Runtime and Quality Tradeoffs in FPGA Placement and Routing

Chandra Mulpuri
Department of Electrical Engineering
University of Washington, Seattle, WA 98195, USA
chandi@ee.washington.edu

Scott Hauck
Department of Electrical Engineering
University of Washington, Seattle, WA 98195, USA
hauck@ee.washington.edu

Abstract

Many applications of FPGAs, especially logic emulation and custom computing, require the quick placement and routing of circuit designs. In these applications, the advantages FPGA-based systems have over software simulation are diminished by the long run-times of current CAD software used to map the circuit onto FPGAs. To improve the run-time advantage of FPGA systems, users may be willing to trade some mapping quality for a reduction in CAD tool runtimes. In this paper, we seek to establish how much quality degradation is necessary to achieve a given runtime improvement. For this purpose, we implemented and investigated numerous placement and routing algorithms for FPGAs. We also developed new tradeoff-oriented algorithms, where a tuning parameter can be used to control this quality vs. runtime tradeoff. We show how different algorithms can achieve different points within this tradeoff spectrum, as well as how a single algorithm can be tuned to form a curve in the spectrum. We demonstrate that the algorithms vary widely in their tradeoffs, with the fastest algorithm being 8x faster than the slowest, and the highest quality algorithm being 5x better than the least quality algorithm. Compared to the commercial Xilinx CAD tools, we can achieve a 3x speed-up by allowing 1.27x degradation in quality, and a factor of 1.6x quality improvement with 2x slowdown.

Keywords

FPGAs, Computer-aided Design, Placement, Routing, Fast CAD for FPGAs

1. Introduction

Most CAD development efforts have focused on the creation of as efficient a mapping as possible for a given computation. The application of complex optimization techniques, for the solving of multiple NP-Hard problems, has yielded efficient mapping tools that can take hours to produce an implementation.

For the design of ASIC circuits, producing the highest quality results at the cost of significant runtimes is justified by the long fabrication times and large costs. However the development of FPGAs, where a new computation can be realized in hardware in milliseconds, may require the re-evaluation of this tradeoff.

New FPGA-based systems push the issue of CAD tool runtimes into the spotlight. In custom-computing systems, the FPGA hardware is often used as a form of software accelerator, where designers expect fast turnaround from specification to implementation. In some systems the runtimes can even become part of the execution time of the system, where slow CAD performance directly impacts the utility provided to the user. For example, in logic emulation a circuit under development may need to be remapped to the accelerator on a weekly, daily, or even hourly basis, as modifications are made to the circuit while it is debugged. The longer the CAD tools take to operate, the smaller the advantage the emulation system has over software simulation, since simulators typically do not require such sophisticated pre-processing.

Emulation is in fact one example of problem-specific compilation in custom-computing devices. In such systems the execution of the system involves first creating an FPGA (or multiple FPGA) configuration(s) from a specification for a given problem instance, and then executing the configuration on the FPGA hardware.

In many FPGA-based systems the CAD tool performance can thus be a critical concern. In fact, users may be willing to trade some mapping quality (typically measured in critical path length and/or device capacity) for a reduction in CAD tool runtimes. For example, users may have excess FPGA capacity available to accelerate the mapping process. Alternatively, a slowing down of the FPGA execution because of lengthened critical paths may be more than balanced by the decrease in CAD runtimes, yielding an overall performance increase. However, what is unclear is how much quality must be yielded for a significant improvement in runtimes.

What is allowable in tradeoff depends on the applications. For some systems no reduction in mapping quality is acceptable (and in fact, for some systems only hand-design yields the required mapping quality). For others, larger quality reductions may be justified.

While there is no one rule for what is acceptable, there are some applications, like logic emulation where such allowable

tradeoffs are evident. For example Quickturn has simultaneously sold two different commercially viable emulation technologies, the CoBALT and Mercury systems. While each of these systems are capable of supporting roughly equivalent circuit complexities, and have equivalent system costs, the CoBALT system provides more than an order of magnitude reduction in mapping time (days to hours) at the cost of 1-2 orders of magnitude increase in system delay (MHz to 100 KHz performance). The fact that both systems were simultaneously commercially viable indicates that, at least for some applications, users are willing to accept huge quality losses for significant CAD runtime improvements.

In some instances, the slow runtimes of current CAD tools completely eliminate some opportunities for FPGA-based systems. For example, there has been much interest in problem-specific custom-computing for Boolean Satisfiability, where the inherent parallelism of FPGAs provides high-performance hardware to finding solutions to arbitrary Boolean equations. In [1] a comparison of software runtimes and total solution time using FPGAs for SAT solver circuits reveals that while there is 94.8x speedup that could be achieved over software in solving a200_6_0_y1_1 problem, large compilation times using a commercial CAD tool resulted in a 781x slowdown for the FPGA based solution. As the complexities of target circuits and FPGAs increase, the effectiveness and efficiency of these CAD tools become even more important in such applications. Techniques that accelerate core CAD algorithms can bring about important changes in product design times for these applications.

