Research Proposal Accompanying Application for CSC Grant

Hardware Acceleration for FPGA Placement and Routing

Applicant: Yun ZHOU

zhouyun_zy@foxmail.com

SouthWest Jiaotong University, China
School of Information Science and Technology

Advisor: ir. Elias Vansteenkiste
Elias.Vansteenkiste@ugent.be

Promotor: prof. dr. ir. Dirk Stroobandt
Dirk.Stroobandt@ugent.be

Ghent University, Belgium
Department of Electronics and Information Systems
Computer Systems Lab
Hardware and Embedded Systems Group

Contents

Personal profiles

Name of Applicant: Yun ZHOU *Gender:* Female *Nationality:* Chinese
Date of Birth: October 2, 1992 *Phone Number:* +86 18782186684
Degree: Bachelor's Degree in Electronic and Information Engineering
Institution of Applicant: School of Information Science and Technology, SouthWest Jiaotong University, China
Institution to Which the Applicant is Applying:
Department of Electronics and Information Systems, Ghent University, Belgium

Signature of the Promotor:

Date:

1 Problem Definition

A Field Programmable Gate Array (FPGA) is an integrated circuit made up of a grid of programmable functional blocks embedded in a programmable interconnection network. An FPGA can be reconfigured at boot time by writing a different bitstream to the configuration memory, which makes it remarkably flexible to implement any arbitrary digital circuit. Mainly due to their flexibility that leads to a substantial reduction of the economic risk of developing hardware accelerators, FPGAs are becoming ubiquitous components in digital systems and datacenters [**?**, **?**]. The flexibility also explains the rise in the popularity of the FPGA [**?**].

Unfortunately the flexibility of the FPGA comes at a price. Designers typically needs to compile his application numerous times to test if his/her design meets the constraints of the application. Each time the design has to be compiled to an FPGA configuration and this is a complex and time consuming process, relying heavily on the FPGA computer-aided design (CAD) tools developed to compile a bitstream starting from a high level description. However, the power of the FPGA CAD tools lags behind when FPGA devices and application sizes still keep increasing, following Moore's law [**?**]. The traditional CAD heuristics scale very badly in terms of application and device size, which translates to waiting times of hours up until days for compilation to be finished [**?**]. Reducing the compilation time to improve the efficiency of CAD tools has become an active research field.

The compilation/translation of the FPGA CAD tool flow is typically divided into five steps: synthesis, technology mapping, placement, routing and configuration, dealt with by separate tools. Runtimes of the different steps in the compilation of the toolflow can be indicated by the pie chart in Figure 1. These were obtained for the popular academic tool flow Verilog-To-Routing [**?**] and for the commercial tool flow from Xilinx, Vivado [**?**], using the framework in [**?**]. As also stated by other important publications, such as [**?**], the most time-consuming steps are the placement and routing (PAR) steps, which have to be performed to obtain the needed accurate timing information to assess if the design meets the constraints. Therefore reducing PAR time is of great importance to reduce the overall runtime of the CAD flow.

One of the most promising approaches to keep the runtime of PAR in FPGA CAD tool flows reasonable is to develop multi-threaded versions of the heuristic algorithms [**?**, **?**, **?**, **?**], but unfortunately scalability issues still exist and large designs still need hours to be compiled. To further speed up compilation, a straightforward path is to use hardware acceleration, but unfortunately the current multi-threaded algorithms for placement and routing are hard to adapt for the wide scale parallelisation used in hardware accelerators such as Graphics Processing Units (GPUs) and FPGAs, which support massively parallel computing.

Summary: *FPGA design compilation takes too much time to allow efficient design turnaround times. Hardware acceleration of the compilation process is needed but the current placement and routing algorithms (see section 3 for details)are not very well suited for this.*

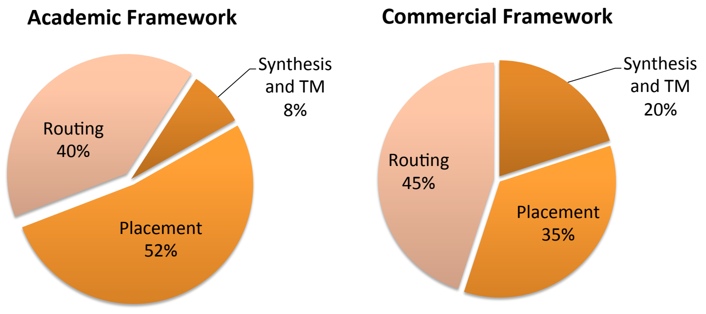

Figure 1: Breakdown per tool of the compilation runtime.

