Deterministic Timing-Driven Parallel Placement by Simulated Annealing using Half-Box Window Decomposition

Jeffrey B. Goeders, Guy G. F. Lemieux, and Steven J. E. Wilton,
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, V6T 1Z4
jgoeders, lemieux, stevew @ece.ubc.ca

***Abstract*—As each generation of FPGAs grow in size, the run time of the associated CAD tools is rapidly increasing. Many past efforts have aimed at improving the CAD run time through parallelization of the placement algorithm. Wang and Lemieux presented an algorithm that is scalable, deterministic, timing-driven and achieves speedup over VPR [Wang and Lemieux FPGA'11]. This paper provides two significant alterations to Wang and Lemieux's algorithm, resulting in additional speedup and quality improvement.**

The first contribution is a new data decomposition scheme, called the half-box window technique, which achieves speedup by reducing the frequency of thread synchronization. The second contribution is the development of an improved annealing schedule, which further improves run time and slightly improves the quality of results.

Together, these modifications achieve run time speedups of up to 70%. To put this in perspective, Wang and Lemieux required 25 threads to achieve best speedup, while this work requires only 16 threads. For a 10% degradation in quality, the new 16-thread algorithm achieves a 51x speedup over VPR, compared to a 35x speedup by the 25-thread original algorithm. Regarding quality, the best quality of results achieved by the new algorithm is a 5% degradation versus VPR, compared to a 8% degradation of the original Wang and Lemieux algorithm.

***Index Terms*—FPGA; CAD; parallel placement**

I. Introduction

As CMOS technology continues to scale, the number of elements within an FPGA is continuing to increase. This doubling in size every generation means that the associated FPGA CAD tools must perform more computation with each generation as well — even if the computation is linear, it must do twice the work. Thus, the amount of computation is growing much faster than the speed of the processors upon which these CAD tools are to be run [1]. As a result, the run-times of these tools are increasing dramatically, which in turn slows product development, increases the design cost and lengthens the time to market. The rate at which CAD run time is increasing is not sustainable. This has driven significant research into creating faster CAD tools, e.g., [1]–[4].

One way to accelerate FPGA CAD tools is to parallelize the CAD flow. Modern processors contain multiple cores, and if these cores can operate together, the turn-around time of FPGA CAD iterations can be significantly shortened. There have been a number of papers from both industry and academia describing parallel CAD algorithms [5]–[13]. Each of these previous works describes how the problem space can be divided or *decomposed* into tasks for individual processors, and how strategic communication and synchronization can ensure acceptable quality of results. In this paper, we enhance the timing-driven parallel placement algorithm described by Wang and Lemieux in [5]. Their solution divides the entire FPGA device into regions, each of which can be processed in parallel, and uses barriers to ensure that the overall result is deterministic. Compared to the original sequential placement algorithm in VPR 5.0 [14], they achieve a speedup of 123x

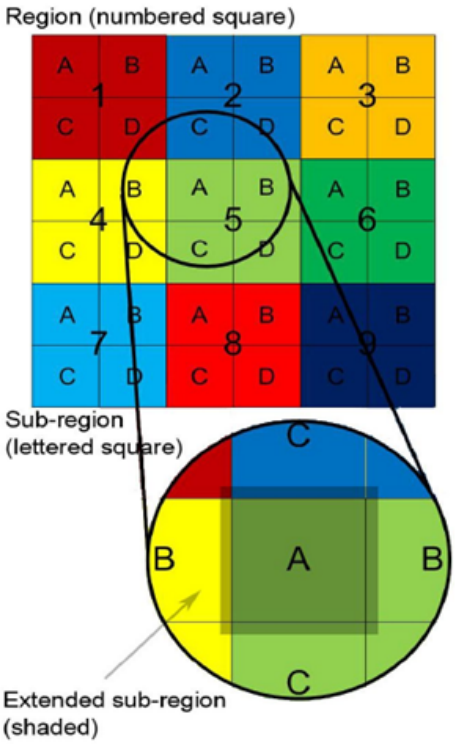

Fig. 1. Data decomposition of Wang and Lemieux's algorithm, from [5].

when using 25 threads, but with an 8% increase in the critical path delay and a 11% increase in the bounding box metrics.

In this paper, we consider two enhancements to the algorithm in [5]. First, *we show that an improved decomposition strategy reduces the number of barriers required for synchronization, improving the overall run-time* with the same quality of result (QoR). In [5], barriers are used to ensure that each processor sees a consistent view of the current solution; by being more aggressive in the manner in which processors share the optimization space, we can reduce the number of barriers while maintaining the deterministic nature of the original algorithm. We call the new approach the half-box window decomposition method, or simply the half-box method for short.