Though several works like VPR [2], Ultra Fast Placer [3], and Parallel Pathfinder [4] have concentrated on speeding up the individual CAD processes, not much has been documented in the way of actual trade-offs involved in choosing different algorithms for different CAD processes. In this paper we consider the tradeoff between the runtimes of CAD algorithms and the resulting critical path lengths of the resulting mappings. We seek to establish how much quality degradation is necessary to achieve a given runtime increase. As part of this process we investigate multiple placement and routing algorithms for FPGAs. We also develop new tradeoff-oriented physical design tools, where a tuning parameter can be used to control this balance. As part of these efforts we show both how different algorithms can achieve different points within this tradeoff spectrum, as well as how a single algorithm can be broadened in its applicability.

The rest of the paper is organized as follows. Section 2 briefly describes the experimental setup and the evaluation methodology for different algorithms. Section 3 describes the various placement algorithms implemented, and how these different place algorithms compare in terms of run-times and quality of the solution. Section 4 describes all the route algorithms implemented and compares their performances. Section 5 analyzes the trade-offs within different combinations of place and route algorithms. Section 6 looks at the dependence of the performance of algorithms on benchmarks and the sizes of the target FPGAs.

2. Implementation details

In order to quantify the tradeoff between CAD tool runtimes and the resulting quality, we implemented multiple placement and routing algorithms. Our algorithms are targeted to the Xilinx XC4000E family of FPGAs [5]. For representing the exact logic and routing resources in this FPGA architecture, as well as for the LCA format file input and output, we utilized the routines developed at the University of California, Santa Cruz for their implementation of parallel pathfinder [4]. These routines were originally developed for the XC4000 family of FPGAs. We retargeted them to the XC4000E series of FPGAs.

Our results were obtained by running the algorithms on SUN UltraSparc 5 workstations with 512 MB of memory. Twelve combinatorial benchmarks were used from the MCNC benchmark circuits [6], and range in size from 189 logic blocks to 1020 logic blocks. The properties of the benchmarks we used are summarized in the following table.

Benchmark	FPGA device	Number of nets	Number of CLBs
k2	4005E	261	189
misex3	4005E	244	192
alu4	4005E	276	194
seq	4008E	629	300
apex4	4010E	1235	388
tseng	4013E	1099	542
ex5p	4013E	1072	570
diffeq	4020E	946	751
dsip	4020E	1093	780
s298	4025E	1304	1002
des	4025E	1360	1013
bigkey	4025E	1501	1020

The algorithms were evaluated based on the comparison of their run-times to the delay of the mapped circuit. The critical path delay results were obtained by using Xdelay, which is part of the commercial Xilinx CAD tools.

3. Placement

The logic circuit that is to be placed and routed is specified in terms of configurable logic blocks, which are the basic logic elements that make up the array architecture of the FPGA. Placement is essentially assigning a unique position inside the FPGA to each of the circuit's configurable logic blocks.

We have implemented four different algorithms for placement, and a fifth placer was obtained from Xilinx. Our aim was to compare each of these placers in terms of their run-time vs. quality characteristics.

As part of this work we have developed runtime-adaptive versions of Simulated Annealing and Force-directed placement. In these algorithms, a balance parameter is introduced which can apply more or less effort, trading runtimes for resulting quality. These algorithms therefore are represented on the run-time vs. quality graph not by a single point, but by a set of points corresponding to different values of the tuning parameter. The placement algorithms we used are briefly explained below.

3.1 Placement Algorithms

3.1.1. Fiduccia-Mattheyses

This implementation is based on the Fiduccia-Mattheyses algorithm [7]. The FPGA is divided into two halves, and the Fiduccia-Mattheyses algorithm is applied to determine which logic block goes into which half. These two halves of the FPGA are further partitioned into two halves each, and this recursive process is applied on each FPGA partition until the partitions become small (contain less than nine CLBs). A greedy algorithm then determines the exact placement of the logic blocks corresponding to each of the small partitions. During the bipartitioning, the logic blocks in both partitions are arranged in a decreasing order of the gain obtained if they were to be moved across the partition. This gain is computed using the "Terminal Propagation" technique [8], which considers nets connected to logic blocks from other partitions as well. Once a logic block is moved across the partition, it is locked and the gains are updated for the rest of the logic blocks. This process is repeated until there is no further cost improvement.

3.1.2. Force-directed

This implementation is based on the Force-directed algorithm [9]. From an initial random placement, each logic block is moved to its "best location", which is determined as the closest available location to the centroid of all the other logic blocks to which it is connected. If another logic block already exists at this location, the two logic blocks are interchanged. After this move, the logic block in consideration is "locked", and the location is considered unavailable for other logic blocks. Once all the logic blocks are locked, they are unlocked and the process is repeated until a terminating condition is met.