2 Research Objectives and Paths

Research Objectives *In this Ph.D. research proposal, the overall high level goal is to significantly reduce the FPGA design compilation time by efficiently parallelizing placement and routing algorithms onto a GPU/FPGA, in order to bring benefit to the developemnt of FPGA CAD tools and the design of FPGA based system.*

What we want to investigate in the project is how much placement and routing can be accelerated by writing a specific GPU-kernel based on a massively parallelizable placement algorithm, with regard to the existing single-thread ones or the improved parallel ones that are performed on CPUs.

Research Paths The first step is to have a modified version of the placement algorithm LIQUID that can be implemented on GPU by parallelizing and improving it. LIQUID is a placement algorithm that is being developed at Ghent University [**?**]. The second is to implement the adapted version of LIQUID with Compute Unified Device Architecture (CUDA) [**?**] and demonstrate and measure its performance.

Next comes the development of new routing algorithms that are much more susceptible to parallelization and much more efficiently accelerated by hardware than the current implementations. We will investigate the introduction of new routing algorithms that can be massively parallelized for being accelerated by a GPU implementation. Another option is to use an FPGA to accelerate the routing algorithm. In both cases, significantly novel routing techniques will have to be developed to enable a significant efficiency increase of the routing algorithm implementation.

3 Existing Approaches

3.1 The FPGA Placement Problem

An FPGA placement algorithm takes two inputs: the mapped input circuit and a description of the target FPGA architecture. The algorithm searches a legal placement for the functional blocks of the input circuit so that circuit wiring is optimised. In a legal placement every functional block is associated to (placed on) one of the physical blocks (without overlap) that is capable of implementing the functional block.

The main optimisation goal is to minimise the total wire length required to route the wires in the given placement. Placers that are only based on this goal are called wire-length-driven placers. More complex tools such as routability-driven [?] and timing-driven placers [?] trade some of the wire-length for a more balanced wiring density across the FPGA or a higher maximum clock frequency of the circuit, respectively.

Finding a high quality placement is very important, because poor quality placements generally cannot be routed or lead to low operation frequencies and high power consumption. Worse, the placement problem is computationally hard, so there are no known algorithms that can find an optimal solution in a reasonable time. The two popular heuristical approaches are the simulating annealing based algorithm [?] and the analytical placement techniques [?].

3.1.1 Simulated Annealing

Simulated annealing [?] is inspired on annealing of metals, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. A detailed description of the algorithm can be found in [?].

The initial temperature, the rate at which the temperature is decreased, the number of moves that are attempted at each temperature, the way in which potential moves are selected and the exit criterion of the annealer are called the annealing schedule. A good annealing schedule is crucial for finding a good solution in a reasonable amount of time.

Problems with using Simulated Annealing Typically the number of moves performed per temperature step is not linear with the number of blocks, which leads to long runtimes. Although it seems straightforward to adapt simulated annealing for multiple threads, the quality of the solution and the speedup gain per thread deteriorate fast as the number of threads increase [?]. This is mainly caused by collisions and synchronization issues. In multi-threaded solutions each thread swaps blocks independently, instead of swapping blocks sequentially.

3.1.2 Analytical Placement

In an analytical placement algorithm, the placement problem is represented as a linear system of equations that is built based on both the wire length of the connections and the distance to the legalized position, called anchor positions. These anchors are refreshed each time a solution is legalized. Multiple iterations of solve-legalize are performed, while increasing the weight of the anchor connections. The algorithm typically stops if the decrease in cost of the legalized solution starts to slow down and converge towards the cost of the solved solution.

Problems with using Analytical Placement Each iteration the linear system is solved, it takes time to build it. A matrix of size NxN is to be constructed, with N being the number of connections. Although the matrix is sparse and can be stored efficiently with compressed row format, it requires extra memory, making analytical placement less scalable in terms of memory. Both solving the linear system and the legalisation method are hard to parallelise and require synchronisation.

