Corolla: GPU-Accelerated FPGA Routing Based on Subgraph Dynamic Expansion

Minghua Shen* and Guojie Luo*$^{\hat{\star}}$
Center for Energy-efficient Computing and Applications, School of EECS, Peking University, China*
Collaborative Innovation Center of High Performance Computing, NUDT, China$^{\hat{\star}}$
{msung, gluo}@pku.edu.cn

ABSTRACT

FPGAs are increasingly popular as application-specific accelerators because they lead to a good balance between flexibility and energy efficiency, compared to CPUs and ASICs. However, the long routing time imposes a barrier on FPGA computing, which significantly hinders the design productivity. Existing attempts of parallelizing the FPGA routing either do not fully exploit the parallelism or suffer from an excessive quality loss. Massive parallelism using GPUs has the potential to solve this issue but faces non-trivial challenges.

To cope with these challenges, this work presents Corolla, a GPU-accelerated FPGA routing method. Corolla enables applying the GPU-friendly shortest path algorithm in FPGA routing, leveraging the idea of problem size reduction by limiting the search in routing subgraphs. We maintain the convergence after problem size reduction using the dynamic expansion of the routing resource subgraphs. In addition, Corolla explores the fine-grained single-net parallelism and proposes a hybrid approach to combine the static and dynamic parallelism on GPU. To explore the coarse-grained multi-net parallelism, Corolla proposes an effective method to parallelize mutli-net routing while preserving the equivalent routing results as the original single-net routing. Experimental results show that Corolla achieves an average of 18.72× speedup on GPU with a tolerable loss in the routing quality and sustains a scalable speedup on large-scale routing graphs. To our knowledge, this is the first work to demonstrate the effectiveness of GPU-accelerated FPGA routing.

1. INTRODUCTION

With the slowdown of Moore's Law, the computing landscape is becoming increasingly parallel and heterogeneous, consisting of a larger number of cores and customized accelerators. FPGAs shows particularly promising as an acceleration technology with its reconfigurability and customizability, owing to that they can provide performance and energy improvements in a broad range of applications [1, 2, 3]. For example, Microsoft's large-scale FPGA-based cluster has been used thus far to accelerate Bing web search engine and deep neural network processing [4, 5]. Compared with other competitive accelerators like GPUs, FPGAs usually offer much better energy efficiency and can still deliver high performance for datacenter computing infrastructures. However, the increasingly lengthy compilation time associated with FPGA computer-aided design (CAD) algorithms has been a severe limitation to broader adoption of this technology [6].

Routing is one of the most complex and time-consuming steps in the FPGA CAD flow [6]. Since routing quality directly affects the maximum clock frequency and other design metrics such as routability and power, it also becomes a critical step in the design cycle. The PathFinder routing algorithm [8] is in dominant use in the FPGA community due to its superior performance and quality of results. This algorithm enables the nets to negotiate with each other to find a feasible routing. However, this process is often lengthy in runtime, and a promising direction to overcome the runtime challenge is through parallelization [9]. Several recent works on parallelizing the FPGA routing have been reported [10, 11, 12, 13, 14, 15]. However, there is a lack of literature on the GPU acceleration of FPGA routing. In this paper, we explore how to use GPU efficiently for a fast FPGA routing algorithm.

The Graphics Processing Unit (GPU) provides a massively parallel computing platform to cope with the tedious and time-consuming problems [16]. GPU acceleration techniques show excellent performance in applications with the data parallel paradigm. Several algorithms in the area of FPGA CAD have been successful accelerated using GPUs [17, 18, 19, 22]. However, the PathFinder algorithm for FPGA routing is sequential in nature. Dependencies exist in the routing process of different nets, as well as the routing of a single net. Such dependencies violate the requirement of independence in the data parallel paradigm. Accordingly, the existing routing algorithms must be thoroughly revised to take full advantage of GPU acceleration techniques.

The kernel of FPGA routing is, in fact, a single source shortest path (SSSP) solver. Several GPU-based approaches have been proposed to accelerate the SSSP solver [20, 21], but the available speedup is insignificant due to the synchronization cost and the irregularity of memory accesses [22]. Also, the GPU-accelerated path-finding solvers for video games [23] and the global routing problem for ASICs [24] assume the routing structures as rectilinear grids. There-

FPGA '17, February 22-24, 2017, Monterey, CA, USA

DOI: http://dx.doi.org/10.1145/3020078.3021732

fore, these parallelization techniques cannot be directly applied to FPGA routing, whose complex routing resources form a general graph. Among the GPU-based SSSP solvers for general graphs, the Bellman-Ford algorithm provides the greatest speedup so far [25, 26], although its worst-case time complexity is inferior to the Dijkstra's algorithm. Moreover, the serial Bellman-Ford algorithm excels when running on a small graph [27].

