second step might be a source of performance degradation, as most likely, the remaining cells will be placed in distant positions. In order to minimize the negative effects, the tile splitting process must aim at distributing the obstacle area judiciously among the tiles. This is achieved by first creating an $N \times M$ partition into roughly equally sized horizontal zones (line 2–3), and subsequently defining the vertical tile boundaries (lines 4–14). For this reason, the average free space per tile is calculated (line 5), and the x-axis is split into s candidate cutting points (line 6). In the experiments, s = 1,000. The candidate cutting points are scanned each time, defining tile vertical boundaries so that the free space per tile is close to the expected average (lines 7–14). The last tile within a horizontal zone is defined by the right chip area boundary (line 13).

Figure 3 continues the example of Figure 1, with two different partitioning scenarios for poTetris and poAbacus, respectively. It also points out the different impact that horizontal and vertical cuts have on algorithms' performance. In Figure 3a, cells A, B, C, and D belong to the left tile, while the rest belong to the right. The introduction of a vertical cut forces poTetris to place E at its optimal position, whereas without it, the candidate position at the 4th row would have been just after the obstacle of the 4th row (for a larger displacement). In Figure 3b, the cells are also divided into two disjoint sets based on where their left-down *x*-coordinate belongs. Notice, that the final placement achieved by poAbacus is identical to the one of Figure 2b. However, the complexity of the individual cell placement decisions is almost halved, as seven sub-rows must be evaluated (the ones belonging to the relevant tile) instead of the 14 that exist in the total chip area.



(**a**) poTetris placement (1 x 2 partition) (**b**) poAbacus placement (2 x 1 partition)