The terminating condition dictates the per-iteration percent change in cost at which the algorithm will halt. If this percentage is small, the algorithm will perform significant optimization, with a commensurate increase in runtimes. With a large percentage change (including only performing a single iteration regardless of change in cost) the algorithm will perform a much lower quality optimization, but with much faster runtimes. We therefore use the stopping criteria as a tuning parameter for the Force-directed Placement algorithm.

3.1.3. Scatter

The fastest possible legal placement algorithm would simply randomly scatter the logic blocks across the chip area. For circuits that are fairly small, or with very easy requirements,

such an algorithm might achieve usable results with very fast runtimes. However, if the logic blocks are placed very close together, the resultant congestion will affect the route times for the worse. The Scatter algorithm therefore does arbitrary placement, but ensures that logic blocks are placed reasonably far apart.

3.1.4. Simulated Annealing

The Simulated Annealing implementation is based on the VPR placer [2]. The various simulated annealing parameters that define the algorithm are explained below:

Cost function: This cost function is described by the equation:

$$Cost = \sum_{n=1}^{N_{nets}} q(n) \, [\, bb_x(n) + bb_y(n) \,]$$

The summation is over all the nets in the circuit. For each net, bb_x and bb_y denote the horizontal and vertical spans of its bounding box respectively. The $q(n)$ factor is a value used to compensate for the underestimation of cost when nets have more than 3 terminals, and is obtained from a lookup table.

Initial Temperature: Let N_{blocks} be the total number of logic blocks and I/O pads in the circuit. From an initial random placement, N_{blocks} random pairwise swaps of logic blocks or I/O pads are performed, and the standard deviation of the cost is computed. The initial temperature is set to 20 times this standard deviation.

Cooling Schedule: The new temperature is computed as $T_{new} = \alpha \, T_{old}$, where the value of α depends on the fraction of attempted moves that were accepted (R_{accept}) at T_{old}, as shown:

Fraction of moves accepted (R_{accept})	α
$R_{accept} > 0.96$	0.5
$0.8 < R_{accept} \le 0.96$	0.9
$0.15 < R_{accept} \le 0.8$	0.95
$R_{accept} \le 0.15$	0.8

Since it is desirable to keep R_{accept} near 0.44 for as long as possible [10], the algorithm uses a range limiter which swaps blocks that are less than D_{limit} units apart in either the horizontal or vertical direction. If R_{accept} were less than 0.44, D_{limit} would be reduced thereby forcing moves over a smaller range and hence greater acceptance. This D_{limit} is updated across temperatures according to:
$D^{new}_{limit} = D^{old}_{limit} * (1 - 0.44 + R^{old}_{accept})$
and then clamped to $1 \le D_{limit} \le$ maximum FPGA dimension.

Final Temperature: The annealing is terminated when the temperature is less than 0.5% of the average cost per net.

Number of moves at each temperature: In the original VPR placer, at each temperature $10*(N_{blocks})^{1.33}$ moves are evaluated. However, the number of moves evaluated at each temperature directly controls the amount of time Annealing spends in searching for a good solution. Increasing or decreasing this number will directly increase or decrease the run-time, and will therefore affect the quality of the solution. Hence we can perform a tradeoff between CAD runtimes and resulting quality by using $C*(N_{blocks})^{1.33}$ moves, where

increasing C results in higher quality results at the expense of longer runtimes.

3.1.5. Xilinx Placer

This placement tool is part of the commercially available Xilinx Alliance Software Series 2.1i for Solaris. The placement is run at different effort levels (1-5), which indicate the amount of time the tool spends searching for a better quality solution. An effort level of 1 indicates that the tool terminates when it finds a low quality placement and an effort level of 5 indicates that the tool searches for a high quality placement.

3.2 Placement Results

For our initial comparison all the placement outputs were routed using the Xilinx router with effort level set to 5. Figure 1 shows the placement run-times vs. critical path delay graph. The results we obtained were normalized with the average values for each benchmark across the algorithms. The graph represents the geometric mean of these normalized values for all the benchmarks. If any of the placement results failed to route with the Xilinx router, they were placed again with the Xilinx placement tool with the effort level set to 5, and the runtimes of both Xilinx router and the Xilinx placement tool were added to the corresponding placement runtimes as a failure penalty.

Figure 1 demonstrates the tradeoff between the runtimes of algorithms and the quality of the results achieved. The height of each graph represents the critical path delay achieved, with a higher value representing a worse result. Horizontally we move from the fastest algorithms (at left) to the slowest (at right). Because of the structure of the graph, any point "dominates" all others that lie above and to the right of that point. This is because the given point gives equal or better results in equal or lower runtimes.
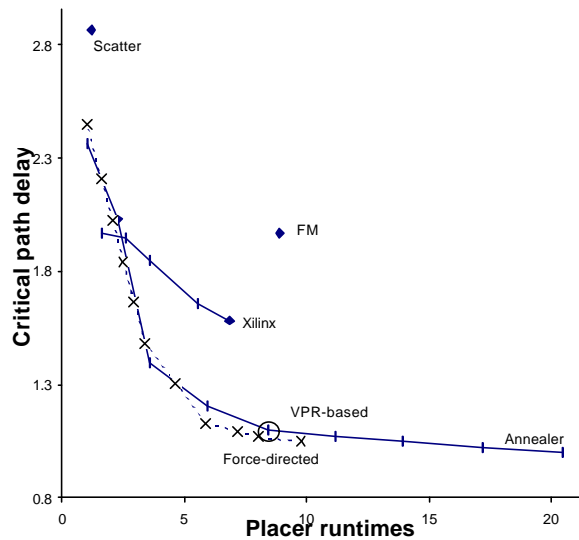

Figure 1. Critical path delays vs. runtimes for placement algorithms.

