Architecture-Adaptive Range Limit Windowing for Simulated Annealing FPGA Placement

Ken Eguro Scott Hauck Akshay Sharma

Department of Electrical Engineering
University of Washington
Seattle, WA

{eguro, hauck, akshay}@ee.washington.edu

ABSTRACT

Previous research has shown both theoretically and practically that simulated annealing can greatly benefit from the incorporation of an adaptive range limiting window to control the acceptance ratio of swaps during placement. However, the implementation of such a system is not necessarily obvious. Existing range limiting techniques have several fundamental shortcomings when dealing with both standard island-style FPGAs and more exotic architectures. In this paper we discuss the nature of these problems and present a new algorithm that attempts to deal with these issues.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids – *Placement and routing.*

General Terms: Algorithms, Design.

Keywords

Reconfigurable logic, placement, simulated annealing, windowing, range limiting, architecture-adaptive.

1. INTRODUCTION

In [5], the authors discuss the relationship between move acceptance rate and magnitude of perturbation during simulated annealing, and their combined effect on overall solution quality and speed of convergence. They present a theoretical proof that roughly states that the best results can be obtained in the shortest amount of time when the ratio of moves accepted is kept as close as possible to 44% for the duration of the annealing. Towards this end they suggest an adaptive range limiter that controls the maximum distance by which any given block can be moved at different points of the annealing process.

Figure 1 shows us an example of the behavior of the acceptance rate during annealing. The bottom-most line assumes that our moveset allows any block in the array to be swapped with any other block in the array for the entire cooling schedule. Here we can see that we quickly migrate from nearly 100% acceptance, due to the very high temperature, to our target of 44%. However, we can also see that we would quickly drop below this acceptance rate as the annealing continues if we were to maintain this approach. Of course, once the acceptance rate drops too low, our annealing process makes very slow progress towards exploring new solutions.

Instead, as suggested in [5], when the annealing reaches the 44% crossover point we can maintain the target acceptance rate by introducing a range limiter window. Simulated annealing operates by always accepting moves that reduce or do not affect the cost function. Move that increases the cost function are accepted based on a probability that is directly related to the temperature and inversely related to how much worse the move makes the system as a whole. Thus, we can increase the acceptance rate by reducing the maximum amount by which any given block can migrate in a single move. This is both because a move of a smaller magnitude has a greater chance of being a zero-cost move and because any potential increase in the cost function would be naturally smaller. Thus, we can see that until we reach a range limit of one, the cooling schedule's temperature reduction can be compensated for by gradually shrinking the maximum distance by which we can move any block and the acceptance rate will roughly follow the dotted line in Figure 1 between the two extreme range limits.

VPR [1] adapts this method to be used for FPGA placement by defining an R_{limit} term that changes during the annealing. In this case, when we perturb the system we form an imaginary $(2R_{limit}+1)$ x $(2R_{limit}+1)$ bounding box centered on a particular block and the moveset randomly chooses to swap with another block within this bounding box. Although this system has been

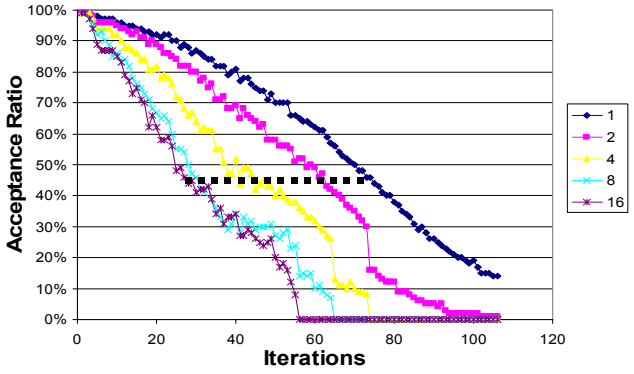


Figure 1. Acceptance rate during annealing for constant maximum move range.

DAC 2005, June 13–17, 2005, Anaheim, California, USA.

widely accepted as the de facto standard for FPGA placement range limiting, our paper will discuss some of the shortcomings of this approach both on island and non-island style FPGAs. This will motivate our introduction of a new technique that better embodies the original intent of the theoretical work done in [5].