In this paper, we leverage multiple techniques to enable the usage of the GPU-friendly Bellman-Ford algorithm to increase the speedup and restrict its weakness. Specifically, we observe that the bounding box of the final routing tree of most nets is only slightly larger than the bounding box of the pins. Thus, we can use the Bellman-Ford algorithm in a subgraph defined by a limited-size bounding box to replace the Dijkstra or A* algorithm in FPGA routing. We make use of such observation and idea in our GPU-accelerated routing method, named Corolla. Corolla applies the dynamic expansion strategy of the routing resource subgraph to control the problem size so that it can adopt the GPU-based Bellman-Ford algorithm to achieve a high speedup to route a single net. Moreover, we explore the fine-grained node and edge parallelism in the routing of a single net, and we discuss the possibility to leverage coarse-grained net parallelism to route multiple nets concurrently.

In summary, Corolla presents a novel GPU acceleration technique for FPGA routing. The main contributions of this work are described as follows:

- The GPU-friendly Bellman-Ford algorithm becomes practical for FPGA routing, attributed to our problem size reduction technique by exploiting the coverage estimation and dynamic expansion on routing resource subgraphs.
- We further improve the speedup by considering the single-net and multi-net parallelism. On the single-net parallelization, we propose a hybrid approach that combines the advantages of both the static and dynamic parallelism in the SSSP solver for FPGA routing. On the multi-net parallelization, we present a deterministic net-parallel technique that guarantees equivalent routing results as the original ordered net-by-net routing.
- The proposed method provides an average of 18.72× speedup on GPU. We also analyze it scalability using large-scale routing graphs. To our knowledge, this is the first work on utilizing GPU to efficiently accelerate FPGA routing.

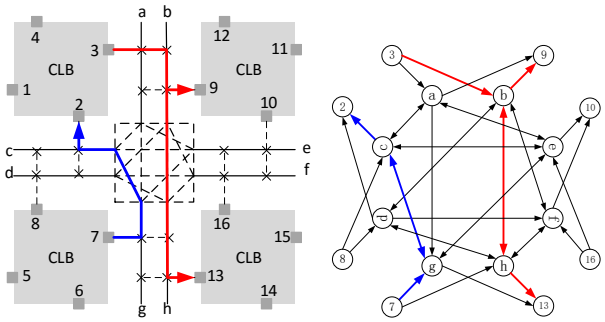
Relying on GPU acceleration, Corolla achieves significantly greater speedups than the publicly available VPR [38] router and the state-of-the-art VPR-based parallel routers. We believe it will have many useful applications and implications for fast physical designs due to the fundamental importance of routing.

2. BACKGROUND

In this section, we review the routing problem and then discuss the negotiation-based PathFinder algorithm.

2.1 FPGA Routing Problem

The routing resources of an FPGA can be modeled as a directed graph $G(V, E)$, the *routing resource graph*, where



(a) Architecture (b) Routing graph

Figure 1: FPGA routing resource graph.

each vertex v_i represents an electrical pin or a wire segment, and each edge e_{ij} corresponds to a programmable connection between an electrical pin and a wire segment, or a programmable routing switch between two wire segments. Fig. 1(a) shows an example of a partial routing architecture, and the corresponded routing resource graph is shown in Fig. 1(b) with a channel width of two.

The routing problem is to find disjoint paths in $G(V, E)$ to connect the pins of the source and the sinks for each net, as shown in Fig. 1(b). A net N_i has one source node s_i and a few sinks t_{ij} that are logically connected to the source. Both the source and sinks are vertices in V, and thus, the net N_i is a subset of V. The routing of net N_i is to find a subtree in graph G that includes all vertices in N_i, and this subtree is called the routing tree RT_i of net N_i. The source s_i is the root node of RT_i, and the sinks t_{ij} are the terminal nodes. The routing trees for different nets are disjoint in G, to prevent short circuits.

In this paper, we focus on accelerating the engine of the negotiation-based router. The same data structures and algorithms can be applied to a router with various objectives, such as timing.

2.2 PathFinder Algorithm

A summary is described below on the negotiation-based PathFinder algorithm [8]. PathFinder routes one net at a time in each iteration, where congestions are temporally allowed in the intermediate routing solutions. The nets must negotiate with each other to decide who will make a detour around the congested resource nodes in subsequent iterations, until all the congestions are resolved to obtain a complete legal routing solution.