While the Scatter and Fiduccia-Mattheyses algorithms are depicted as single points the Xilinx placer as well as Annealing and Force-Directed algorithms are shown as curves. This is because the latter algorithms were run a number of times while varying the tuning parameter of each algorithm. For the Xilinx placer this tuning parameter was the effort level, which was varied from 1 to 5. For Annealing, this tuning parameter was the number of moves attempted at each temperature, ranging from $1*(N_{blocks})^{1.33}$ to $20*(N_{blocks})^{1.33}$. For the Force-directed algorithm, this tuning parameter was the terminating condition. We varied it from <20% cost decrease across iterations to <0.5% cost decrease across iterations. Also, we used an ultra-fast version of Force-directed algorithm that runs for only a single iteration.

There are several striking features of these graphs. First is the comparison between the Force-directed and Simulated Annealing algorithms. For relatively fast runtimes the Force-directed algorithm produces mappings with equivalent critical path delays to the Simulated Annealing algorithm. Although the Scatter algorithm run extremely fast for certain benchmarks, it fails to produce a routable design for certain large benchmarks. This adds a huge failure penalty to the runtimes and thereby brings down the overall performance of Scatter algorithm. Thus, at least for placement runtimes, achieving the best tradeoff between quality and runtimes may require different algorithms at different points in the spectrum.

A second observation is that several of the algorithms simply are not competitive when only placement performance is considered. The partitioning-based (FM) placer and the Xilinx placer provide significantly worse results than the other approaches.

From this data we can quantify the overall tradeoff between runtimes and quality for FPGA placement. For example, when we compare the fastest to the slowest competitive algorithms, we can achieve a speedup of 20x with a degradation of a factor of 2.3x in critical path delay. Also, compared to the VPR placer, if we allow a factor of 1.34x degradation in quality we can achieve 2.5x speedup in placement times, and a factor of 5.2x speedup if we allow 1.9x degradation in quality.

4. Routing

While the previous comparisons have considered only placement, the physical design process includes both placement and routing. In this section we present a similar algorithm development and comparison for FPGA routing.

All the routing algorithms we implemented represent the architecture of the XC4000 series FPGA as a directed resource graph $G = (N, E)$. A node $n \in N$ represents a routing resource such as a wire or terminal, and an edge $e \in E$ represents a switch or a feasible connection between two nodes. Each net consists of one Source node, and a set of Sink nodes. Routing a signal is essentially assigning routing resources such that all sinks are connected to the source.

We have 5 different routers, including the standard commercial router from Xilinx. Our aim is to compare each of these routers in terms of their run-time vs. quality

characteristic. As in the case of placement, some of the routers have tuning parameters with which they can be forced to spend more time searching for, and therefore potentially arrive at, a better quality solution. The following sub-sections detail the various routing algorithms implemented.

4.1 Routing Algorithms

4.1.1. Original Pathfinder

The original pathfinder algorithm was developed at the University of Washington [11], and has showed very high quality results. For our work we utilized an implementation of this algorithm that was developed at the University of California, Santa Cruz [4]. This is a negotiation-based router in which each net negotiates the use of shared resources with other nets until none of the resources are shared. Congestion costs are assigned to the shared resources and are increased with each iteration, thereby forcing some signals to explore alternate routes. The cost of using a node n is given by

$$c_n = (d_n + h_n) * (p_n + 1)$$

where d_n is the basic delay cost for using the node. The first order congestion term p_n, is the number of signals that currently share the node. The second order congestion term h_n grows monotonically with each iteration in which the node is shared. In order to minimize both congestion and delay, the actual cost function for using a resource when routing a net joining Source n_s to Sink t_{ij}, is defined as

$$C_n = A_{ij}d_n + (1 - A_{ij}) c_n$$

where c_n is the cost as defined earlier. The slack ratio A_{ij} is the ratio of the delay of the longest path containing the edge (n_s, t_{ij}) to the maximum delay over all paths. This slack ratio becomes 1 if the source-sink pair lies on the critical path, thereby reducing the cost to just the delay term. If the source-sink pair lies on a totally non-critical path, the congestion term will dominate, resulting in a route that avoids congestion at the expense of extra delay.

4.1.2. Modified Pathfinder

This version of the Pathfinder algorithm has two modifications over the Original Pathfinder, both intended to decrease the run-time of the algorithm.