2. VPR Range Limit Windowing

The range limiting functionality of VPR, detailed in [2], selects, with equal probability, any of the blocks that have an X and Y location within R_{limit} of a block under consideration. This approach is simple to implement, and if we examine the Manhattan distance of each of the possible target locations we can easily determine the likelihood of selecting a given distance move. As can be seen in Figure 2, a range limit window of $R_{limit} = 2$ encompasses eight locations that would result in either a distance two or three move and four that would result in either a distance one or four move. In Figure 3 we extrapolate out to the case of $R_{limit} = 5$. If we assume an infinite-sized FPGA (to avoid edge effects), the fact that any block within the range limit window has equal probability of being chosen creates the distribution of possible length moves shown in the right-most graph in Figure 3. We can see that moves of length five and six are significantly more likely than shorter moves and the probability of selecting blocks outside of the $R_{limit} = 5$ window drops to zero.

3. Implications of Hard Macros, Incremental Placement and Routing-Poor Architectures

Unfortunately, since this technique only considers swaps within a hard rectangular bounding box, it fails when not allowed free reign over the entire architecture during placement. First let us consider the case in which we are given large pre-placed and routed macros or are attempting incremental place and route for new portions of a circuit around unmodified sections. Seen in Figure 4, if these unchangeable portions of the circuit occupy the center of the array, when we attempt to place the rest of logic the range limiter window quickly eliminates any possibility of swapping blocks between the left and right sides. Effectively, the very early, high-temperature stages of the annealing process completely determine the partitioning of the circuit.

We can also consider the situation in which we forbid the occupation of certain locations before or during the placement process. One simple example we encountered occurred during follow-on work to explore some of the issues regarding placement on routing-poor architectures that we first introduced in [4]. Consider the example in Figure 5 (top) in which we place a 6-block circuit within a much larger array. Since the normal VPR toolflow does not consider the distribution of routing resources during the placement phase, it will place the circuit as tightly as possible. If the routing channels are not wide enough to support this close placement, routing will fail. Notice that there might actually be a routable placement, but in this case, the majority of the routing resources are inaccessible since they are in unoccupied regions of the chip.

If we were to take that circuit and evenly space the blocks to cover the entire array we can roughly double the usable routing in both the horizontal and vertical directions. One very simple algorithm that might be able to generate such a depopulated mapping forbids the placement tool from ever occupying three out of every group of four logic blocks. Unfortunately, when we anneal on such a depopulated architecture, the placement will eventually fail when the range limit window reaches one, since there are no valid locations for any of the occupied blocks to swap with, as shown in Figure 5 (bottom).

Although this specific case can be dealt with simply enough, problems such as these become much more difficult to solve as the depopulation becomes more sophisticated or less predictable. For example, it would be very difficult to design an algorithm that could deal with arbitrary depopulation due to incremental place-and-route or hard macros. Another extremely relevant example, we might desire a tool that can place and route around randomly

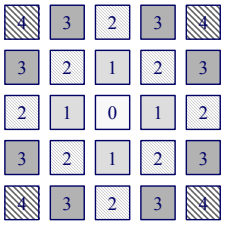


Figure 2. Manhattan distance within $R_{limit} = 2$ bounding box.

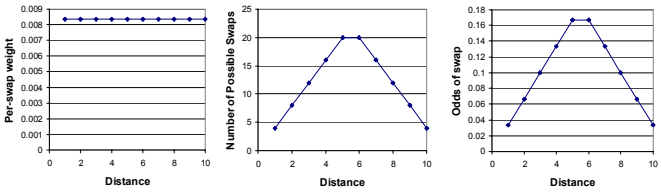


Figure 3. Probability of a distance N swaps using VPR's bounding box-based range limiter. The per-swap weight (left) shows the even probability assigned to each move of distance N. Given the distribution of possible moves within the square range limiter (middle), we get the overall distribution of moves (right).

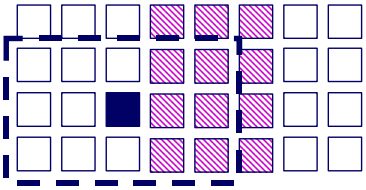


Figure 4. Placement disrupted by previously placed and routed blocks.