The second contribution of this paper is an *improved parallel annealing schedule that gets closer to the full quality of the original VPR 5.0 algorithm*. Our work demonstrates that the Wang and Lemieux annealing schedule, which emphasizes additional moves early to allow blocks to migrate across the chip, is not necessary. We find that given a sufficiently long run-time, the original Wang and Lemieux algorithm can achieve, at best, a 7.5% bounding box cost increase compared to VPR 5.0. The new annealing schedule, combined with the half-box method, reduces the bounding box cost increase to just 5%.

By combining both the half-box method and the newly tuned annealing schedule, the run time is improved up to 70%. The 70% reduction in run-time is significant, since it allows for faster turn-around times and shorter design cycles.

The paper is organized as follows. Section II describes previous work, with an emphasis on the algorithm described in [5]. Section III shows how the number of barriers in the baseline algorithm can be reduced, and quantifies the impact on overall run-time. Section IV then describes our experiments aimed at improving the QoR by tuning the annealing parameters. Section V provides final results and section VI concludes.

II. Previous Works

Parallel placement research has been ongoing for many years; however, most past solutions do not meet the requirements imposed by modern FPGA CAD tools. Almost all of the previous work has targeted wirelength only and does not support the timing-driven requirement of current CAD tools [9]–[13]. In addition, these algorithms, with the exception of [9], are nondeterministic, meaning they do not produce the same result with each execution. Commercial CAD developers will opt for deterministic algorithms as they ensure reproducibility, which is essential to effective regression testing and debugging [8].

Ludwin [8] presents a solution that is both timing-driven and deterministic. CLB swaps are speculatively executed in parallel, but committed sequentially to prevent conflicts that would arise from two threads attempting to swap the same CLB. The algorithm achieves a 2.2x speedup on four cores. However, the fact that swaps are committed sequentially means that the algorithm will likely not scale well to a large number of processors, where conflicts will be more likely.

Wang and Lemieux [5] provide a solution that is timing-driven, deterministic and scalable. To the best of our knowledge, this is the only published work that meets all of these requirements.

A. Wang and Lemieux's Algorithm

Wang and Lemieux's algorithm is a simulated annealing approach based on a parallelization of VPR 5.0. The majority of VPR processing takes place in a set of nested loops. The outer loop controls the annealing temperature, and the inner loop considers many CLB swaps at each temperature. Wang and Lemieux's algorithm parallelizes the inner loop such that all threads are considering CLB swaps simultaneously.

The grid of CLBs is divided into equal-sized partitions, called *regions*, each assigned to one thread. Regions are further subdivided into four smaller *subregions*. Subregions are extended by 2 CLBs in each direction, forming *extended subregions*. The regions and subregions are respectively numbered and lettered in Figure 1, with an extended subregion shown in the inset.

Each thread loops through their four subregions, considering many swaps at each phase. CLBs can be swapped from the subregion to anywhere in the extended subregion, which allows for migration of blocks between regions. Synchronization keeps all threads working in the same respective subregion, which ensures that no two threads are simultaneously operating on the same CLBs. For example, all threads will consider swaps within subregion A at the same time. Then, the threads will move on to subregion B, then C and finally D before repeating the process. Since no two extended subregions of the same letter overlap, collisions will be prevented.

Between each change of subregion a barrier allows for synchronization of the threads. This barrier is used to:

1) Keep all threads working in the same subregion.
2) Allow threads to synchronize their changes with a global copy of the grid.

While operating within a subregion, each thread is unaware of the changes being made by the other threads, and will not receive the changes until the next synchronization. Although this causes some swap choices to be made based on stale data, it has been shown that this does not affect the QoR [5] [15].

In the remainder of this paper, Wang and Lemieux's algorithm will be referred to as the *baseline algorithm*.

III. Improving Domain Decomposition

One potential way to improve Wang and Lemieux's algorithm is to reduce the number of synchronization barriers required. According to the data in [5], up to 17% of the execution time is spent waiting at barriers while switching between subregions. The algorithm must wait idle at the barrier, as using this time to perform additional swaps would cause nondeterministic behaviour. The algorithm requires barriers and thread synchronization between iterations of each subregion,

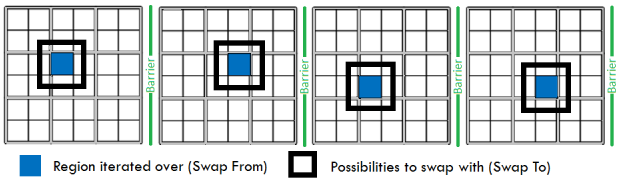

Fig. 2. Wang and Lemieux's algorithm. Swap-From and Swap-To regions of 1 thread. Execution cycles through stages from left to right, repeatedly.

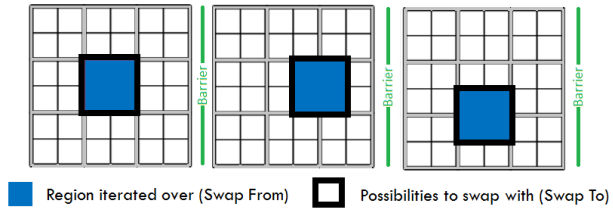

Fig. 3. Full-Box technique. Swap-From and Swap-To regions of 1 thread. Execution cycles through stages from left to right, repeatedly.

totalling four times while iterating over a thread region. By reducing the number of barriers, the run time can be improved.