The first modification is based on VPRs router [2]. It aids in routing multi-terminal nets more efficiently and has no tradeoff associated with it. The Original Pathfinder algorithm uses the maze router to route between a given Source and a Sink. For multi-terminal nets, this means that the wavefront generated while routing between the source and k^{th} sink will be discarded and a whole new wavefront will be generated to route the source to the $(k+1)^{th}$ sink. This requires considerable CPU time for high-fanout nets, since the partial routing used as the net source will be very large. Instead, in this implementation, we just update the wavefront around the newly found path until it reaches the same expansion level as the rest of the wavefront, and then proceed to find the next sink. Since the path from the existing wavefront to the newly found sink is fairly small, it will take little time to add this to the wavefront, and the next sink will be reached quicker than if the whole wavefront was to be generated again.

The second modification provides a decrease in the routing runtime at a small cost in quality. During each iteration, the Original Pathfinder algorithm rips up and reroutes all the nets so as to eliminate dependencies on the order of the nets. In our implementation, only those nets that are routed through congested resources are ripped up and rerouted. This may adversely affect the net delays, but considerably speeds up the algorithm.

One of the parameters of all versions of Pathfinder, the history cost for a node, indirectly influences the run-time of the algorithm. If we raise the per iteration history cost increase, the algorithm will more quickly resolve node sharing. While this may increase the corresponding net delays, because of the reduced number of iterations, the algorithm tends to run faster. Hence, by varying this history cost, we obtain several versions of the Modified Pathfinder algorithm that exhibit different run-time vs. quality characteristics.

We varied this history cost parameter in two different ways. One is by assigning fixed values to the history cost, which represents multiplying the standard pathfinder history cost by a scaling factor. Different history cost settings were considered across a wide range of workable assignments. The second approach was taken to determine if there is any dependence of the history cost of a node on its basic delay. Hence, in this approach, the history cost of a node equals its basic delay multiplied by a scaling factor. This scaling factor was again varied till the algorithm either fails to converge on a solution, or fails to route the given placement.

4.1.3. Xilinx Router

This routing tool is part of the commercially available Xilinx Alliance Software Series 2.1i for Solaris. The tool is run at different effort levels (1-5), which indicate the amount of time the tool spends searching for a better quality solution. An effort level of 1 indicates that the tool terminates when it finds a low quality routing and an effort level of 5 indicates that the tool searches for a high quality routing.

4.1.4. Hierarchical Router

This routing algorithm is partly based on the Timing-Driven Router developed in [12]. Starting with the entire FPGA, a cut line is chosen to divide the FPGA into 2 parts. Across the cut line there are routing sections that represent routing spaces on the chip. A routing section is a group of tracks in a channel with the same segment length. Each net crossing the cut line is assigned a cost similar to the pathfinder cost function with congestion considered at a segment level rather than at the track level. At each hierarchical level, the algorithm assigns routing sections to the nets crossing the cut line. After finishing routing at this hierarchical level, both parts separated by the cut line are independently routed by the same method recursively. When the parts become small enough, a simple greedy algorithm determines all the tracks in the routing sections that are assigned to each net.

4.1.5. Simple Router

This is a fast router based on the maze running algorithm [13]. Each net is routed once, with no rip-up-and-retry or

other technique for avoiding congestion. It simply seeks the shortest available route from source to destination(s), avoiding resources used by previous signals. If the maze runner fails to find any unshared paths from the source to the sink, the router declares the placement as unroutable and exits.

4.2 Routing Results

For all the routers, we used the placement obtained from the VPR-based Annealer. Figure 2 shows the routing run-times vs. critical path delay graph. Similar to placement, for each benchmark the results were normalized with average values across all algorithms and a geometric mean of these normalized values is indicated in the graph.

While the Original Pathfinder, Hierarchical Router and Simple Router appear as single points on the figures, Xilinx router and Modified Pathfinder are shown as curves. This is because the Xilinx router was run with different effort levels (1-5) with effort level 1 indicating a low quality routing and an effort level 5 indicating a high quality routing. The Modified Pathfinder was run a number of times while varying the history congestion cost from 0.7 to 5 as a fixed value, and from $d_n / 2$ to $2*d_n$ as a delay dependent value, and these two variations are shown as two different curves.