Each iteration rips up an existing routing tree and reroutes it by invoking the maze expansion [7], which computes a path from the source to each sink in the routing resource graph. It is also the most computationally expensive task in FPGA routing. All of the unvisited vertices are first stored in a priority queue based on their cost, and the vertex v_{min} with the minimum cost is extracted during maze expansion. If v_{min} is a sink, a routing path will be constructed by invoking a backtrace procedure. Otherwise, each neighbor v of v_{min}, which has not been previously visited, is inserted into the priority queue and the maze expansion continues until a legal routing tree is found.

Prior work has shown that the maze expansion accounts for about 68% of total runtime [12, 13]. Our effort is to completely revamp and accelerate it using GPU acceleration techniques.

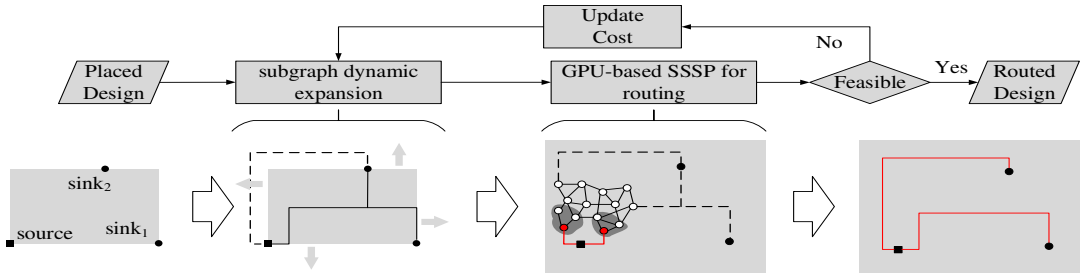


Figure 2: The algorithmic flow of Corolla.

3. OVERVIEW

In this section, we give an overview of Corolla, our GPU-accelerated routing method for FPGAs.

3.1 Motivation

FPGA routing, a tedious and time-consuming process, is sequential in nature. In the coarse grain, the "present costs" in the PathFinder routing algorithm are sequentially updated net after net within an iteration. While in the fine grain, the priority queue in the maze expansion routing of a single net limits the practical concurrency. Such data sharing in the coarse grain and the fine grain violates the requirement of data independence of the GPU-friendly data-parallel paradigm. Thus, the existing routing algorithms are not designed for GPU acceleration and must be revisited.

In this paper, we explore the capability of GPU acceleration for FPGA routing. The computational kernel in FPGA routing is the single-source shortest path (SSSP) solver. Firstly, we present the subgraph dynamic expansion method to enable the use of a GPU-friendly SSSP algorithm for FPGA routing in Section 4. Secondly, we explore different GPU-based parallelizations and propose an efficient hybrid solution, followed by multi-net parallelization in Section 5.

3.2 Algorithmic Flow

The algorithmic flow of Corolla is shown in Fig. 2. In the overall flow, we preserve the negotiation-based framework that iteratively reduces the routing congestion by ripping up and rerouting the nets. In subgraph dynamic expansion, the routing subgraph of each net is extracted according to the *initial coverage* strategy at the first iteration, and its size may be expanded according to the *dynamic expansion* strategy. In the GPU-based SSSP for routing, we propose multiple techniques to improve parallelism and speedup. We explore the node and edge parallelism and a hybrid approach to accelerate the single-net routing on GPU. We also leverage the net parallelism to accelerate the multi-net routing. The kernel of Corolla is the GPU-friendly SSSP algorithm to route every net inside its routing subgraph.

Corolla guarantees deterministic results, although it produces different results from the original PathFinder algorithm. Assume there is a sequential version of Corolla that applies the same subgraph dynamic expansion strategy, it is obvious that the single-net parallelization is sequential equivalent, and we will propose a multi-net parallelization that is also sequential equivalent.

4. SUBGRAPH DYNAMIC EXPANSION

The computational kernel in FPGA routing is a solver for the single-source shortest path (SSSP) problem, usually using Dijkstra's algorithm or A* search[1]. The fundamental data structure in both algorithms is a priority queue, which causes contentions and bottlenecks in a GPU implementation. Therefore, most existing literature and publicly-available solvers for the GPU-accelerated SSSP algorithm are based on the Bellman-Ford algorithm for a greater parallelism and speedup, although the worst-case sequential time complexity of the Bellman-Ford algorithm is higher than the Dijkstra's algorithm.