The following section explores different techniques designed to reduce the number of synchronization barriers. Three different techniques are presented; however, only the last technique showed improvement in run time over the baseline algorithm.

A. Explanation of Representation

Figures are used to represent the decomposition methods of the different techniques. Figure 2 is provided as an example, and demonstrates the operation the baseline algorithm for one thread in a 9-thread configuration. In each figure the shaded area indicates the region that is iterated over (swap-from). Each CLB in this region is considered for swapping. The thick black boxed area indicates the region of blocks that can be swapped with (swap-to). Barriers are indicated where necessary. Although the figures show shaded regions for only one or two threads, the pattern is replicated across all thread regions.

B. Full-Box Technique

The Full-Box technique eliminates as many barriers as possible. The baseline algorithm requires a barrier between each subregion. By modifying the swap regions, four subregions can be iterated over between each barrier, essentially reducing the number of barriers by a factor of four. This can be accomplished by making the swap-from and swap-to regions coincident. Figure 3 illustrates the technique. The first step allows blocks to move around within the regions, while the second and third steps allow for horizontal and vertical migration.

This technique shows no significant improvement over the original algorithm. This is likely due to the fact that although the number of barriers is reduced, the ability for blocks to migrate is reduced. If a certain block would produce better

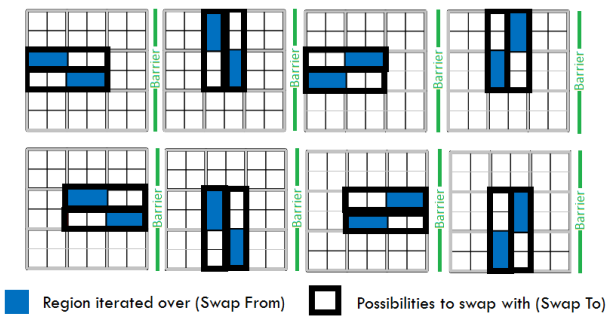

Fig. 4. Strip technique. Swap-From and Swap-To regions of 2 threads. Execution cycles through stages from left to right, top row then bottom row, repeatedly.

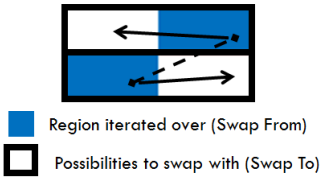

Fig. 5. Example of stale data problems in long strip-based regions. The two points represent CLBs connected by a net.

QoR on the other side of the FPGA, it will take much longer to migrate there compared to the original method. With the Full-Box technique, the swap-to and swap-from regions are identical. In the original algorithm, blocks which migrate outside of the swap-from subregion are never selected again as the first move candidate, so they are unlikely to move back into the swap-from subregion. In the Full-Box technique, blocks which move toward the correct half of the subregion on their first move are, on average, 50% likely to be visited again as a first move candidate. As a result, they can randomly be moved back to the incorrect half, and make no progress this round. In contrast, the baseline algorithm is more likely to make progress every round.

C. Strip Technique

Another approach is to allow for greater migration. Instead of square regions, this technique employs longer rectangular regions. Figure 4 illustrates the algorithm. This allows a block to migrate up to three positions for every two subregion iterations. This method is also able to perform two subregion iterations between each barrier, using half as many barriers as the baseline algorithm. The algorithm alternates between horizontal and vertical rectangular regions to allow for migration in each direction. The algorithm cycles through multiple combinations of swap-from/swap-to regions to ensure each part of the grid receives fair treatment.

This technique also showed no significant improvement over the baseline algorithm. We suspect that the primary reason is that long strip-sized regions can increase the effect of stale data. The original solution used small square subregions, so the effect of stale data was minimal. In contrast, if long narrow subregions are used, CLBs can move great distances

within a subregion, causing other threads to make incorrect decisions regarding swaps. Figure 5 illustrates an example of this problem. In this theoretical scenario, both threads will swap two connected CLBs in order to shorten the distance between them. However, since neither is aware of the other's choice, both make the swap and no benefit is gained.

D. Half-Box Technique

The final approach aims to increase block migration, but without making long narrow strips that potentially create stale data problems. This method requires half of the number of barriers as the baseline algorithm, while also allowing for migration in all directions. Figure 6 illustrates the technique. The swap-from region is set to include two of the four subregions assigned to the thread, which reduces the number of barriers by a factor of two. The swap-to region includes the entire swap-from region as well as the two subregions from a neighbouring thread that are not included in the neighbour's swap-from region. For threads operating on the periphery of the device, it may not have a neighbouring thread to use for part of its swap-to region. When this occurs, the swap-to region is not expanded and the swap-to region only includes the swap-from region. The algorithm alternates between vertical and horizontal neighbours, allowing blocks to migrate in all directions. To ensure fairness between the regions near the periphery of the device, the algorithm alternates the direction of the swap-to neighbour. Thus, the four stages of the algorithm as illustrated in Figure 6 are swapping with the thread neighbour above, then to the right, then below, and finally to the left. This four-stage approach performed much better than just using the first two stages.