Figure 3. Obstacle-aware placement with tiles

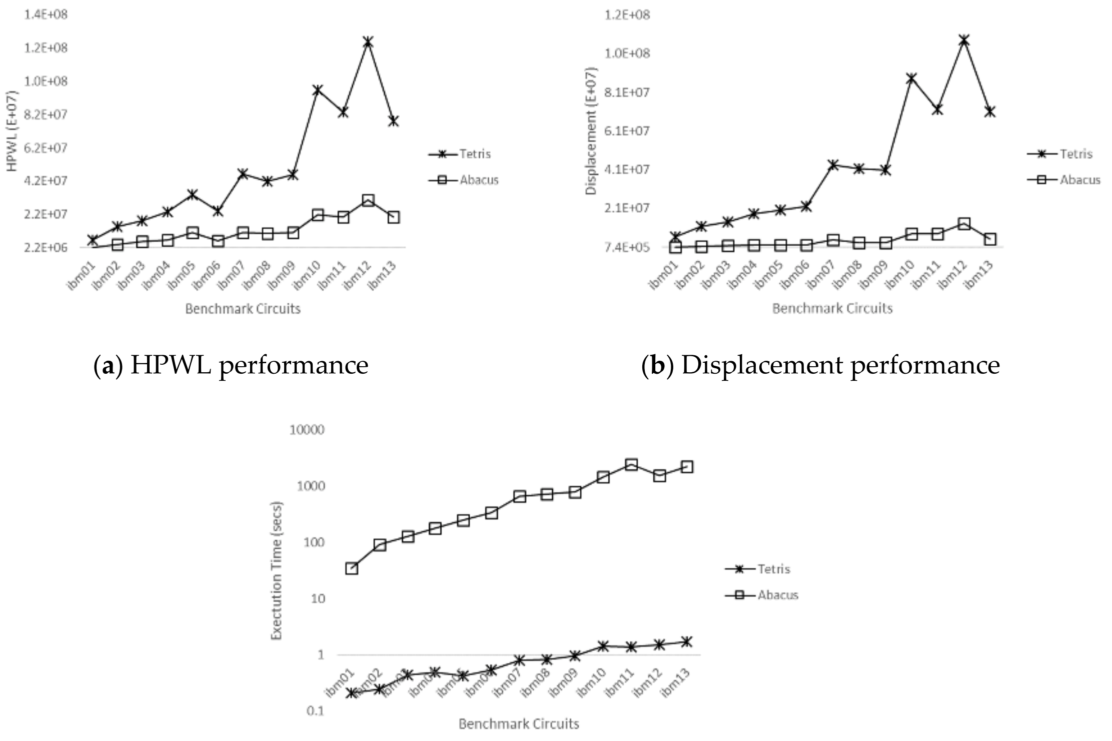
4. Experiments

4.1. Experimental Setup

Experiments were carried out using the ibm01-13 benchmark circuits provided by the authors of [37]. As these circuits have no obstacles, random obstacles were introduced so that they cumulatively cover a specific percentage of the free space (calculated by subtracting the total area of cells from the chip area). It should be noted that the introduction of the random obstacles rather accounts for the worst case algorithmic wise, notably for poAbacus. This is due to the fact that in real life designs, obstacles are independent rectangular areas with some spacing in between them. On the other hand, random obstacles may lead to non-rectangularly shaped continuous obstacle areas, which make tile partitioning harder. Different scenarios concerning free space were evaluated per circuit. For each scenario, 10 runs were conducted and the results were averaged. Performance evaluation was done across the following three metrics: net wirelength, displacement, and running time. Displacement was measured as the Manhattan distance between the starting and end cell position. Net wirelength was measured using the half perimeter wirelength (HPWL) of the minimum bounding rectangle, which contains all of the cells of a net. NTUplace3 [10] was used as a global placer to obtain the starting cell positions. These positions formed the input upon which the proposed legalization methods were evaluated. Multithreaded parallelism was implemented using OpenMP. Experiments were conducted on a Linux server with two Intel Xeon E5-2630 processors (2.3 GHz) using hyper threading (12 physical cores total).

4.2. Standalone Tetris and Abacus Evaluation

In a first experiment, we evaluated the performance of the standalone Tetris and Abacus. Figure 4 shows the resulting performance on the three metrics, for the case where 10% of the free space exists. It can be clearly seen that Abacus outperforms Tetris by even an order of magnitude (in certain cases) in both HPWL and displacement terms. On the other hand, as shown in Figure 4c, Abacus' performance comes at the cost of a particularly higher running time, by three orders of magnitude in most cases. These first results undoubtedly illustrate the necessity of introducing faster approaches compared to the baseline Abacus algorithm.



(**a**) HPWL performance (**b**) Displacement performance

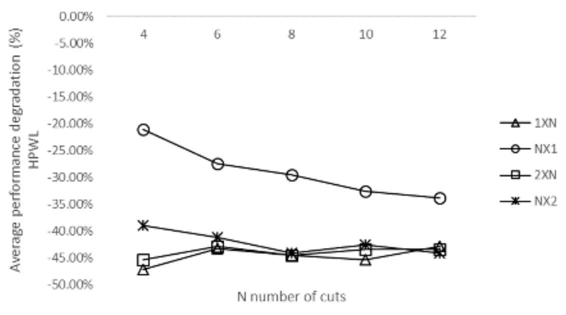
Figure 4. Performance comparison standalone Tetris and Abacus. HPWL—half perimeter wirelength.

4.3. Evaluation of poTetris and poAbacus

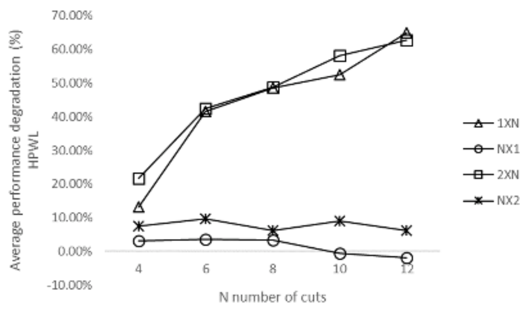
Next, we proceeded with evaluating the performance of poTetris and poAbacus. Figure 5 compiles the relevant performance degradation in HPWL and the displacement terms of poTetris and poAbacus as a percentage of the related Tetris and Abacus performance. Specifically, for poTetris, the degradation percentage is given by 100((*perf*(poTetris)-*perf*(Tetris)/*perf*(Tetris)), and similarly for poAbacus. Each point in the plots depict the average percentage results for the 13 benchmark circuits, assuming 20% free space. Figure 5 plots the performance of the algorithms for four different tile partitioning cases and five different number of cuts.