The graph features some interesting results. Firstly, the Original Pathfinder algorithm gives the best quality and the Simple Router runs the fastest. However, the simple router fails on some circuits, and thus may not be useable in all situations. In such cases, the placement was rerouted using the Xilinx router with an effort level of 5, and this run-time was added to the Simple routers runtime as a failure penalty.
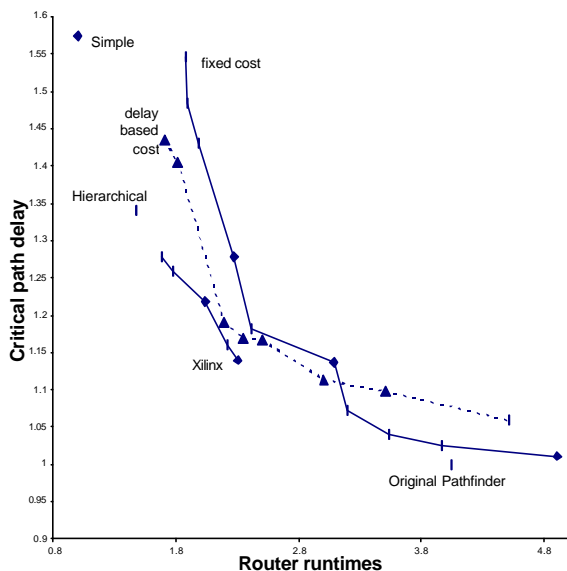

Figure 2. Critical path delay vs. runtimes for routing algorithms.

Secondly, the fact that the Modified Pathfinder curves never give a better quality than the Original Pathfinder algorithm suggests that there is a permanent quality loss associated with ripping up and re-routing only congested nets instead of all

the nets. At values of history cost greater than 5, the Modified Pathfinders failed to route the given placement, and for values less than 0.7, they failed to converge on a possible route.

A third observation is that, for lower runtimes, Modified Pathfinder algorithms with delay based history congestion costs outperformed those with fixed-value history costs. However, at longer runtimes, this situation is exactly reversed, with fixed value history cost based algorithms giving better qualities. One more important observation from the graph is that the Xilinx Router outperforms certain Pathfinders.

Finally, we can quantify the overall tradeoff between runtimes and quality for FPGA routing. For example, when we compare the fastest to the slowest routing algorithms, we can achieve a speedup of 6x with a degradation of a factor of 1.6x in critical path delay. Also, compared to Original pathfinder, if we allow 1.25x degradation in quality, we can achieve a factor of 2.5x speedup in routing times.

However, the runtimes of the routers are dependent on the quality of the placement that is input to them. For example, choosing a faster placement algorithm may make the routers job harder because of less efficient placement and hence increase the runtimes of the router. Quantifying this increase would help us decide on a specific combination of placement and routing algorithms for a particular total runtime constraint.
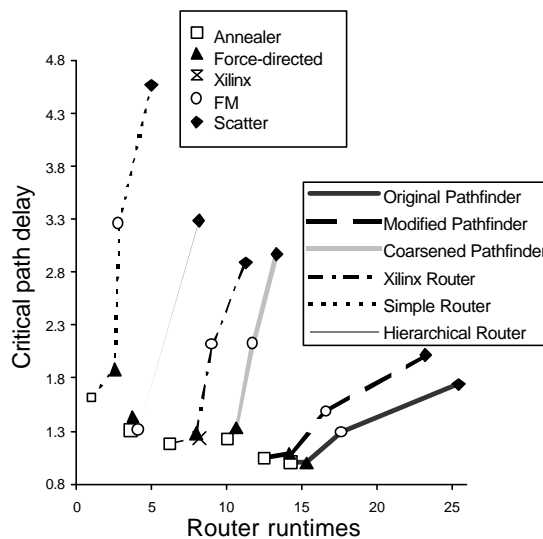

Figure 3. Router runtimes for different choices of Placement.

Figure 3 demonstrates how the routing run-times vary with different choices for placement. For each of the routers, the least runtime and best quality was obtained when Annealer was used as the placer and least quality and longest runtime was obtained when Scatter algorithm was used for placement. The only exception to this is the Hierarchical router, which gave comparable results with both Annealer and FM Placer. Also, the effect of the placement is much more drastic on the Simple Router compared to the Original pathfinder as can be seen from the increased slopes of the curves. This clearly

demonstrates that tradeoffs in placement will have correspondingly opposite tradeoffs in routing.

5.   Combined Place and Route

While the previous results demonstrate that trade-offs exist among different placement algorithms and routing algorithms, they do not address the critical question of deciding which set of algorithms should be chosen for a specified trade-off. In other words, given that the user is willing to trade-off some quality for an improved run-time, should a faster placement algorithm and a slower route algorithm be chosen, or a slower placement algorithm and a faster route algorithm? This choice not only reflects the balancing criteria that the overall runtime should be optimized for a given quality level, but also the fact that the choice of a faster, but less efficient, placement algorithm may increase the runtimes of the router, as it must accommodate a more difficult placement. In other words, tradeoffs made in placement or routing individually may not necessarily translate into corresponding tradeoffs overall.
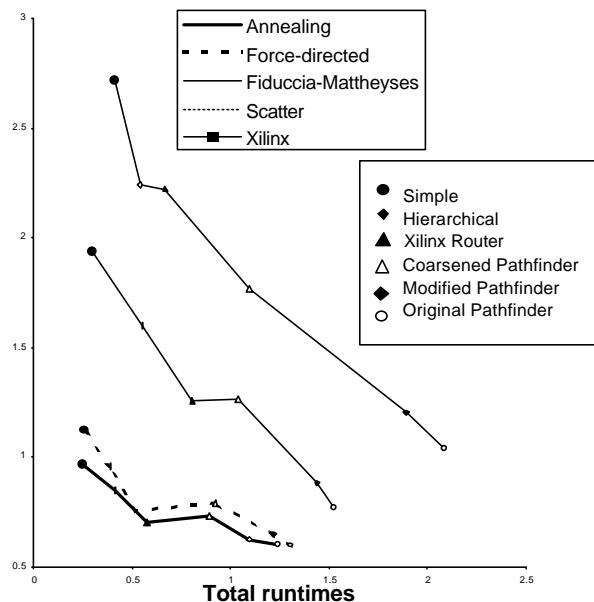