E. Evaluation Methodology

The different decomposition techniques were evaluated by performing placement on 7 synthetic circuits from Un/DoPack [16]. Although these are synthetic circuits, they were built by combining 20 real sub-circuits, which allows them to have properties similar to a real circuit. Each circuit contains roughly 40,000 6-input LUTs. Large circuits are needed to test scaling this parallel placement algorithm to a large number of threads; small circuits cannot be decomposed into enough regions to support a large number of threads. Wall-clock time was used for timing measurements; these measurements include the execution time of the main simulated annealing placement loop only, not the initialization steps (reading the netlist from a file, building a model of the architecture, or the various sanity checks). For quality comparisons, all bounding box and critical path results are normalized to the results produced by VPR. The normalized values for all 7 circuits are averaged using a geometric mean to produce a single comparison value for both bounding box and critical path.

The machine used was a Dell R815, which is a 4-socket configuration, each with an 8-core AMD Opteron 6128 processor running at 2.0GHz, totalling 32 cores. The machine is running a standard installation of Ubuntu Server Linux 10.10. Unless otherwise noted, experiments were run with the parallel placer configured for 25 threads. This was chosen for two reasons:

1) Wang and Lemieux report the best results for these circuits were obtained when using 25 threads and provide results for this configuration.
2) It was the goal to use lots of threads, but not so much that the 32-core machine would be saturated, otherwise operating system threads could interfere with the results.

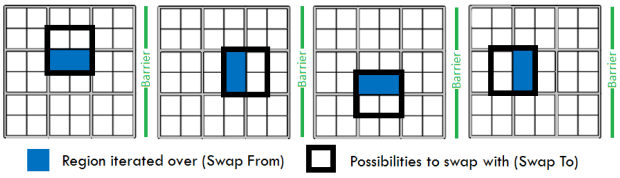

Fig. 6. Half-Box technique. Swap-From and Swap-To regions of 1 thread. Execution cycles through stages from left to right, repeatedly.

F. Evaluation Results

Of the different decomposition techniques, only the Half-Box approach shows improvement over the baseline algorithm. The quality versus run time is plotted for the bounding box cost and critical path delay in Figures 7 and 8. Since additional improvements are made in Section IV, the overall speedup over the baseline is provided later, in Section V.

IV. Improving Quality of Results

A. Motivation

Wang and Lemieux's parallel placer achieves excellent speedup results; however, quality suffers by approximately 10% compared to the serial VPR implementation. During early design optimization, when incremental design changes are frequent, a 10% quality loss could be acceptable as often designers are concerned more with functional correctness than performance. However, in later design phases performance can become a major factor, and a 10% drop in performance may be unacceptable.

One could argue that in cases where performance is key, the single threaded VPR could be run. However, if the size

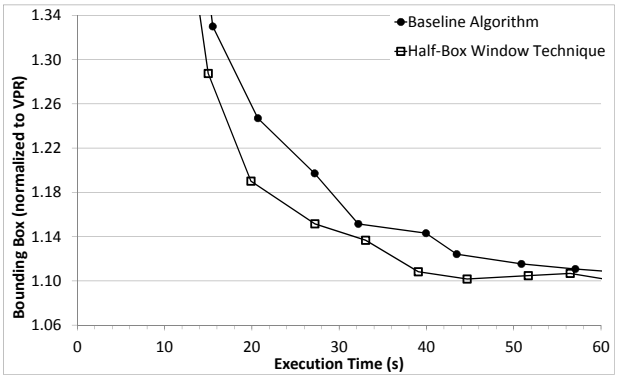

Fig. 7. QoR degradation of the bounding box cost versus run time for the baseline algorithm and the Half-Box technique.

of FPGAs continue to scale according to Moore's Law, the execution time could extend from tens of hours to days or even weeks. This long run time may be unacceptable, even if only run occasionally. Instead, a single, parallel, scalable solution needs to be developed. Ideally, this solution would offer the same quality results as single-threaded VPR.

Wang and Lemieux's algorithm contains a parameter, *region_place_count*, which regulates how many swaps will be considered, and thus can be used to trade off run-time for quality. Generally, the larger the *region_place_count* value, the better the results. However, Wang and Lemieux show that even when this parameter is set to a very large value, the quality difference between their algorithm and the sequential version reaches an asymptotic bound of 8%. Thus, other methods must be explored to determine whether the quality results can be further improved past this point.

