Persistent CAD for in-the-field Power Optimization

Peter Jamieson
Dept. of Electrical and Computer Engineering
Miami University
Address: Oxford, OH, 45056
Email: jamiespa@muohio.edu

***Abstract*—A major focus within the Integrated Chip (IC) industry is reducing power consumption of devices. In this paper, we explore the idea of persistent CAD algorithms that constantly improve the power consumption of consumer devices that use FPGAs. The idea is that the field-programmability of an FPGA allows updates to be deployed in the field, and as CAD algorithms find optimizations for a design, these optimizations can be deployed into the field. To explore this idea, we have created a persistent placement algorithm for FPGAs using a genetic algorithm. We describe the design of this genetic algorithm, and then use it in an experiment to show the impact on power consumption. Our results for one of the larger MCNC benchmarks, clma, shows that over a 60 minute period better placement solutions are found, but the rate at which these solutions are found decreases quickly.**

I. Introduction

Power consumption of electronic devices impacts our environment as we search for more energy resources and impacts the battery longevity of consumer devices. Traditional Computer Aided Design (CAD) for Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) focuses on optimizing speed, area consumption, and power consumption metrics as best as possible in a single tool pass from design to implementation. During this pass a high-level design (possibly an Hardware Description Languages (HDLs) design) is mapped through a set of CAD stages that converts this design into a technological realizable format that can be manufactured or programmed onto a target technology.

These CAD stages, such as placement and routing, use algorithms that attempt to optimize the design for the metrics described earlier, and these problems are NP-complete problems [1]. Various heuristics have been developed to find reasonable solutions to these problems in a fixed amount of time, where reasonable is defined based on designer's perspective. For example, ASIC designers might be able to work with a 1 week run-time for the CAD flow as opposed to FPGA designers expect a 1 hour to a maximum of one day for their CAD process to complete. The reality is that the solutions are not optimal as there is only so much time to explore the optimization problems search space.

An interesting characteristic about an FPGA is that it is possible to reprogram the device in the field, and therefore, it is conceivable that the CAD flow used to map a design to an FPGA could potentially continuously run and find better solutions in the search space. As more optimal solutions are found, these solutions could be updated to the FPGAs resulting in improved device performance. The key benefit of the persistent CAD algorithm is that over time the power consumption of consumer devices can decrease. For example, imagine your cell-phone that contains an FPGA, actually, has increased battery lifetime the longer you own it.

The focus of this work is to study how a genetic algorithm for persistent placement impacts the power consumption of a design mapped to an FPGA. We investigate how to build a persistent placement algorithm within the VPR 5.0 [2] framework that includes power estimation [3]. Using this algorithm we run the algorithm on a benchmark observing how power consumption is improved. Our results show that persistent CAD for placement does improve the power consumption of an FPGA, but the rate of improvements slows as expected.

The remainder of this paper is organized as follows. Section II describes placement algorithms for FPGAs including genetic algorithms. Section III describes the design of our genetic algorithm for persistent placement. Section IV describes our experimental setup and shows results of our experiment for one MCNC benchmark. Finally, Section V concludes this work.

II. Background

A series of CAD flow stages convert an HDL design to a programmable bit-stream that can be uploaded to the FPGA to implement the design. The focus of this work is on the placement stage, which we review below.

A. Details of FPGA Placement

The goal of the placement problem for FPGAs is to place the digital logic, representing the digital design, onto the array of FPGA clusters such that the critical path (the longest path from either a primary input to a primary output, a primary input to flip-flop, flip-flop to flip-flop, or flip-flop to primary output), the power consumption, and the wire-length are minimized. This problem has been shown to be NP-complete to solve optimally, and a number of popular heuristic based algorithms have been used for FPGA placement including simulated annealing ([1], [4]), which is the algorithm used in VPR 5.0, min-cut ([5]), analytic ([6], [7]), which includes force-directed placers.

The simulated annealing (SA) algorithm, when used in the FPGA domain for placement, attempts to minimize the above circuit metrics based on the process of cooling metals. A cooling schedule controls the weighting of a probability function. This function determines if randomly selected swaps between logic mapped on an FPGA are accepted or not. Each random swap of logic will either improve or degrade the critical path, and initially, all swaps are accepted regardless if they improve the critical path or not. As the temperature cools, only swaps that improve the optimization metrics are accepted. In this way, the early phases of the cooling schedule is used to allow hill climbing that will avoid early local minimums in this optimization problem.