As it can be inferred by Figure 5a,c, poTetris outperforms Tetris in both HPWL and displacement terms (negative degradation means an improvement). In particular, the gains in displacement terms reach 80% when *N* vertical cuts are introduced. By comparison, the effects of the vertical partitioning appear to be the opposite in poAbacus, whereby introducing more horizontal cuts ($N \times 1$ and $N \times 2$ plots) apparently leads to z better solution quality compared with the other options. In fact, Figure 5d depicts a substantial improvement in the displacement terms, which can reach more than 20%. These results can be explained for the case of poTetris, as vertical cuts reduce the allowable displacement in the *x*-axis. On the other hand, poAbacus defines the optimal cluster position within each sub-row. Therefore, restricting the allowable cluster movements along the *x*-axis (as vertical cuts do) will likely hinder performance, whereas restricting the *y*-axis will not do so and might in fact prove beneficial in certain scenarios.

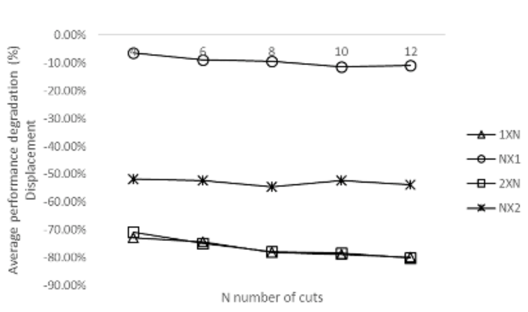
Figure 6 illustrates the time improvement of poTetris and poAbacus over their simple counterparts. Both algorithms achieve a reduced running time, with results for poAbacus being particularly impressive, demonstrating a reduction in running time that reaches 95%.



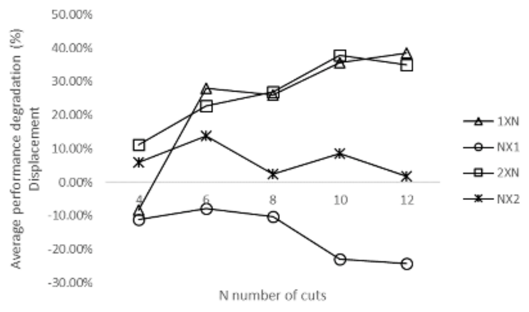
(**a**) poTetris, HPWL performance



(**b**) poAbacus, HPWL performance

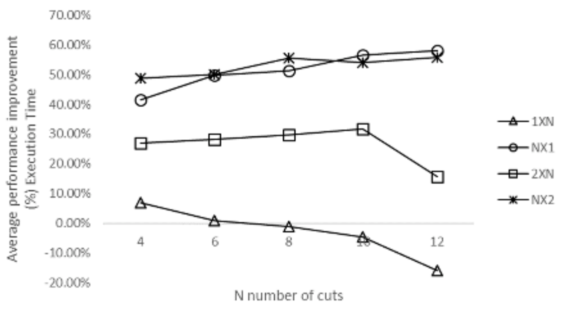


(**c**) poTetris, displacement performance

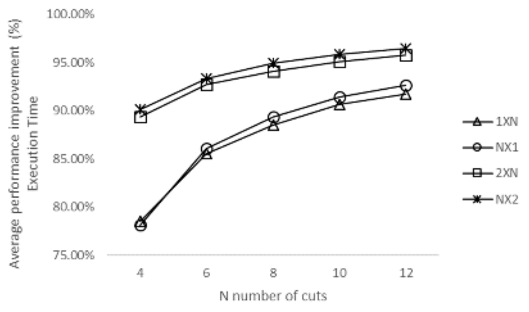


(**d**) poAbacus, displacement performance

Figure 5. Solution quality performance comparison for different partitioning alternatives.



(a) poTetris, time performance

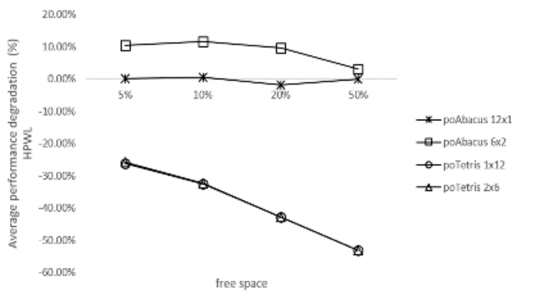


(b) poAbacus, time performance

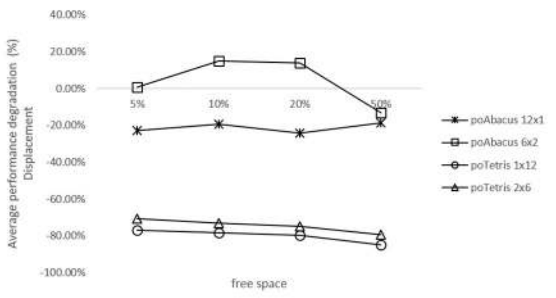
Figure 6. Time comparison for different partition alternatives.