To take full advantage of the existing GPU-based SSSP solvers for FPGA routing, our basic idea is to alleviate its time complexity by reducing the problem size. In general, there are at least two approaches to doing so:

1) One is to perform global routing in a coarsened routing graph.

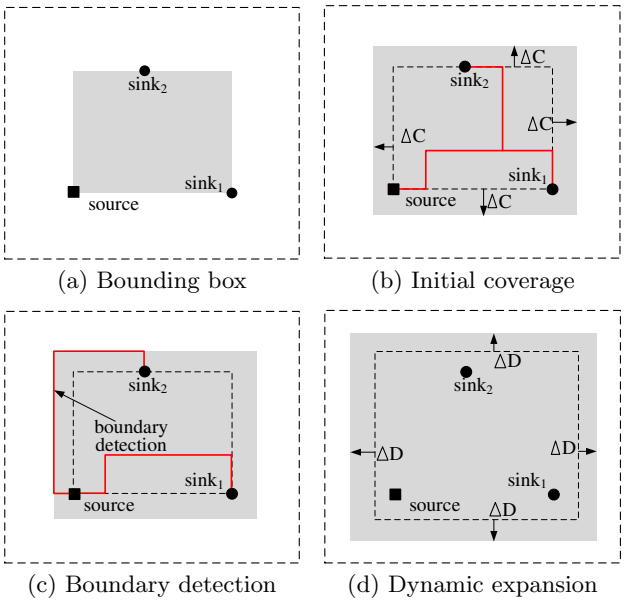
2) The other is to restrict the search space for the SSSP algorithm.

The global routing approach is useful for ASIC routing, but it is less effective for FPGA routing [28]. It may be possible to obtain a pseudo-rectilinear structure from the FPGA routing graph by clustering the routing segment nodes inside the same channel. Taking Fig. 1 as an example, one may cluster the nodes a and b, c and d, e and f, g and h, respectively, so that the clustered nodes form a rectilinear structure for global routing. However, this conversion is not accurate to model the congestion, because it cannot distinguish the congestion cost of the segment nodes in the same channel. According to our analysis of the final routing results of the PathFinder algorithm across multiple benchmarks, we observe that the routing cost of some segment nodes in the same channel differ significantly. The maximum difference (e.g., 24) of the routing cost for the nodes in the same channel usually greater than the mean (e.g., 9) plus one standard deviation (e.g., 10) among the nodes with non-zero costs. Therefore, the global routing approach is not the best choice to reduce the problem size, and we resort to the other approach by restricting the search space.

To ensure the correctness and convergence of the FPGA routing algorithm, we propose the method of *subgraph dynamic expansion* to limit the search space. It contains three essential steps to mitigate the disadvantages of the Bellman-Ford algorithm.

- **Step 1:** subgraph extraction, which is implemented

[1]Dijkstra's algorithm and A* search have similar data structures and algorithmic flow. Thus, we only mention Dijkstra's algorithm to compare with the GPU-friendly Bellman-Ford algorithm in the rest of this paper for conciseness.



(a) Bounding box (b) Initial coverage

(c) Boundary detection (d) Dynamic expansion

Figure 3: The three steps of subgraph dynamic expansion in Corolla: (a) obtain the net bounding box before routing, (b) estimate the initial coverage that provides most nets a sufficiently large subgraph to route, and (c) perform boundary detection to trigger the (d) dynamic expansion to guarantee that the routing subgraph is eventually large enough.

efficiently based on a labeling system that relates the coordinates to the routing resource nodes.

- **Step 2:** initial coverage, which is preprocessed by analyzing the routing results of existing circuits. Our estimation of the initial routing subgraphs provides sufficient routing nodes (e.g., for 98.5% nets in existing circuits), so that each net only needs to explore its routing tree in a small subgraph instead of the overall routing graph.

- **Step 3:** dynamic expansion, which is complementary to initial coverage using a detection strategy to adaptively expand the routing subgraph in a next iteration until a feasible solution is found.

Notice that the key idea of subgraph dynamic expansion is to estimate and find a large-enough routing subgraph for every net to obtain its feasible route. In the initial coverage, we first determine the initial routing subgraphs to cover a significant portion of nets, and then we perform dynamic expansion in case that a few nets need a larger subgraph to find their legal routing trees. This method effectively bounds the number of nodes during the routing exploration, and thus alleviates the complexity overhead of the Bellman-Ford algorithm compared to the Dijkstra's algorithm.