The two most relevant aspects of the annealer as a placement algorithm for FPGAs are the scheduling of the cooling and the cost function that evaluates a current solution. The scheduling of the annealer determines if a random swap of logic is accepted and determines the maximum Manhattan distance of the swap. As time progresses, swaps that do not improve the cost function are not accepted, and the Manhattan distance between the potential swaps is reduced. From this works perspective, the distance of a random swap is relevant and is based on the term R_{limit}. For example, given a 5 by 5 FPGA, R_{limit} can have a maximum value of 5 meaning that digital logic located at the x coordinate 0 and y coordinate 0 could be swapped with another piece of digital logic located at x coordinate 4 and y coordinate 4. As R_{limit} is reduced then the distance of swaps is smaller representing the stabilizing of the placement algorithm.

The second aspect of the annealer is the cost function used to steer the optimization algorithm as various random swaps are attempted. VPR 5.0 cost function consists of two components defined in [4]. The first part of the cost function is the sum of the bounding box dimensions of all nets. Given N nets, $bb_x(i)$ and $bb_y(i)$ are the x and y dimensions of the bounding box for each $net(i)$, and $q(i)$ is a scaling factor for better wire-length estimates. The first component of the cost function is defined as:

$$WiringCost = \sum_{i=1}^{N} q(i) \cdot [bb_x(i) + bb_y(i)] \tag{1}$$

Component two is used to evaluate timing costs of a placement where,

$$TimingCost = \sum_{\forall i,j \in circuit} Delay(i,j) \cdot Criticality(i,j)^(CE) \tag{2}$$

and CE is a constant, Delay(i,j) is the delay of the connection from source i to sink j, and Criticality(i,j) is a measure of how close the given i, j path is close to the critical path.

Lamoureux *et. al.* [8] extended this cost function to be power aware. They added a new term:

$$PowerCost = \sum_{i=1}^{N} q(i) \cdot [bb_x(i) + bb_y(i)] \cdot Activity(i) \tag{3}$$

where $Activity(i)$ is the switching activity on a particular net, and by reducing this component, the power consumed over long and power hungry programmable routing wires is reduced. The placement cost function with all components is:

$$\begin{aligned} Cost = \lambda \cdot \frac{TimingCost}{PreviousTimingCost} + \\ (1-\lambda) \cdot \Bigg[(1-\gamma) \cdot \frac{WiringCost}{PreviousWiringCost} + \\ \gamma \cdot \frac{PowerCost}{PreviousPowerCost}\Bigg] \end{aligned} \tag{4}$$

where the γ factor is used to control how strong or weak the power optimization is and λ parameter is used to weight the importance of each of criticality and wire length. This cost function is also the one used in our genetic algorithm, which we will discuss later in this work.

B. Genetic algorithms and placement

Genetic algorithms (GAs) use the "survival of the fittest" idea to find solutions for an optimization problem. The basic idea is that a population of different solutions is created to solve a problem, and the best individuals/solutions are duplicated and modified, through operations such as cross-breeding and mutation, which create the next new population of solutions.

In addition to the three types of placement algorithms described earlier, GAs have been used for FPGA placement. Venkatraman *et. al.* [9] implemented one of the first GA placers in VPR 4.3 (the predecessor to VPR 5.0). In their work, each piece of the digital logic is considered a gene, and the 2-D location of each of the items is combined to create an individuals genome. A population of these individuals is created and they are evaluated based on a fitness function similar to the cost function described in the previous section. Within the population, the fittest individuals are kept and mutated to create the next generation. Their results show that this algorithm improves the critical path compared to VPR's SA algorithm for ten benchmarks.

More recently, Meng *et. al.* [10] have created an algorithm that combines both GA and SA algorithms for FPGA placement. Their approach claims that the GA aspect of the algorithm is used to escape local minimums (as an alternative to hill climbing) and the SA is used to quickly converge on good solutions. Their results show similar costs compared to VPR 4.3's SA approach with similar run-times.

III. Genetic Algorithm for Persistent Placement

For our persistent placement algorithm, we chose to implement a GA since the idea of continuously creating new populations is much more applicable to persistent CAD than modifying SA timing schedule. In this section, we describe how we implement our GA for persistent placement.

We built our GA placer in VPR 5.0 that supports power estimation [3]. This allows us to use the cost function as shown in equation (4) to evaluate the fitness of our population.