3.2 The FPGA Routing Problem

Once the placement algorithm has placed each of the logic blocks of the input circuit on a physical block of the FPGA architecture, the router needs to determine which of the switches in the routing architecture need to be closed and which need to be opened in order to connect the physical blocks in accordance to the way their associated logic blocks are connected in the input circuit. The goal is not only to find a legal routing solution but also to try to maximise the circuit clock frequency by allowing critical paths to use shorter and faster routing resources.

Most of the popular academic and commercial FPGA routers use a Pathfinder based algorithm[?, ?].

3.2.1 PATHFINDER: A Negotiated Congestion Router

Routing-resource Graph The routing-resource graph (RRG) used as a model for the routing architecture of an FPGA in PATHFINDER [?]. It is a directed graph $C = (N, E)$, where the nodes N represent the routing resources: wire segments, input pins, output pins, sinks and sources.

When the routing architecture of the FPGA is represented as a RRG, the routing algorithm will be reduced to finding a subgraph of the RRG, called a routing tree, for each of the nets. The algorithm stops when the routing trees of the nets are disjoint, which means no shared resources exist anymore.[?] This is achieved by gradually increasing the cost of sharing resources between nets. During the first iteration, nets can share resources at no extra cost and thus, each net is routed with a minimum number of wires. The cost of a routing resource does not only depend on the current sharing cost but also on the sharing history of the resource. Resources that were shared heavily in past routing iterations become more expensive. In this way a congestion map is built, which enables nets to avoid routing through heavily congested resources, if possible.

Routing a Net The task of the net router is to find a minimum cost routing tree for a given net in the resource graph. The routing tree should contain the source and the sinks of the net and a path from the source to each of the sinks. The cost of the resources is determined by the negotiated congestion mechanism.

The search space of all possible routing trees for a net is huge. Therefore a heuristic was developed that finds a low cost routing tree for a given net in a reasonable amount of time. Typically a variant of a maze router is used [?]. The maze router loops over all the sinks of the net and extends the already found routing tree with the shortest path from this routing tree to the sink under consideration. The shortest path is found using Dijkstra's algorithm [?].

Although the PATHFINDER algorithm is very flexible because the RRG can be constructed for any useful routing architecture, it is not suited for wide-scale parallelism.

3.2.2 State-of-the-art advances in routing techniques

In [?] the authors were able to further refine the negotiated congestion mechanism to the level of connections. Nets are partially ripped up and rerouted, this allows us to only reroute congested connections, which helps us to avoid large routing runtimes for the large fanout nets. Others have developed multi-threaded routers. Gort et al.

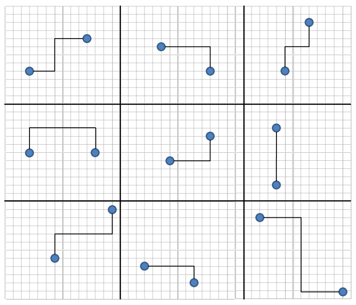

Figure 2: Breaking up sets of nets based on the location of their terminals.

developed a multi-threaded router that splits up the nets in a circuit depending on the location of their terminals on the FPGA [**?**]. Two nets can be routed concurrently in case they have terminals that are not located in each others bound box, as depicted in Figure 2. Another advance is the dynamic expansion of the routing resource graph, which leads to a reduction in memory usage [**?**]. Adapting the routing architecture to speed up the routing is what the authors tried in [**?**].

4 Proposed Techniques

4.1 Placement techniques

For placement we have a clear proposal, because the research group HES, that I would be part of, has developed a placement algorithm with large scale parallelism in mind. The algorithm is called LIQUID and a single threaded version is available [**?**].

LIQUID LIQUID is based on analytical placement. It is also an iterative algorithm in which optimised placement and legalised versions of the placement are generated in each iteration. In contrast to analytical placement there is no linear system that is solved in each iteration. In each iteration all the blocks are moved a few times. The blocks are moved in the direction of the steepest gradient descent. For each block a move vector is calculated based on the previous position of the other blocks that are attached to it. The cost function is the sum of the estimated routing and timing cost of the nets attached to the block. All movement vectors can be calculated independently, which makes it uniquely suitable for parallelisation.