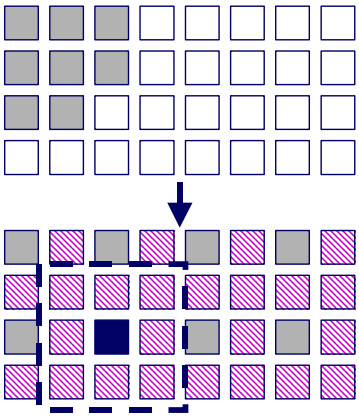


Figure 5. Placement halts when $R_{limit} = 1$.

occurring chip defects. This concern is particularly important given the manufacturing difficulties that will likely be encountered towards the end of the silicon roadmap and the extremely high defect rate of proposed sub-lithographic nanotube and nanowire architectures.

4. Implications of Hierarchical Architectures

Another concern is that a hard range limiter will have particularly poor performance when used on hierarchical architectures. Of course, the purely Manhattan distance nature of customary range limiting is not appropriate for such architectures, but even if we adapt these techniques so that we limit movement based upon the communication distance between two blocks, a hard windowing approach will exhibit an unusual partitioning behavior during placement. Consider the example shown in Figure 6. Here we show a hierarchical FPGA consisting of 16 logic blocks and a range limit window of distance four. Notice that the logic block in the bottom left corner can only be swapped with the three other blocks in its cluster. When the range limit distance is reduced from seven to six to four, logic blocks are permanently locked within their sixteen, eight, or four member clusters, respectively.

Notice that this is dramatically different from the behavior of range limiting in the island-style FPGA case. Although the range limiter window prevents a logic block from moving beyond a particular distance in a single swap, the logic block could eventually move arbitrarily to any location across the array given multiple perturbations. This does not hold true in the case of hierarchical architectures. Instead, each time we reduce the range limit distance we effectively recursively bi-partition the circuit.

5. Distance-Based Weighting

Even considering island-style FPGAs, the standard VPR range limiter technique described above suffers from the artificial constraints of a hard rectangular bounding box. That is, it completely disregards locations outside of the range limiter window even though it does consider equivalent locations inside. As seen in Figure 7, there are multiple locations outside of a $R_{limit} = 2$ window that would constitute a swap of length three or four that should logically be included for consideration. We would also like to have a windowing technique that works for any FPGA architecture (such as HSRA) or placement situation (such as the depopulated or macro-based placement).

To address these concerns we completely eliminate the concept of a hard range limit and instead weight the probability of swapping any two blocks in the array solely based upon their distance from each other:

$$P_i = \frac{1}{D_i^M}$$

Here, D_i represents the distance between two blocks and M is a new dynamically adjusted factor that takes the place of the former R_{limit} term to control the acceptance rate during the annealing process. Notice that when M equals zero, we do not have any range limiting. The left-most graph in Figure 8 shows how the weighting factor of each length move changes across a range of M values. We can also see the probability for swaps from the center of an 11 x 11 block array for the same range of M values in the right-most graph. Although we discuss the implementation more thoroughly in the next section, we can see that the annealing process should begin with a small M factor to allow for large magnitude

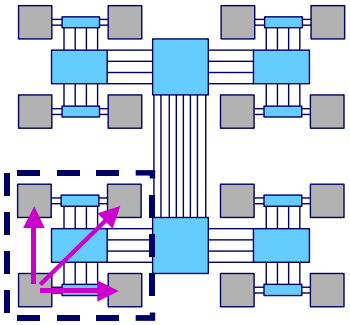


Figure 6. Range limiters on hierarchical architectures.

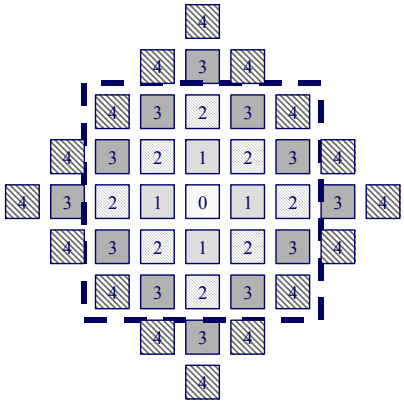


Figure 7. Manhattan distance of blocks outside of the rectangular range limiter window.

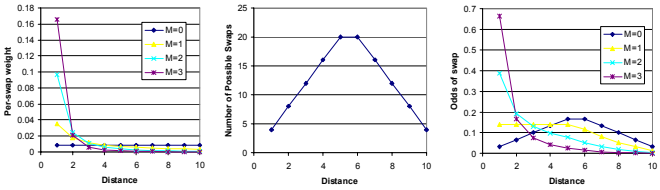


Figure 8. Probability of a distance N swaps using distance-based range limiter

moves. This term should then be gradually increased to reduce the likelihood of long distance moves as the temperature is reduced. Observe the largest difference between this approach and traditional windowing, the probability of selecting even the longest swap never reaches zero regardless of how large M gets.