Similar to other GA implementations for FPGA placement, we have also chosen to create a genome based on the x and y coordinates of each logic piece of the design. In our approach, we create new generations by mutating the fittest individuals

TABLE I
CONFIGURABLE PARAMETERS FOR THE GA

Parameter	Description of parameter
ω	The percentage of the fittest individuals (parents)
α	The percentage of children
β	The percentage of randomly created individuals
σ	The number of individuals in the population
τ	The percentage of genes to randomly mutate
λ	A parameter to weight timing optimization importance
γ	A parameter to weight power optimization importance

in the current population. Each generations population consists of $\omega\%$ of the fittest individuals from the previous generation, which we call the parents, $\alpha\%$ of the population is mutated individuals from the parents, and $\beta\%$ of the population are randomly generated new individuals, where $\omega + \alpha + \beta = 100\%$. The population size of each generation is defined by parameter σ and the number of mutations for the mutated children is defined by $\tau\%$ where the total number of mutations is a percentage of the number of genes in a genome (this is the same as the number of logic elements in the netlist). Table I shows all the parameters associated with our GA including the parameters in the cost function.

A GA follows a set of steps through the evolutionary process, and in our algorithm, we use the term R_{limit} to control the distance of random swaps in our mutation operation. Similar to the SA scheduling algorithm, R_{limit} starts as the maximum size of one dimension of the FPGA and decreases to 1, but in our persistent algorithm, once R_{limit} = 1 for the next generation we set R_{limit} to the maximum size of the FPGA. In this way, the GA algorithm tries both fine grain changes and course grain changes over time in a controlled fashion.

IV. EXPERIMENTAL RESULTS FOR PERSISTENT PLACEMENT FOR FPGAS

To evaluate the idea of a persistent placement algorithm and the quality of our algorithm we execute the GA placement for different values of γ and λ that control the cost functions focus on timing cost, wiring cost, and power consumption cost. The persistent GA placement algorithm, described in the previous section, is implemented within VPR 5.0 software, which includes a power estimation framework. We run this algorithm for one MCNC benchmark and sweep the γ and λ parameters. In this section, we will describe more details of the experimental setup, and then report our results.

A. Experimental Setup

For our experimental setup, there are a number of details that we must describe, and these include:

1) The parameters γ and λ which control the weighting of the placement cost function
2) The architectural parameters describing the FPGA
3) The parameters in Table I describing our Genetic Algorithm
4) The system and conditions on which the algorithm is executed

The reality is that there are a huge number of parameters that could be explored in our experiments. We have chosen to only vary γ and λ parameters so that we can explore the idea of persistent CAD and not so much the quality of our algorithm and the impact of the FPGA architecture. In future work, we hope to explore the quality of a persistent CAD algorithm in more detail.

TABLE II
THE FPGA ARCHITECTURAL PARAMETERS

Parameter	N	K	F_{cin}	F_{cout}	F_s
Value	10	5	0.18	0.1	3

In terms of architectural parameters, we invite the reader to read [4] that describes the FPGA architectural meaning of each parameter. For the sake of space and since there is not a huge need for the reader to understand these parameters relative to this work, we simply present our architectural parameters in Table II.

TABLE III
THE GA ALGORITHMIC PARAMETERS

Parameter	ω	α	β	σ	τ
Value	20%	50%	30%	100	10%

Table III shows the values selected for the GA. These parameters have not been tuned by any special means other than basic intuition and a few trial and error experiments. Again, we are not so much exploring the quality of a persistent algorithm in this work, and instead, we are focusing on the basic concept.

Our experiments are run for one of the MCNC benchmarks, clma. This benchmark is optimized via our GA persistent placement algorithm for different values of the γ and λ parameters for weighting the cost function. The values of γ and λ parameters are 0.0, 0.2, 0.5, 0.8 and 1.0. For our experiments, we run all combinations of these. For each unique parameter pairing of γ and λ, we run a benchmark for 60 minutes, and at each minute of execution, the current best placement (or fittest individual in the population) is output in a placement file. This placement file is post-routed and evaluated in terms of power and speed.

These algorithms are run on a Intel Core Duo E8400 CPU running at 3.00 GHz with 2GB of RAM in Cygwin for Windows XP. The E8400 is a 65 Watt machine that based on Sysmark Benchmarks [11] consumes on average 100.5 Watt-hours [12].

B. General Observations of Power and Speed Improvements over 60 minutes

Based on the experimental setup, every minute we gather the best placements for clma over a 60 minutes for different γ and λ parameters. The hypothesis is that we should see a power consumption improvement over time (depending on the cost function parameters) as the algorithm has more time to