B. Initial Best QoR

An initial experiment was performed to test if the Half-Box implementation could achieve a better QoR than the baseline algorithm. Both the baseline algorithm and the Half-Box implementation were run with large *region_place_count* values of 300 and 310. These data points should be close to the best possible quality as no significant quality improvement was noticed past $region\_place\_count = 180$ for the baseline algorithm. The results showed approximately 7% degradation in bounding box cost and 2-3% degradation in critical path delay. This is similar to the finding of Wang and Lemieux and shows that the Half-Box technique does not inherently improve the QoR. As the Half-Box technique provides similar QoR with better run time, all experiments in the remainder of this paper use the Half-Box approach.

C. Initial Placement

It is possible that the quality loss is directly due to the fact that thread regions restrict the swap distance. This restriction may hinder wide-scale block migration across the grid.

Feasibility: To explore whether this is a possibility, VPR with the Fast option specified (which sets *inner_num*=1) was run with a restriction on the maximum swap distance. Swaps were restricted to a maximum distance of 9. This roughly corresponds to the maximum swap distance for the parallel placer when using 25 threads and a grid size of 70x70, which is representative of the benchmark circuits. The data point for VPR Fast with a restricted swap distance is shown on a bounding box QoR versus run-time graph in Figure 9. The result is a QoR similar to that experienced by the parallel placement code. This correlation suggests that it is the restricted swap distance that is responsible for the quality loss.

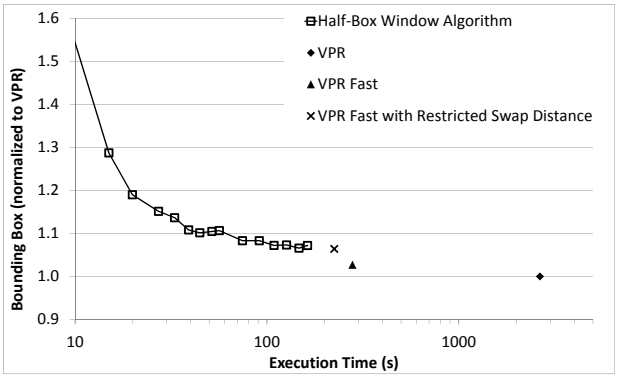

Fig. 9. QoR degradation of the bounding box cost versus run time for the Half-Box placer, including data points for VPR, VPR Fast, and VPR Fast with a restricted swap distance of 9.

One way to help blocks migrate more easily is to use an initial placement in which blocks have already migrated to a location close to their optimal position. To illustrate this, we ran experiments with an initial placement created by the original sequential VPR placer. Although this would not make sense in practice, since it would defeat any run-time gain by using the parallel placer, we used this experiment here to test our hypothesis. If using an excellent initial placement yields no quality improvement, there is no need to explore inferior initial placements. However, if an excellent initial placement does yield improvements, other methods of producing an initial placement of lesser quality, but with shorter run-time could be explored.

Results and Analysis: The standard sequential VPR was used to produce a placement for the circuit. This placement file is used as an initial placement for the Half-Box parallel placement execution. Tests were performed on all seven benchmarks using *region_place_count* of both 300 and 310. The results are similar to those in Section IV-B and show no improvement in QoR. The reason is that during the initial phases of simulated annealing, most swaps are accepted regardless of their impact on quality. This allows the process to produce a random placement before cooling to the best solution. In our experiments, we observed that this randomization completely negates the initial placement. The parallel placer is designed in such a way that each inner loop will consider swapping every block of the grid. This will occur multiple times with very high acceptance rates. This is likely sufficient to fully randomize the placement, even with restricted movement distances.

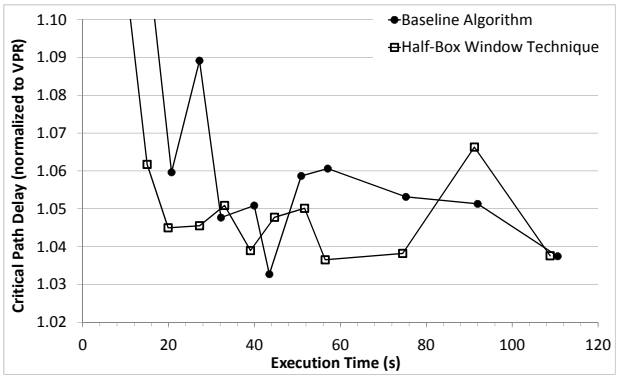

Fig. 8. QoR degradation of the critical path delay versus run time for the baseline algorithm and the Half-Box technique.

Success Rate	Change in parameter	
	temperature	region_place_count
> 0.96	old_t * 0.5	input_region_place_count
> 0.8	old_t * 0.9	input_region_place_count
> 0.15 or rlim > 1	old_t * 0.9	input_region_place_count / 4
otherwise	old_t * 0.6	input_region_place_count / 20

TABLE I
ANNEALING SCHEDULE OF WANG AND LEMIEUX'S PARALLEL PLACER.