6. Implementation Details

Implementing the architecture-adaptive windowing presented in this paper is more complex than the traditional method, but is still relatively straightforward. First, we need the distance between all logic blocks in the architecture. This can either be given in an architecture description file or can be pre-calculated via an all-pairs shortest path search. The more complex issue is how to determine the proper M factor between iterations in order to roughly maintain the 44% acceptance ratio. To do this, we make a somewhat inaccurate assumption – we pick M such that it would have achieved a 44% acceptance ratio in the previous iteration. This is inexact since the next iteration will have a lower temperature, and thus the probabilistic acceptance function will likely accept fewer bad moves. However, we have found this to work acceptably in practice. Adding an empirically-determined correction factor, such

as picking M to achieve a 44%+Δ acceptance ratio in the previous iteration, would be interesting future work.

To determine M, during each iteration we record arrays of the number of attempted and accepted moves for each distance. We can then compute the acceptance ratio at each distance during the previous iteration. Given a pre-calculated array of the number of possible moves for each distance, we can estimate the overall acceptance ratio in the previous iteration for a given M using the formula:

$$\frac{\sum_{D_i=1}^{D_{max}}\left[NumPosSwaps[D_i]*\left(\frac{1}{D_i}\right)^M*\frac{NumAccept[D_i]}{NumAttempt[D_i]}\right]}{\sum_{D_i=1}^{D_{max}}\left[NumPosSwaps[D_i]*\left(\frac{1}{D_i}\right)^M\right]}$$

We can then find the M that would have achieved a 44% acceptance ratio via a binary search. Range limits can also be imposed on M, and we found empirically that clamping the value between one and three provides reasonable results.

Move generation is also more complex in distance windowing because of the irregular weights. As seen in Figure 9, we first create a two-dimensional array *SwapOdds[from][to]*, where the indices are the possible source, *from*, and destination, *to*, of a swap in the architecture. These values are calculated at the beginning of each iteration based upon the M computed above and the distance of the swap. Invalid swaps, such as an IOB to CLB swap are set to 0. We then compute an array *CumeSwapOdds[from][to]* as shown in Figure 9. Specifically:

$$CSO[i][0] = SwapOdds[i][0]$$
$$CSO[i][j] = CSO[i][j-1] + SwapOdds[i][j]$$

With the creation of the *CumeSwapOdds* array, we now have an efficient move generation method. We first pick a random "from" node, then a random number R on the interval [0,1). We can then perform a binary search on *CumeSwapOdds[from]* to determine which other block represents the interval that includes this random number.

7. Results

7.1 Conventional Island-Style Architectures

In Figure 10 we provide a comparison of range limiter techniques on the 20 MCNC benchmarks included in the VPR suite. Here we report the channel width of the best of five placement and detailed routing runs on the 4lut_sanitized architecture using bounding box placement and breadth-first routing. "VPR" indicates that we used normal VPR windowing in which we use the built-in dynamically controlled square range limit window within which all swaps have the same probability. "Distance" refers to our suggested technique in which we eliminate the hard range limit window entirely and any block can be swapped with any other in the array with a probability proportional to $(1/D)^N$. For a point of reference we also include two other tests, "$R_{limit} = \infty$" in which we eliminate all range limiting and allow all length swaps with equal probability, and "$R_{limit} = 1$" in which we only allow swaps between a block and the immediately surrounding blocks. One issue when comparing these windowing techniques is that a change in the moveset can also affect the cooling schedule. Although the CPU time per move was approximately the same for all of the windowing approaches, the adaptive nature of the cooling schedule described in [1] makes the total number of temperature iterations for every placement run variable, even given different seeds using the same windowing technique. To account for this, we have not only conducted multiple runs of each technique using different random seeds, but also varied the number of moves per iteration to build the curves shown in Figure 10.