Next, we evaluate the solution quality of the algorithms for different free space percentages. Figure 7 shows the results. poTetris achieves an improved performance over Tetris, with the gains in the displacement being constantly around 80%. In poAbacus, the displacement gains vary between ±20%, depending on the particular case considered, while HPWL remains unaffected in the 12 × 1 case.

Having established that in terms of solution quality, poAbacus has a comparable performance to Abacus and might even exhibit improvement, depending on the evaluation scenario, we proceed by plotting the speedup trends (*time*(Abacus)/*time*(poAbacus)) for the N × 1 partitioning as the number of threads increases. Results presented in Figure 8 demonstrate an impressive superlinear behavior, whereby the achievable speedup with 12 threads reaches 66. This is a strong testament on the merits of our approach, which combines multithreaded parallelism with search space pruning.



(a) HPWL performance



(b) Displacement performance

Figure 7. Solution quality performance comparison for different free space percentages.

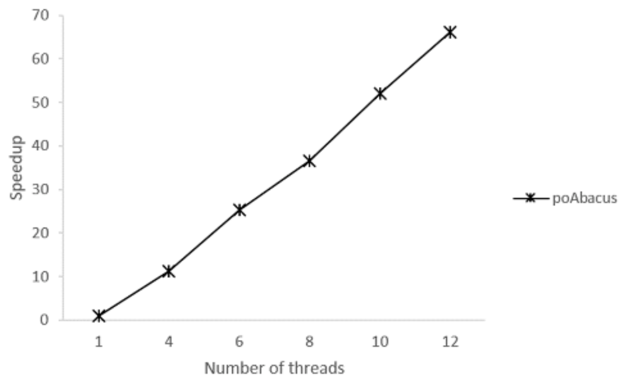


Figure 8. Speedup of poAbacus over Abacus.

5. Conclusions

In this paper, we tackled the problem of improving the performance of a state-of-the-art legalizer, and of a fast and greedy one in the presence of obstacles. This improvement was achieved by inducing a judicious tile partitioning of the chip area, which takes into consideration obstacles and allows for both reducing the search space and for the split of computations into independent tasks that can be trivially parallelized. The results are particularly encouraging, demonstrating an improvement in all aspects of HPWL, displacement, and time for Tetris. They also demonstrate that the proposed poAbacus scheme can achieve an impressive superlinear speedup over its simple counterpart without negatively affecting HPWL and displacement, if horizontally defined tiles are used.

Author Contributions: Conceptualization, P.O. and T.L.; methodology, A.N.D., K.K., and A.K.; software, P.O. and K.K.; validation, P.O., A.N.D., and A.K.; formal analysis, P.O. and K.K.; investigation, A.N.D. and T.L.; resources, S.G.I.; data curation, A.N.D.; writing (original draft preparation), P.O., A.N.D., K.K., T.L., and A.K.; writing (review and editing), G.I.S.; visualization, P.O. and T.L.; supervision, G.I.S.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hill, D. Method and System for High Speed Detailed Placement of Cells within an Integrated Circuit Design. U.S. Patent 6370673 B1, 9 April 2002.
2. Spindler, P.; Schlichtmann, U.; Johannes, F.M. Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement. In Proceedings of the 2008 International Symposium on Physical Design (ISPD '08), Portland, OR, USA, 13–16 April 2008; ACM Press: New York, NY, USA, 2008.
3. Oikonomou, P.; Dadaliaris, A.N.; Loukopoulos, T.; Kakarountas, A.; Stamoulis, G.I. A Tetris-based Legalization Heuristic for Standard Cell Placement with Obstacles. In Proceedings of the 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 7–9 May 2018; IEEE: Piscataway, NJ, USA, 2018.