Legalizer In the current implementation the blocks are moved following the steepest gradient descent a few times before the positions are legalised. The legaliser tries to spread, the blocks and assign a legal position to each block without any overlapping blocks. Regions of overlapping blocks are gradually spread, but sometimes two regions clash while expanding the region. This and similar edge cases have to be captured and handled correctly. This is hard to make concurrent, because there are a lot of dependencies between the blocks. Instead, our idea is to keep the movement of the blocks independent and introduce a legalising phase in which we will move the blocks a few steps not following the steepest gradient descent of the cost function, but trying to move each block to a legal position, while also repelling each other to prevent overlap. These moves will not lead to a completely legalised solution, but a nearly legalised solution. Based on experiments we know that it does not really influence the quality of the final solution if the intermediate solutions are not completely legalised. At the end there will be one step of completely legalising, but this will be performed on the CPUs.

Timing analysis In a timing analysis the most critical connections are identified by traversing the timing graph. The connections determine the weights of the nets. In the current implementation of LIQUID, we don't do a timing analysis after each optimize-legalise iteration, but we only do it each 5 iterations, because the criticality of the nets doesn't change extremely between iterations. So there is an option to do timing analysis on the CPUs concurrently, while the moves are being performed by the GPUs, but it will be the subject of research if this will degrade the quality of result of the final placement. Another research path will be to also adapt timing analysis to be able to run it on the GPU. This will involve accelerating breadth-first search, because that is one of the main components of timing analysis. This is not straightforward because it involves graph traversal and pointer chasing, but Merrill et al. showed significant speedups in [**?**].

4.2 Routing techniques

Many multithreading strategies for the routing problem have been proposed in the past, but the limitations of the SIMD architecture present in GPUs forces us to simplify the problem if we want to make use of the massive parallelism offered by the GPU. In what follows we describe the different strategies that will be investigated during the Ph.D. research. Depending on the result, different strategies may be combined.

Route sets of connections in parallel As mentioned in Section 3, the first routing iteration takes the largest portion of runtime because each connection needs to be routed. It is important that the connections in the first iterations are routed independently because in the first iteration the router typically does not take into account congestion cost, but only delay.

Enlightened by the popularity of the A* path-finding search algorithm in large-scale games for concurrently finding the shortest distance between two nodes in a speedy way, we can perform multiple A*-search in the routing resource graph of the FPGA to route the connections in parallel. Making use of GPU's highly parallel multi-threaded nature suits this scenario perfectly [**?**, **?**].

All A*-search wavefronts will be started in parallel. Each thread will behave as a single router and will have its own priority queue. The instructions will be the same for all the router threads and can therefore be executed in one warp. Pop a node from the priority queue. Check if the target is reached and if not add the neighbours to the priority queue. After each router thread finds its target, a new warp can be started with threads that backtrace the path that lead to the target. Each thread spawns its own routing trace on a per cycle basis. For each connection there will be one backtrace thread. All backtrace threads will have the same instructions. They are also responsible for producing a series of history congestion cost updates. These updates are used after a batch of connections is done to build/update the congestion map.

Parallelize the Search A finer grained approach is parallelizing the A*-search in itself. The wavefront is expanded by different threads. A fixed number of nodes are popped from the priority queue and each thread processes the node separately. Each thread checks if the target is reached and if not the cost of the neighbouring nodes is calculated and added to the priority queue in a similar way as done in [**?**]. The wavefront of an A*-search is typically directed towards the goal and so it only makes sense to expand a limited amount of nodes at a time, so we suggest to use the maximum number of threads in one warp, this is typically 32. This finer grained approach could be applied in the later router iterations in which the negotiated congestion mechanism is more sensitive to synchronization and connections have to be routed one after the other or only connections in other locations can be routed concurrently [**?**].

Tile based Compression of the Routing Resource Graph The routing resource graph (RRG) is actually built up of stitched instances of a few regular tiles. An expanded RRG can easily surpass today's workstation memory capacities [**?**]. It is important the routing resource graph fits in the GPU memory banks. An FPGA consists of a grid of logic blocks interspersed with columns of memory blocks and DSP blocks. The interconnection fabric surrounding each type of block is typically adapted to the size and the number of I/O of the block. For each tile type a template RRG tile can be kept in the shared memory of the router threads. It is constant and read-only. Tile-based compression of the RRG has been attempted in the past [**?**], but was only evaluated in a single thread implementation on a CPU.