Success Rate	Change in parameter	
	temperature	num_moves
> 0.96	old_t * 0.5	inner_num * $num\_blocks^{\frac{4}{3}}$
> 0.8	old_t * 0.9	
> 0.15 or rlim > 1	old_t * 0.95	
otherwise	old_t * 0.8	

TABLE II
ANNEALING SCHEDULE OF VPR.

Phase where $\alpha = 5$	Quality Loss against VPR	
	Bounding Box	Critical Path
None	4.8%	2.9%
> 0.96	5.0%	3.6%
> 0.8	5.0%	3.5%
> 0.15 or rlim > 1	4.5%	3.6%
otherwise	4.7%	2.9%

TABLE III
QUALITY RESULTS FOR AN ANNEALING SCHEDULE WITH BIASED SCHEDULER PHASES.

Although Figure 9 suggests that swap distances are responsible for the quality loss, there is no evidence that this prevents the early phases of the annealer from fully randomizing the placement. These results show that it is likely that the quality loss comes as a result of limited swap distance during cooling, and the cooling schedule itself, but not the randomization phases.

D. Annealing Schedule

The second area of investigation is whether the annealing schedule can be modified to improve the QoR. For all experiments in this section, the same methodology was used as in Section III-E.

The Baseline Algorithm: The annealing schedule used by Wang and Lemieux's parallel placer is presented in Table I, which can be contrasted with the original VPR scheduler in Table II. There are two major differences. First, the parallel placer algorithm uses an accelerated cooling rate, scaling the temperature by 0.9 instead of 0.95 and 0.6 instead of 0.8. Second, VPR uses *inner_num* to control the effort level, while the baseline algorithm uses *region_place_count*. It is not immediately clear how these two terms relate.

Relating region_place_count to inner_num: Without knowing how how *region_place_count* and *inner_num* are related, it is difficult to determine whether the parallel placer is considering more, equal, or fewer swaps than VPR. Knowing this would be helpful in evaluating the QoR against VPR. In

Success Rate	Change in parameter	
	temperature	num_moves
> 0.96	old_t * 0.4	inner_num * $num\_blocks^{\frac{4}{3}}$
> 0.8	old_t * 0.9	
> 0.15 or rlim > 1	old_t * 0.95	
otherwise	old_t * 0.8	

TABLE IV
THE IMPROVED ANNEALING SCHEDULE.

VPR, the number of swaps considered at each temperature is:

$$num\_moves = inner\_num * num\_blocks^{\frac{4}{3}}$$

The inner loop of VPR considers *num_moves* swaps between random blocks before moving on to the next temperature. The parallel placer is structured somewhat differently. For each inner loop iteration, every block on the grid has a 90% of being considered for a swap. *Region_place_count* dictates the number of inner loop iterations, and changes with temperature as shown in table I. Thus, the number of swaps considered by the parallel placer at each temperature is:

$$num\_moves = 0.9 * input\_region\_place\_count * num\_blocks * \alpha$$

Where α values of 1, 1, 0.25 and 0.05 are progressively used during cooling. It becomes apparent that the number of moves considered by the parallel placer grows linearly with the block count, while in VPR it grows by a power of 4/3. This may become an issue if the baseline algorithm is used for much larger test circuits, since it may mean that for the same level of effort (*region_place_count*), the QoR is worse. To fix this issue, and to align the placer with normal VPR conventions, the baseline algorithm was modified to employ the *inner_num* parameter instead of *region_place_count*. Since each inner loop iteration performs $0.9 * num\_blocks$ swaps, the number of inner loop iterations is set to $1.1 * inner\_num * num\_blocks^{\frac{1}{3}}$. Based on these equations, an inner_num of 4.7 in our algorithm gives an effective region_place_count of 90, the value used in Wang and Lemieux's results.

Biasing of Annealing Phases: The baseline algorithm uses different α values during phases of the annealing schedule. The weightings are selected so that more time is spent on earlier phases of the annealing, compared to VPR. Wang and Lemieux explain that this is done to "alleviate the effect of restricted block movement".

Experimentation was performed using different α weights during the four phases of the annealing schedule. Four different biased annealing schedules were considered, and for each case $\alpha = 5$ for one of the four phases and $\alpha = 1$ for the other three phases. A fifth experiment was performed with no biasing, where $\alpha = 1$ for all four phases. For all tests, *inner_num* = 20, which is twice the effort used by VPR. This was done to explore the limits of the QoR. The results of the experiment are presented in Table III.

The results with no α biasing showed an improvement over the baseline algorithm, with the bounding box cost degraded

4.8% compared to VPR and the critical path delay degraded 2.9%. This shows that with additional effort and a modified annealing schedule, the parallel placer can achieve QoR levels closer to VPR. In addition, the results of biasing different phases of the annealing schedule do not have a large impact on quality. We observed that biasing the later phases slightly improves quality, while biasing earlier phases slightly lowers quality. This is the opposite of what is suggested in Wang and Lemieux's paper. However, this does support the findings of the initial placement experiment (Section IV-C), which suggested that the QoR loss occurs in the later phases of the annealing schedule, not during the early randomization phase. More investigation is needed to investigate this discrepancy and the full effect on quality.