4. Cardoso, M.S.; Smaniotto, G.H.; Bubolz, A.A.O.; Moreira, M.T.; da Rosa, L.S.; de Souza Marques, F. Libra: An automatic design methodology for CMOS complex gates. *IEEE Trans. Circuits Syst. II Express Br.* **2018**, *65*, 1345–1349. [CrossRef]
5. Guo, J.; Zhu, L.; Sun, Y.; Cao, H.; Huang, H.; Wang, T.; Qi, C.; Zhang, R.; Cao, X.; Xiao, L.; et al. Design of area-efficient and highly reliable RHBD 10T memory cell for aerospace applications. *IEEE Trans. Very Larg. Scale Integr. Syst.* **2018**, *26*, 991–994. [CrossRef]
6. Mishra, V.K.; Chauhan, R.K. Area efficient layout design of CMOS circuit for high-density ICs. *Int. J. Electron.* **2018**, *105*, 73–87. [CrossRef]
7. Dadaliaris, A.; Oikonomou, P.; Koziri, M.; Nerantzaki, E.; Hatzaras, Y.; Garyfallou, D.; Loukopoulos, T.; Stamoulis, G. Heuristics to augment the performance of tetris legalization: Making a fast but inferior method competitive. *J. Low Power Electron.* **2017**, *13*, 220–230. [CrossRef]
8. Chou, S.; Ho, T.-Y. OAL: An obstacle-aware legalization in standard cell placement with displacement minimization. In Proceedings of the 2009 IEEE International SOC Conference (SOCC), Belfast, UK, 9–11 September 2009.
9. Spindler, P.; Schlichtmann, U.; Johannes, F.M. Kraftwerk2—A fast force-directed quadratic placement approach using an accurate net model. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2008**, *27*, 1398–1411. [CrossRef]
10. Chen, T.-C.; Jiang, Z.-W.; Hsu, T.-C.; Chen, H.-C.; Chang, Y.-W. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2008**, *27*, 1228–1240. [CrossRef]
11. Popovych, S.; Lai, H.-H.; Wang, C.-M.; Li, Y.-L.; Liu, W.-H.; Wang, T.-C. Density-Aware Detailed Placement with Instant Legalization. In Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference (DAC '14), San Francisco, CA, USA, 1–5 June 2014; ACM Press: New York, NY, USA, 2014.
12. Netto, R.; Guth, C.; Livramento, V.; Castro, M.; Pilla, L.L.; Guntzel, J.L. Exploiting parallelism to speed up circuit legalization. In Proceedings of the 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, Monaco, 11–14 December 2016.
13. Oikonomou, P.; Koziri, M.G.; Dadaliaris, A.N.; Loukopoulos, T.; Stamoulis, G.I. Domocus: Lock free parallel legalization in standard cell placement. In Proceedings of the 6th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 4–6 May 2017.
14. Wang, C.-H.; Wu, Y.-Y.; Chen, J.; Chang, Y.-W.; Kuo, S.-Y.; Zhu, W.; Fan, G. An effective legalization algorithm for mixed-cell-height standard cells. In Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Chiba, Japan, 16–19 January 2017.
15. Kennings, A.; Darav, N.K.; Behjat, L. Detailed placement accounting for technology constraints. In Proceedings of the 2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC), Playa del Carmen, Mexico, 6–8 October 2014.
16. Sketopoulos, N.; Sotiriou, C.; Simoglou, S. Abax: 2D/3D legaliser supporting look-ahead legalisation and blockage strategies. In Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), Dresden, Germany, 19–23 March 2018.
17. Viswanathan, N.; Pan, M.; Chu, C. FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In Proceedings of the 2007 Asia and South Pacific Design Automation Conference, Yokohama, Japan, 23–26 January 2007.
18. Lu, J.; Chen, P.; Chang, C.-C.; Sha, L.; Huang, D.J.-H.; Teng, C.-C.; Cheng, C.-K. ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method. *ACM Trans. Des. Autom. Electron. Syst.* **2015**, *20*, 1–34. [CrossRef]
19. Lu, J.; Zhuang, H.; Chen, P.; Chang, H.; Chang, C.-C.; Wong, Y.-C.; Sha, L.; Huang, D.; Luo, Y.; Teng, C.-C.; et al. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 685–698. [CrossRef]
20. Wu, G.; Chu, C. Detailed Placement Algorithm for VLSI Design with Double-Row Height Standard Cells. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1569–1573. [CrossRef]
21. Taghavi, T.; Yang, X.; Choi, B.-K. Dragon2005: Large-Scale Mixed-Size Placement Tool. In Proceedings of the 2005 International Symposium on Physical Design (ISPD '05), San Francisco, CA, USA, 3–6 April 2005; ACM Press: New York, NY, USA, 2005.