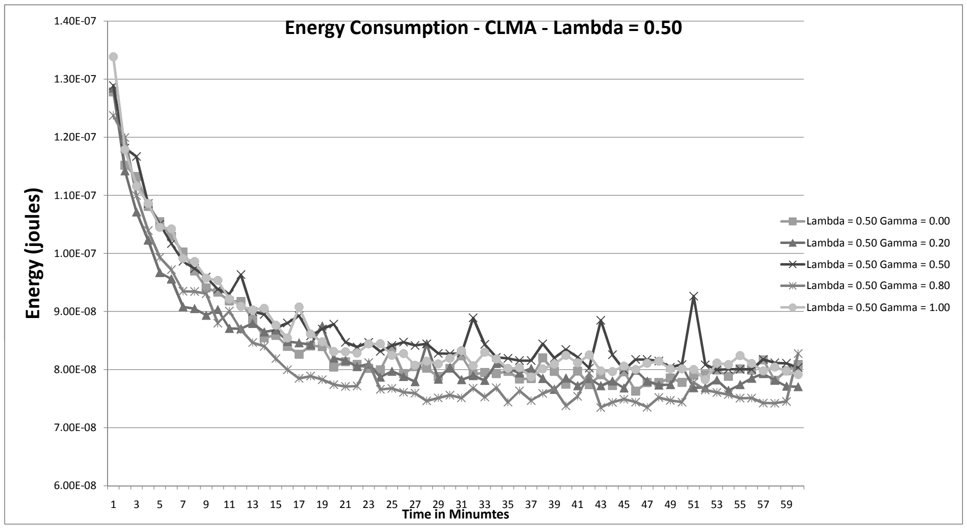

Fig. 1. Energy consumption results over 60 minutes for Lambda = 0.50 and Gamma changing

move around in the search space. Our results showed that the best value for λ is 0.5 and γ is 0.80, which is the same as those found in [8] for SA. For this reason, we only present results based on these parameter values.

The power results over time, as shown in Figure 1, shows that we do see incremental improvements in the overall energy consumption in a decay like function. We can see that over time the energy results are improving. Initially, for the first 10 minutes we see a rapid improvement of the power results, with some noise, and then from the 10th minute on we see a much slower energy improvement. We also see the similar improvement in speed, but do not show these results due to space limitations.

The reason for incremental improvement that tappers off is the nature of our algorithm implementation where we are, likely, stuck in a local minimum due to a hereditary path. Even though our GA algorithm introduces a percentage of new individuals (based on the β factor) as time progresses the fittest individuals are from the same lineage and are exploring a local minimum that is dominant with respect to the new random individuals. In future persistent CAD algorithms, more thought needs to be put into exploring more possible candidates, and different lineages and crossbreeding operators need to be considered.

V. Conclusion

In this work, we introduced the idea of persistent CAD optimizations and showed how in the FPGA domain this idea might be used to reduce power consumption of devices deployed in the field (though we have not discussed the engineering details of this in the filed updating). To experiment with this idea, we created a GA for placement that persistently looks for power consumption and speed optimizations. We built this algorithm within the VPR 5.0 framework and described the algorithmic parameters that control our implemented algorithm.

To study our algorithm that persistently seeks to improve a design's power consumption, we ran a GA over a 60 minute period (under a number of conditions) to observe how the sustained time in the search space improves the power consumption and speed of one MCNC benchmark. Our results show that, initially, there are significant gains that over time decreases. We believe that this is partially due to our algorithm getting stuck in a local minimum, with a small probability of escaping, and future work might focus on a better approach to dealing with this problem.

References

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[2] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and Architecture xploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling," in *ACM/SIGDA International Symposium on FPGAs*, Feb 2009.

[3] P. Jamieson, W. Luk, S. J. Wilton, and G. A. Constantinides, "An energy and power consumption analysis of fpga routing architectures," in *International Conference on Field-Programmable Technology*, 2009, pp. 324–327.

[4] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[5] 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, pp. 92–98, Jan. 1985.

[6] B. M. Riess and G. G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement," in *IEEE International Symposium on Circuits*, 1995, pp. 377–380.

[7] K. Vorwerk, A. Kennings, and A. Vannelli, "Engineering details of a stable force-directed placer," in *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, 2004, pp. 573–580.

[8] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware fpga cad algorithms," in *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, 2003, p. 701.

[9] R. Venkatraman and L. M. Patnaik, "An evolutionary approach to timing driven fpga placement," in *GLSVLSI '00: Proceedings of the 10th Great Lakes symposium on VLSI*, 2000, pp. 81–85.

[10] Y. Meng, A. E. A. Almaini, and W. Pengjun, "Fpga placement optimization by two-step unified genetic algorithm and simulated annealing algorithm," *Journal of Electronics (China)*, vol. 23, no. 4, pp. 632–636, 2007.

[11] Bapco, "http://www.bapco.com/products/sysmark2007preview/," 2009.

[12] tom's hardware, "http://www.tomshardware.com/reviews/wolfdale-steroids,1777-16.html," 2010.