A Modified Annealing Schedule: Based on the previous findings, a modified annealing schedule was created that spends less effort on the early annealing phases and more effort on the later phases (Table IV). This schedule is very similar to the VPR annealing schedule, with the exception of faster temperature cooling in the first phase. Results of the improved algorithm are provided in the following section.

V. Final Results

Experiments were performed to compare the baseline algorithm to the improved algorithm that uses both the Half-Box technique and the new annealing schedule. Both 16-thread and 25-thread versions of the baseline and improved algorithms were tested. The 16-thread version was included here because low-cost, 16-core workstations will soon be widely available, while more than 16 cores will likely be more exotic and more expensive. The methodology described in Section III-E was used. In order to generate QoR versus runtime data, *inner_num* values were varied from 0.1 to 20. The results are illustrated in Figures 10 and 11. Finally, the speedup of the improved algorithm versus the baseline algorithm was calculated for various fixed levels of bounding box QoR. The speedup results are presented in Figure 12. The results show that both the 16-thread and 25-thread versions of the improved algorithm achieve up to a 70% speedup over the 25-thread baseline algorithm.

Testing was performed with fewer than 16 threads, but the results obtained were worse. Although the results show little improvement from 16 to 25 threads, this is likely due to the relatively small size of the benchmark circuits. As more threads are used, the region size decreases, restricting movement of blocks. The benchmarks use roughly 40,000 LUTs, whereas modern FPGAs support circuits up to 2,000,000 LUTs. Larger benchmarks would be required to properly evaluate the benefit of using more than 16 threads. Unfortunately, benchmarks of this size are not readily available.

In [5], Wang and Lemieux achieved a 123x speedup over VPR with approximately a 10% QoR degradation. However, these experiments were performed on an older architecture processor, the Sun Niagara 2, which has very slow single-thread performance. On the more modern 32-core AMD processor system used for these experiments, Wang and Lemieux's algorithm achieves only a 35x speedup for equivalent QoR degradation. In contrast, the improved algorithm presented in this paper achieves a 51x speedup over VPR (a speedup of 1.46x), and requires only 16 threads instead of 25. Hence, our new algorithm better exploits lower-cost workstations with fewer cores.

VI. Conclusion

This paper provides two significant enhancements to the FPGA parallel placer designed by Wang and Lemieux [5]. First, a new data decomposition scheme, the Half-Box technique is introduced, which reduces the number of synchronization barriers by 50%. This modification improves run time while providing the same quality of results (QoR). Secondly, this paper outlines the development of an improved annealing schedule, which spends less time in the early phases of annealing and more time during the later cooling phases, an approach contrary to Wang and Lemieux's experience. This

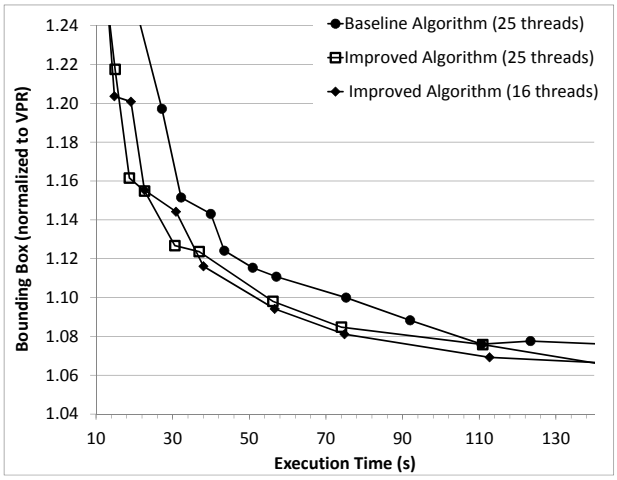

Fig. 10. QoR degradation of the bounding box cost versus run time for the baseline and improved algorithms.

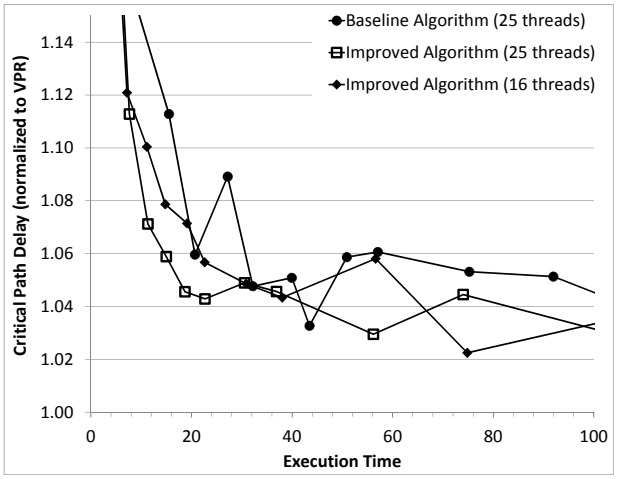

Fig. 11. QoR degradation of the critical path delay versus run time for the baseline and improved algorithms.

modification further improves run time, as well as providing slight QoR improvements.