As predicted by the curves we introduced in Figure 1, eliminating range-limiting altogether clearly causes the annealing to settle on mediocre results when the acceptance ratio becomes too small to explore new solutions. Although restricting all movements to nearest-neighbor performs slightly better, it likely runs into problems adequately exploring diverse portions of the problem space, particularly for shorter placement runs. Although since we report detailed routing results and not merely placement cost there is some noise in the data, we can also see that our window-less distance range limiter generally performs as well or better the original VPR algorithm. It is also quite possible that there is room for further improvement. Any gains provided by a more graceful distribution of possible swap lengths might be offset by our very rudimentary adaptive weighting function and we might spend too much time exploring inappropriate length moves. Although we would like to explore this further and refine our probability weighting function, most importantly this testing shows that our new system performs at least as well or better than the state-of-the-art adaptive VPR range limiter.

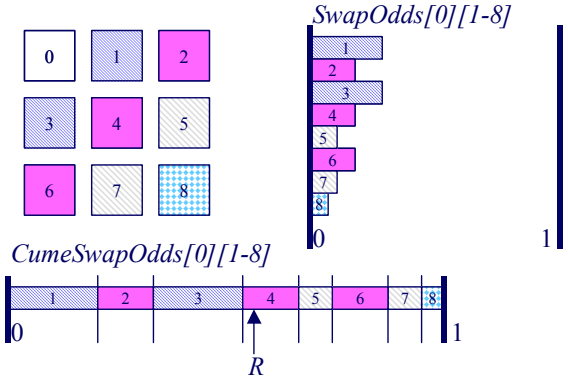
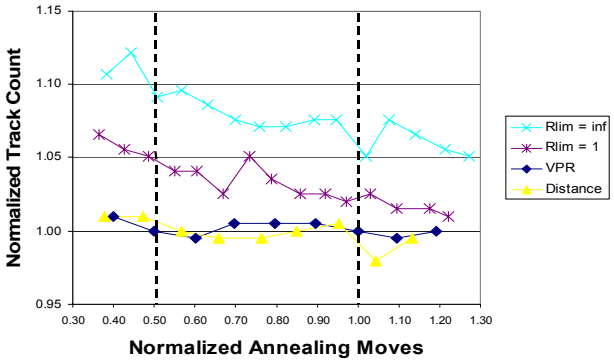


Figure 9. Implementation of move generation



	VPR	Distance	$R_{limit} = \infty$	$R_{limit} = 1$
Place Move	0.50	0.47	0.51	0.49
Tracks	**1.00**	**1.01**	**1.09**	**1.05**
Place Move	1.00	1.04	1.02	0.97
Tracks	**1.00**	**0.98**	**1.05**	**1.02**

Figure 10. Comparison of range limiter techniques – Detailed routing channel width for 20 MCNC benchmarks

7.2 Depopulated Architectures

Of course, conventional placement on an island-style FPGA is not necessarily where we expect the distance-based weighting system to truly show its value. Figure 11 and Figure 12 show how the architecture adaptive window-less range limiter compares to a custom-designed solution for a depopulated architecture.

In this testing we determined the baseline routing channel width for each of the twenty benchmarks included in the VPR suite on the 4x4lut_sanitized architecture using the default placement and routing options. We then re-placed and routed the benchmarks on arrays with fewer routing tracks per channel and determined how much larger an array was needed to compensate for the thinner routing channels.

As seen in Figure 11, the normal VPR toolflow does not consider routing density during placement and thus produces tight configurations in which the extra routing capacity afforded by a larger array cannot be utilized. Arrays up to four times minimum size were considered before a technique was deemed a failure. We can see that none of the twenty benchmarks are successfully placed and routed using normal VPR placement once we reach a routing channel width of 70% the baseline.

To account for this we repeated the testing, but evenly distributed specifically forbidden blocks starting from the initial placement. For every size array we determined how many unoccupied blocks there would be in the architecture and uniformly "locked" the extra logic locations throughout the array. We then placed the benchmarks on these arrays using both our distance-based probability weighting system and a slightly modified VPR windowing system in which instead of clamping R_{limit} between the maximum size of the array and one, we clamp the value between the maximum size of the array and two. Notice that since we tested architectures that have down to half of the "suggested" channel width, feasible placements almost certainly exist for arrays four times the minimum size. In this case we will rarely forbid more than three out of every group of four blocks, as shown in Figure 5, so restricting R_{limit} to a minimum of two will always allow an occupied block to find at least one other valid location to swap with during placement.