Figure 4. Critical path delay vs. Total runtimes for combinations of placement and routing algorithms.

We therefore compared all combinations of place and route algorithms mentioned (note that the Xilinx Placer was only run with the Xilinx router because we could not read the intermediate format). The results are depicted in Figure 4. In this figure, we chose a single representative of the tunable placement and routing algorithms. Hence, Annealer represents the VPR-based annealing algorithm with 10 * $(N_{blocks})^{1.33}$ number of moves at each temperature, Force-directed placement represents the Force-directed algorithm with <5% cost decrease across iterations as the terminating condition, Modified Pathfinder router represents the Modified Pathfinder algorithm with history cost $h_n = 1$ which is the same value as for the Original Pathfinder, and Coarsened Pathfinder router represents the Modified

Pathfinder algorithm with history cost $h_n = d_n / 1.35$. The Xilinx algorithms were run at an effort level of 5.

The graph demonstrates the tradeoff between the overall runtimes and the quality of the results achieved by the different combinations of placement and routing algorithms. As in the earlier graphs, the height of each graph represents the quality of the result decreasing as we move up. Horizontally, the runtimes increase as we move to the right. Because of the structure of the graph, any point "dominates" all others that lie above and to the right of that point. This is because the given point gives equal or better results in equal or lower runtimes.

One noticeable feature of this graph is that, in combination with any router, the VPR-based Annealing algorithm always dominates the other placement algorithms. However, the Force-directed algorithm performs almost as well as Annealer. This illustrates an interesting point that any trade-off to be made in run-time vs. quality is better made in route algorithms, while the Annealer algorithm should be used as the placer. This is primarily due to the fact that for most of our algorithm combinations the routing time dominates the placement time. Also, consistent with the results for routing algorithms, for each placement the Original Pathfinder algorithm always gives the best quality mapping and the Simple Router always gives the fastest solution (when it actually succeeds in routing).

One more important observation is that the combination of Xilinx Placer and Xilinx Router performs much better than we observed in the earlier results for individual placement or routing.

Finally, we can quantify the overall tradeoff between total runtimes and quality for FPGA CAD tools. For example, when we compare the fastest to the slowest algorithms, we can achieve a speedup of 8x with a quality loss of just 1.1x in critical path delays. However, the combinations vary very widely, from speed-ups up to 8x, and up to 4.5x quality degradation. Also, compared to the Xilinx CAD tool, if we allow 1.15x degradation in quality, we can achieve a 3x speedup in total runtime. Compared against the VPR-based Annealer, and Original Pathfinder (two of the most successful research CAD tools), we can achieve a factor of 5x speedup if we allow a factor of 2.5x degradation in quality, and a factor of 2.2x speedup if we allow 1.8x degradation in quality.

6.   Effect of FPGA sizes on the algorithms

The results obtained so far assume the user to be resource constrained, in that all the benchmarks were fitted into the smallest real FPGA possible. However, if the user were to have no such constraint, the algorithms will have more freedom in dealing with congestion and hence might result in a better quality of solutions. In order to investigate this dependence of algorithms on the amount of FPGA resources utilized, we ran a set of place and route algorithms on three benchmarks for five different FPGA sizes. This set of algorithms was chosen from the dominant set from the previous section. Hence, they contain all router combinations with Annealer as the placer, and the combination of Force-

directed placement and Original Pathfinder. While three of these sizes correspond to real FPGAs, two of the sizes correspond to hypothetical FPGAs whose sizes are one step immediately above and below the originally targeted FPGA. Figure 5 illustrates the results obtained in graph form. The vertical axis of the graph represents the critical path delays obtained, and the horizontal axis represents the runtimes for placement and routing. The results were normalized with average values across the algorithms and a geometric mean of these normalized values are represented in the graph. Each point in the graph denotes the mean result of all the algorithms used for a particular FPGA size. The percentages next to these points denote the amount of resources occupied in the FPGA.
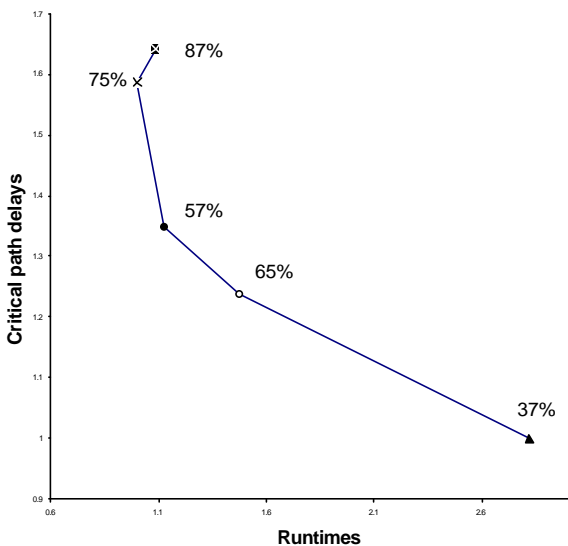

Figure 5. Effect of Resource Usage on the performance of algorithms.