The paper shows that even with a high level of placement effort, the QoR achieved does not equal that of VPR. However, it is closer than the original algorithm. The best QoR achieved is a 5% degradation in bounding box cost and 3% degradation in critical path compared to VPR, versus 8% and 3% with the original algorithm. We believe some of this residual QoR loss may be recovered by further tuning to the annealing schedule and algorithm tweaking, e.g., by range limiter adjustments during the low-temperature phase. However, we also believe that the limits of using stale data may eventually be reached before matching VPR's original QoR.

The combined effect of improving QoR slightly and reducing barrier synchronizations leads to significant run-time reductions. The Half-Box domain decomposition technique coupled with the improved annealing schedule provides up to 70% speedup over the original design at fixed levels of QoR using 25 threads. Furthermore, this 70% speedup can also be achieved with fewer threads, requiring only 16 threads in the improved algorithm compared to 25 threads in the original design. For a 10% degradation in QoR, our 16-thread improved algorithm achieves a 51x speedup over VPR, compared to only 35x speedup achieved by the 25-thread original algorithm. Hence, in addition to a performance improvement, a more modest, low-cost workstation can be used to achieve these good results.

ACKNOWLEDGEMENT

The authors would like to thank Chris Wang for providing his parallel placement source code and for assistance he provided in using it. In addition, we would like to thank the many contributors to VPR 5.0, particularly Vaughn Betz and Jason Luu, for making their source available. All of our source code changes will be made available, and ideally integrated into the new VPR 6.0 tree.

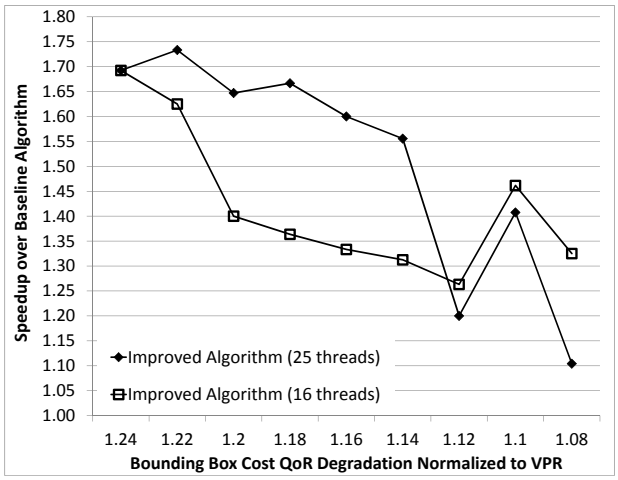

Fig. 12. Speedup of the improved algorithm over the 25-thread baseline algorithm for various bounding box QoR degradation levels.

REFERENCES

[1] S. Y. Chin and S. J. Wilton, "Towards scalable FPGA CAD through architecture," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011.

[2] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[3] K. Vorwerk, A. Kennings, and J. Greene, "Improving simulated annealing-based FPGA placement with directed moves," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 179 –192, 2009.

[4] R. Tessier, "Fast placement approaches for FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, pp. 284–305, 2002.

[5] C. C. Wang and G. G. Lemieux, "Scalable and deterministic timing-driven parallel placement for FPGAs," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011.

[6] Altera Corporation, "Quartus II 10.1 handbook," http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf, 2010.

[7] M. Santarini, "Xilinx tailors four tool flows to customer design disciplines in ISE design suite 11.1," http://xilinx.com/support/documentation/white_papers/wp307.pdf, 2009.

[8] A. Ludwin, V. Betz, and K. Padalia, "High-quality, deterministic parallel placement for FPGAs on commodity hardware," in *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, 2008.

[9] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "Parallel algorithms for FPGA placement," in *Great Lakes Symposium on VLSI*, 2000.

[10] P. Banerjee, M. Jones, and J. Sargent, "Parallel simulated annealing algorithms for cell placement on hypercube multiprocessors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 1, no. 1, pp. 91 –106, 1990.

[11] A. Choong, R. Beidas, and J. Zhu, "Parallelizing simulated annealing-based placement using GPGPU," *International Conference on Field Programmable Logic and Applications*, 2010.

[12] S. Kravitz and R. Rutenbar, "Placement by simulated annealing on a multiprocessor," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 6, no. 4, pp. 534 – 549, 1987.

[13] W.-J. Sun and C. Sechen, "A loosely coupled parallel algorithm for standard cell placement," in *IEEE/ACM International Conference on Computer-Aided Design*, 1994, pp. 137–144.

[14] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2009.

[15] M. G. Wrighton and A. M. Dehon, "Hardware-assisted simulated annealing with application for fast FPGA placement," in *International Symposium on Field-Programmable Gate Arrays*, 2003.

[16] 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 *IEEE/ACM International Conference on Computer-Aided Design*, 2006.