As seen in Figure 11, the depopulation technique indeed finds valid placements and routings for the entire suite of twenty benchmarks in all but the highest stress cases. As seen in Figure 12, our distance-based range limiter performs on par with the modified VPR windowing system using 29% fewer placement moves. Although due to computing time and resource constraints we could not repeat an exploration of placement effort versus quality as in Section 7.1, producing virtually identical results with a large difference in placement moves indicates again, that at the very least our new windowing technique performs as well or better than VPR's more traditional formulation.

More importantly, to use the existing VPR hard windowing approach we needed to manually adjust the parameters on the limits of the range window to prevent the placement from entering an infinite loop when R_{limit}= 1. Not only does this potentially require the intervention of a specialist, determining the lower limit for the range window is not obvious if the distribution of forbidden locations is not predictable, as in the cases mentioned earlier of hard macros, incremental placement or placement in the face of manufacturing defects.

7.3 Hierarchical Architectures

We also expect that a hard range limiter will perform poorly when considering hierarchical architectures. To test this theory we examine benchmarks on the HSRA[6] architecture. In this architecture, logic blocks reside at the leaf nodes and are interconnected via a tree structure that is defined by a base channel width, or the number of tracks connecting neighboring pairs of leaves together, and an interconnect growth rate, which defines how rich the routing resource are compared to the base channel width as we move up the tree.

For our testing we began by randomly selecting fifteen netlists that require between 128 and 512 nodes out of the 180 netlists examined in [3]. To determine a baseline architecture for each netlist we used the HSRA placement tool also from that paper. This placement tool takes in two parameters in addition to the netlist: a target base channel width and an interconnect growth rate. Based upon these three factors the tool not only produces a placement but also determines what it believes to be an appropriate width, or *lsize*, architecture. Notice that the width of the architecture also determines the number of interconnect levels or height of the architecture. If we were following the normal toolflow this information would be forwarded to the HSRA routing tool, *arvc*. Here, based upon the specified interconnect growth rate the router determines the minimum base channel width – normally larger than the target base channel width provided to the placement tool.

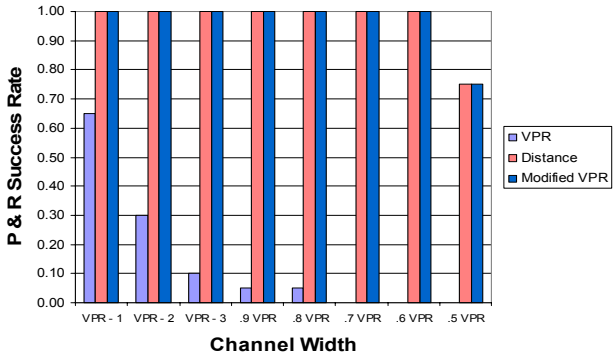
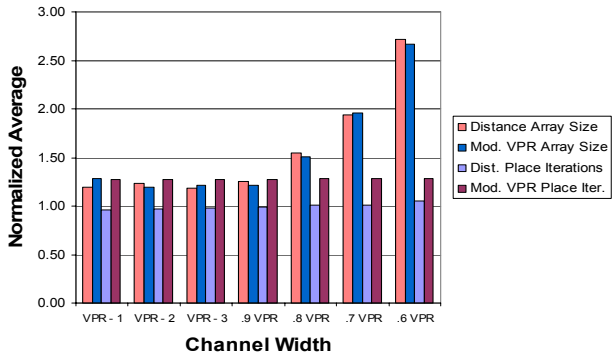
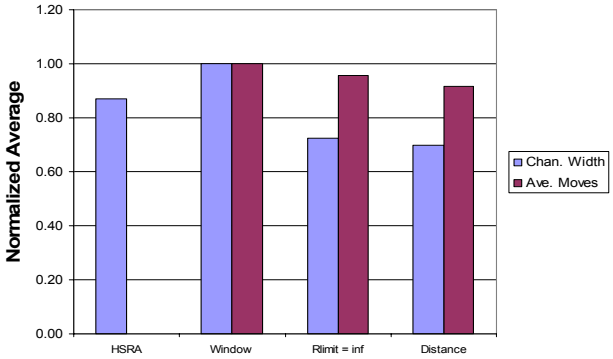


Figure 11. P&R success rate on routing-poor architectures



	Distance	Modified VPR
Ave. Array Size	**1.58**	**1.58**
Ave. Place Moves	0.99	1.28

Figure 12. Routing-poor architecture results. Average across 20 MCNC benchmarks and normalized to default VPR results.