5 Experimental Evaluation and Expected Outcomes

All experiments will be conducted on a host machine with a dual socket, dual core 2.27 GHz processor and 4 GB of memory. A single NVIDIA GeForce 310M GPU will be used to run CUDA applications. Since the overall high level research goal is to investigate how much the PAR can be accelerated using the proposed solutions and to provide credible results, the performance of the proposed PAR technique will be evaluated and then quantitatively compared to that of the existing popular ones or the improved parallel ones that are performed on CPUs in terms of the runtime, total wire length and maximum circuit clock frequency targeting the same architecture and FPGA circuits. As stated in section 3.1, a mapped circuit involving a description of a specific FPGA architecture is necessary. We plan to adopt academic benchmark circuits and use the commercial tool Quartus II as a front-end to generate the mapped circuits, which will bring several advantages that are hard to replicate in open-source flows [**?**].

Benchmark Circuits The FPGA community relies heavily on benchmarks to evaluate performance of FPGA CAD tools, because benchmark circuits play a key role in the quantitative comparison of tools [**?**]. It is important that these benchmarks reflect modern large-scale systems that make use of heterogeneous resources. In the past, a challenge in evaluating PAR performance and quality of result is the lack of large academic circuits for FPGAs. Thus, to mitigate this problem and provide credible results, some researchers may adopted more than one set of benchmarks [?]. But now the large and complex benchmarks in academic CAD research, Titan23 benchmark suite targeting Altera's Stratix IV FPGA, is available online [?].

Possible Baseline CPU CAD Tools The possible references/baselines can be divided into two types. One is the advanced academic tool, such as VPR wire-length driven placer with default settings and the VPR breadth-first router, and the other can be a commercial one like Altera's Quartus II FPGA CAD software. The fair and robust comparison of the quality and runtime between the proposed PAR and the PAR engines of the commercial one will not only help exam where academic FPGA PAR tools lag behind industry but also definitely inspire potential improvements for our proposed PAR techniques.

Expected Outcomes Efficient parallel placement and routing algorithms will be implemented on GPU or FPGA hardware, resulting in the remarkable decrease of the placement and routing runtime. This hardware acceleration of placement and routing tools will significantly shorten the design cycle for FPGA designs, which will definitely enable a strong reduction of the FPGA design turnaround time and hence speeds up the optimization of FPGA designs. The results of this project published on the journals and conference proceedings will therefore benefit thousands of designers worldwide and facilitate the development of future FPGA architectures and CAD tools.

6 Feasibility

6.1 Background to Support the Research

For this application, we have analyzed many academic publications relating to this project, trying to grasp the essence of this important FPGA placement and routing issue and come up with a clear and specific proposal. Thus we have a full understanding of the scientific significance, tasks and key problems. The reason why the benefits brought by the researchers who also have tried to parallelize PAR algorithms before have not been tremendous is that previous researches mainly focused on the simulated annealing based placers which are very difficult to parallelize. The prospects for paralleling the analytical placement tool LIQUID, developed at Ghent University, are much better. Therefore, we are confident that we will succeed in obtaining very large speed-ups in the placement step (in part I of this research project). Lessons learned from the parallelization of the placement step will help us in finding requirements for a new routing algorithm that is much better suited for parallelization. Then the additional speedup of also parallelizing the routing step (part II of this research project) can also be expected to be obtained.

Additionally for the applicant, this project can be considered as a in-depth branch relating to her knowledge scope, since she has always been working on implementations of FPGA-based system design during her Bachelor's and Master's years, and is familiar with the process of placement and routing. Meanwhile, knowledge of Basic Technology of Computer Software involving data structure and searching algorithms, as well as the essential ability of C language and hardware description language programming is an advantageous complement.

6.2 Context and Strategy of the Research Group

This research will be conducted at Ghent University in the research group Hardware and Embedded Systems (HES) lead by professor Dirk Stroobandt. The HES group has a lot of experience with developping FPGA CAD tool software and has developed the LIQUID analytical placer. The group also has worked a lot on improving routing tools (mainly to extend them in the context of run-time reconfigurable routing). It is the strategy of the group to further its expertise in PAR algorithms, as well as exploit its knowledge on hardware implementations for accelerating algorithms. The group's extensive contacts with the major researchers in the field worldwide and its active presence at the most important FPGA design conferences will further feed this project with ideas and suggestions and will help to keep it focused on its goals.