22. Ababei, C.; Navaratnasothie, S.; Bazargan, K.; Karypis, G. Multi-objective circuit partitioning for cutsize and path-based delay minimization. In Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD 2002), San Jose, CA, USA, 10–14 November 2002.
23. Kim, M.-C.; Lee, D.-J.; Markov, I.L. SimPL: An effective placement algorithm. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2012**, *31*, 50–60. [CrossRef]
24. Brenner, U. VLSI legalization with minimum perturbation by iterative augmentation. In Proceedings of the 2012 Design, Automation and Test in Europe Conference and Exhibition (DATE), Dresden, Germany, 12–16 March 2012.
25. Brenner, U. BonnPlace legalization: Minimizing movement by iterative augmentation. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2013**, *32*, 1215–1227. [CrossRef]
26. Hu, J.; Zhou, Q.; Gao, W.; Qian, X.; Zhou, Q. An effective legalization approach based on multiple ordering. In Proceedings of the 2013 International Conference on Communications, Circuits and Systems (ICCCAS), Chengdu, China, 15–17 November 2013.
27. Zhou, Q.; Hu, J.; Zhou, Q. An effective iterative density aware detailed placement algorithm. In Proceedings of the 2014 IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne, VIC, Australia, 1–5 June 2014.
28. Li, H.; Chow, W.-K.; Chen, G.; Young, E.F.Y.; Yu, B. Routability-driven and fence-aware legalization for mixed-cell-height circuits. In Proceedings of the 55th Annual Design Automation Conference on (DAC '18), San Francisco, CA, USA, 24–29 June 2018; ACM Press: New York, NY, USA, 2018.
29. Darav, N.K.; Bustany, I.S.; Kennings, A.; Westwick, D.; Behjat, L. Eh?Legalizer: A high performance standard-cell legalizer observing technology constraints. *ACM Trans. Des. Autom. Electron. Syst.* **2018**, *23*, 1–25. [CrossRef]
30. Cho, M.; Ren, H.; Xiang, H.; Puri, R. History-Based VLSI Legalization Using Network Flow. In Proceedings of the 47th Design Automation Conference (DAC '10), Anaheim, CA, USA, 13–18 June 2010; ACM Press: New York, NY, USA, 2010.
31. Fabre, S.; Guntzel, J.L.; Pilla, L.; Netto, R.; Fontana, T.; Livramento, V. Enhancing Multi-threaded legalization through k-d tree circuit partitioning. In Proceedings of the 2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI), Bento Goncalves, Brazil, 27–31 August 2018; pp. 1–6.
32. Yan, C.; Salman, E. Mono3D: Open source cell library for monolithic 3-D integrated circuits. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 1075–1085. [CrossRef]
33. Xu, Q.; Chen, S.; Xu, X.; Yu, B. Clustered fault tolerance TSV planning for 3-D integrated circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2017**, *36*, 1287–1300. [CrossRef]
34. Lepercq, É.; Blaquière, Y.; Savaria, Y. A pattern-based routing algorithm for a novel electronic system prototyping platform. *Integration* **2018**, *62*, 224–237. [CrossRef]
35. Carbajal-Gomez, V.; Tlelo-Cuautle, E.; Sanchez-Lopez, C.; Fernandez-Fernandez, F. PVT-robust CMOS programmable chaotic oscillator: Synchronization of two 7-scroll attractors. *Electronics* **2018**, *7*, 252. [CrossRef]
36. Abbas, Z.; Olivieri, M.; Ripp, A. Yield-driven power-delay-optimal CMOS full-adder design complying with automotive product specifications of PVT variations and NBTI degradations. *J. Comput. Electron.* **2016**, *15*, 1424–1439. [CrossRef]
37. ISPD04 Benchmark Circuits. Available online: http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.public.iastate.edu/~{}nataraj/ISPD04_Bench.html (accessed on 21 December 2018).