	HSRA	Window	$R_{limit} = \infty$	Distance
Chan. Width	**0.87**	**1.00**	**0.72**	**0.70**
Place Iterations	-	1.00	0.96	0.92

Figure 13. HSRA testing results. Average results normalized to conventional hard range limiting data.

We have developed our own simulated annealing-based placement tool for HSRA [6], which produces very high-quality placements for HSRA. To determine the effect of different range limiting methodologies on placement quality we implemented three range limiting techniques: a hard adaptive range limit window based upon communication distance, as described in Section 4; a windowless distance-based range limiter, as described in Section 5; and no range limiting. We used the HSRA placement tool to determine an appropriate *lsize* for each netlist given a target base channel width of eight and an interconnect growth rate of 0.5. These parameters were chosen to provide a medium-stress placement problem for all of the netlists. We then fed the *lsize* determined by the HSRA placement tool and the netlist into our own tool to get three placements, one for each range limiting technique. These placements were then fed into *arvc* to provide the results shown in Figure 13.

As predicted, a hard range limit window ends up unnaturally partitioning the circuit very early in the annealing process, producing poor results. We can see that the original quadratic/partitioning-based HSRA placement tool is able to beat this approach by an average of 13% base channel tracks. One surprise, however, is that the naïve windowing technique of allowing all swaps during the annealing process easily outperforms both the hard range limit window and original HSRA placement tool by an average of 28% and 15% respectively. Finally, our adaptive window-less distance-based technique surpasses all of the other techniques: quadratic placement/recursive bi-partitioning, adaptive hard windowing, and no range limiting by 30%, 17%, and 2% fewer base routing tracks.

Again, due to computing time and resource constraints we could not repeat a detailed exploration of placement effort versus quality as in Section 7.1. However, in the brief testing we were able to perform we determined that, similar to the results in Figure 10, we do not expect the amount of required routing to change dramatically even over a wide range of placement moves and that our architecture-adaptive windowless range limiter will consistently out-perform other approaches.

8. Conclusions

In this paper we have shown that although range limiting techniques have the ability to greatly improve the convergence speed and quality of simulated annealing placement, existing methodologies has fundamental shortcomings when dealing with both island-style and hierarchical FPGA architectures. Not only is there a question about solution quality, the conventional approach has significant issues that might limit its use considering modern problems such as incremental or defect-tolerant placement. In the worst case, conventional placement tools may not function at all, requiring specialist intervention to develop problem-specific solutions.

We presented a new range limiting technique that attempts to address these issues. Our approach eliminates the concept of a range limit window entirely. Instead, any block has the possibility of swapping with any other block in the array throughout the annealing process and we control the movement acceptance ratio by changing the relative probability of longer versus shorter length moves. This technique, without any tuning or special cases, provides an architecture-independent methodology that performed equal to or better than standard windowing in all three placement situations considered: conventional island-style placement, island-style placement in the face of block restrictions and hierarchical architectures.

9. Acknowledgements

We would like to thank André DeHon at Caltech for providing the HSRA toolflow and helping us to understand various aspects of the architecture.

10. References

[1] Betz, Vaughn and Jonathon Rose. "VPR: A New Packing, Placement and Routing Tool for FPGA Research." International Workshop on Field Programmable Logic and Applications, 1997: 213-22.

[2] Betz, Vaughn, Jonathan Rose, and Alexander Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, 1999.

[3] DeHon, A. "Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization)." International Symposium on Field Programmable Gate Arrays, 1999: 125-34.

[4] Eguro, K. and S. Hauck, "Issues of Wirelength Cost Models in Routing-Constrained FPGAs", University of Washington, Dept. of EE Technical Report UWEETR-2004-0006, 2004.

[5] Lam, J. and J. M. Delosme, "Performance of a New Annealing Schedule." Proc. 25th Design Automation Conf., 1988, pp. 306-11.

[6] Sharma, A., C. Ebeling, and S. Hauck. "Architecture Adaptive Routability-Driven Placement for FPGAs". Submitted to International Symposium on Field Programmable Logic and Applications, 2005.

[7] Tsu, W., K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon. "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array". International Symposium on Field Programmable Gate Arrays, 1999: 125-34.