7 Timetable

The PhD research in this project will be divided in 3 big stages, as shown in Figure 3:

Hardware Acceleration of the placement (24 months)

Adapt the LIQUID placement algorithm for wide-scale parallelization (9+1 months)
In the first step the placer prototype will be further developed and elaborated with the features described in the previous section. After this work there is one month allocated for writing a conference paper to report the initial results.

Implementation of GPU Accelerator for Placement (9+1 months)
After developing the placement algorithm, it will be implemented on the GPU. Measurements and results will be reported in another conference paper and a demonstration will be held at one of the workshops accompanying a conference.

Explore the concurrent execution of timing analysis (3+1 months)
While the GPU implementation is calculating the moves, the timing analysis can be performed concurrently on the CPU. If this does not degrade the quality of the result, we will publish a paper on this, or we will try to accelerate timing analysis on the GPU as well.

Hardware Acceleration of the routing (18 months)

Exploration of new routing algorithms for wide-scale parallelization (8+1 months)
Exploring different routing algorithms and devising new algorithms with wide-scale parallelisation in mind. After exploration, the prototype will be implemented and tested. The results will be reported in a conference publication.

Implementation of GPU/FPGA accelerator for Routing (8+1 months)
After developing the routing algorithm, it will be implemented on the

GPU/FPGA. Measurements and results will be reported in another conference paper and a demonstration will be held at one of the workshops accompanying a conference.

Writing the doctoral thesis and reporting final results (6 months) This time is allocated to write the thesis and to report the finalized results in journals and conferences.

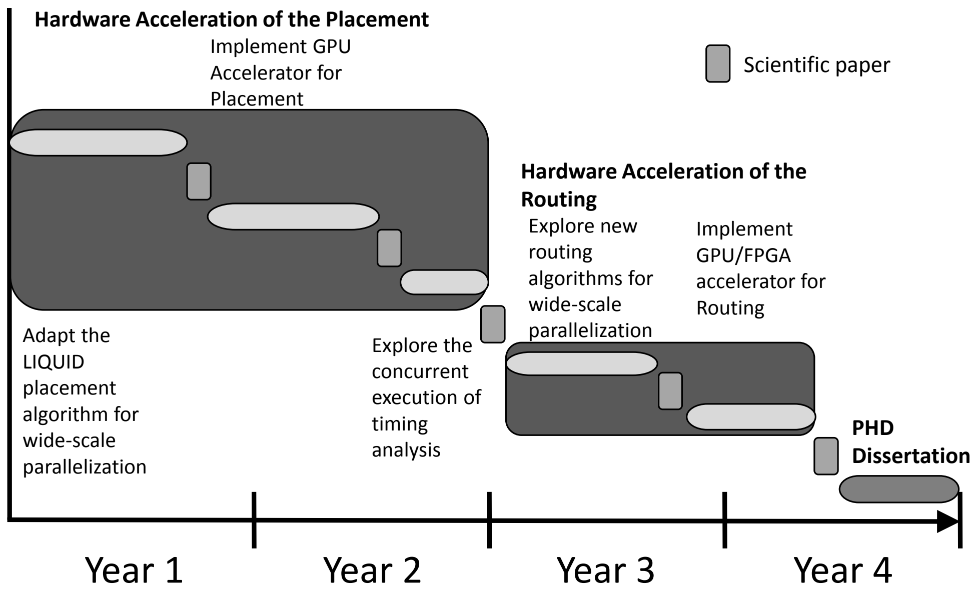
Hardware Acceleration of the Placement
Implement GPU Accelerator for Placement
Scientific paper
Hardware Acceleration of the Routing
Explore new routing algorithms for wide-scale parallelization
Implement GPU/FPGA accelerator for Routing
Adapt the LIQUID placement algorithm for wide-scale parallelization
Explore the concurrent execution of timing analysis
PHD Dissertation
Year 1
Year 2
Year 3
Year 4

Figure 3: Proposed timetable for the PhD research.