We did not observe any appreciable deviance in the behavior of individual algorithms as opposed to the mean behavior, which is what is represented in the graph. While the runtimes of the algorithms keep increasing with size, which is to be expected with the increase in available resources, the quality of the solutions keep increasing too. However, for the lowest size the quality as well as the runtimes are adversely affected due to an increase in congestion.

Unfortunately, these results indicate that one cannot simply throw more resources at the problem to increase the performance of the CAD tools – adding more space in the FPGA slows the runtimes, even though (or perhaps precisely because) the placement and routing are less constrained. Thus, performance improvements come from algorithm changes, not the easing of FPGA resource constraints.

7. Conclusions

The results we obtained demonstrate that the CAD algorithms are dispersed widely in the quality vs. runtimes tradeoff spectrum from a speedup of 8x to a 4.5x quality degradation. However, compared to the slowest combination of algorithms (Scatter and Original Pathfinder), the fastest combination

(Annealer and Simple Router) produces only 1.2x degradation in the quality of the solution with 8x speedup in total runtime.

Apart from implementing a variety of placement and routing algorithms, we developed tradeoff-oriented algorithms for both placement and routing. These algorithms can be tuned to obtain different tradeoffs by varying a single parameter. This tuning helps in broadening the applicability of individual algorithms. For example, Force-directed placer can be tuned to run 9x faster with only a 1.3x quality degradation.

Achieving the best results requires varying different algorithms as well as varying the tuning parameters of these algorithms. Also, for the best results, both place and route times need to be considered since a faster (but lower quality) placement can slow down the router. In fact, using Annealing algorithm for placement in combination with other routers gives the best quality solutions for a given run-time. Thus, the advantages of a faster, but lower-quality, placement must be balanced against the runtime and quality degradations this will cause in the router.

In order to take advantage of these opportunities, it is critical to develop methodologies for automatically choosing the best combination of placer and router, as well as the correct tuning parameter setting, to get the desired result in the best time. In our future work we will seek to quantify the tradeoffs involved, and automatically seek the best combination of CAD algorithms on a problem-by-problem basis. Most importantly, we will seek to meet requirements on the critical path delay set by the user or the available resources, while performing placement and routing as quickly as possible.

8. Acknowledgements

9. References

[1]   Peixin Zhong, Margaret Martonosi, Pranav Ashar, and Sharad Malik. "Accelerating Boolean Satisfiability with Configurable Hardware". IEEE Symposium on FPGAs for Custom Computing Machines, April. 1998, pp. 186—195.

[2]   Vaughn Betz and Jonathan Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research", International Workshop on Field Programmable Logic and Applications, 1997, pp.213-222.

[3]   Yaska Sankar and Jonathan Rose, "Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs", International ACM/SIGDA Symposium on Field-Programmable Gate Arrays, Feb. 1999, pp.157-166.

[4]   Pak K. Chan and Martine D.F.Schlag, "New Parallelization and Convergence Results for NC: A Negotiation-Based FPGA Router", ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Feb. 2000.

[5]   Xilinx, "The Programmable Logic Data Book", 1996, pp.4.5-4.106.

[6]   S.Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0", Tech. Report, Microelectronics center of North Carolina, 1991.

[7]   C.M.Fiduccia and R.M.Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", Design Automation Conference, May 1984, pp.175-181.

[8]   A.E.Dunlop and B.W.Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits", IEEE Transactions on Computer-Aided Design, Vol. 4, No. 1, 1985, pp.92-98.

[9]   K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques", ACM Computing Surveys, vol.23, No.2, June 1991, pp.143-220.

[10]   J. Lam and J. Delosme, "Performance of a New Annealing Schedule", Design Automation Conference, 1988, pp.306 – 311.

[11]   L.McMurchie and C.Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs", Proceedings of 3rd International ACM/SIGDA Symposium on Field-Programmable Gate Arrays, Feb. 1995, pp.111-117.

[12]   Timing-Driven Routing for Symmetrical Array-Based FPGAs, ACM Transactions on Design Automation of Electronic Systems, Vol. 5, No. 3, July 2000, pp. 433-450.

[13]   K.W. Lee and C. Sechen, "A New Global Router for Row-Based Layout", International Conference on Computer-Aided Design, IEEE, 1988, pp.180-183.