The usage of subgraph dynamic expansion in Corolla is illustrated in Fig. 3. For example in Fig. 3(a), a straightforward estimation of the routing subgraph is the one within the bounding box of *source*, *sink1* and *sink2*. Using the bounding box to estimate the initial subgraph does not provide enough routing resources in many cases, and thus, Corolla determines a good-enough subgraph at the initial coverage stage, as shown in Fig. 3(b). The static estimation is unlikely 100% accurate, and a detour path outside the initial coverage may be necessary for a legal routing solution. So we apply an adaptive strategy to expand the subgraph, whenever a detour path touching the boundary of the current routing subgraph is detected, as illustrated in Fig. 3(c). Finally, Corolla can find a routing solution inside the estimated and expanded subgraph as shown in Fig. 3(d). Evaluations show that this approach effectively reduces the problem size without affecting the quality of the routing solution.

4.1 Subgraph Extraction

As discussed earlier, due to the limitations of the global routing approach for FPGAs, we resort to the method of restricting the search space (i.e., the routing subgraph) for the GPU-based SSSP solvers. The ideal restricted search space with the minimum number of routing resource nodes should let an SSSP algorithm generate the same or similar result as in the original search space. However, this very ideal case is non-trivial to obtain. In addition, extracting an irregular subgraph requires some timing-consuming graph traversals that affect the efficiency. Thus, we relax the ideal routing resource subgraph to be a net-specific box, which contains sufficient routing resource nodes for the given net. In the following, we present the details how to extract a routing resource subgraph inside a box.

We make the following assumptions for the proposed subgraph extraction. The routing graph consists of pin nodes and segment nodes. A pin node corresponding to a pin of a logic block is assigned with the placement coordinates of this logic block. For every segment node, there exists an edge connecting to a pin node, and this segment node shares the same coordinates with its neighboring pin node. Since some segment nodes are neighbors of two or more pin nodes with different coordinates, these segment nodes have multiple coordinates.

The subgraph corresponding to a box in the FPGA routing region includes all the pin nodes and segment nodes with coordinates inside this box, as well as the routing edges between these nodes. Given any box in the FPGA routing graph, we can efficiently extract the subgraph in the box defined above. In the next two subsections, we will discuss how to determine the size of such box for any given net and guarantee the convergence of the routing algorithm.

4.2 Initial Coverage

The routing subgraph defined in the previous subsection reduces the problem size. The next question is how to find the dimension of the box for the subgraph extraction, given any net to be routed.

As discussed earlier, to ensure the correctness and convergence of the routing algorithm, the box should contain sufficient routing resources for a given net, and its size is expected to be as small as possible. A simple choice is the minimum bounding box of the pins in a given net, which is available before routing. But this simple choice rarely provides enough routing resources. Another choice is the minimum bounding box of the final routing tree, which is of course only available after the routing finishes. This choice provides sufficient routing resources but is impractical and not implementable. To get a good-enough initial subgraph before routing, we propose the *initial coverage* in Corolla to estimate the initial boxes to cover a significant portion of nets with enough routing resources. The idea of this esti-

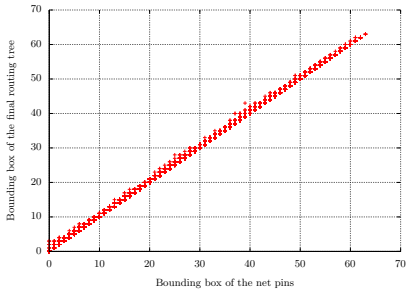


Figure 4: The bounding box of the final routing tree is only slightly larger than the bounding box of the net pins for Δ_C.

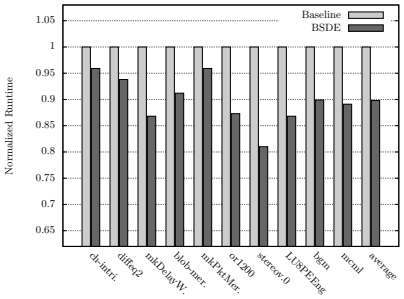


Figure 5: Comparisons of the sequential routing runtime between the baseline and BSDE.

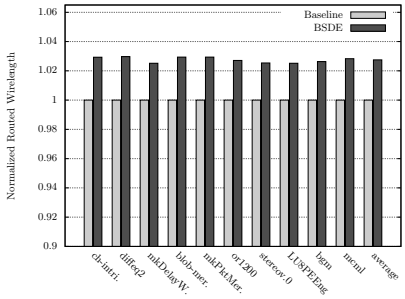


Figure 6: Comparisons of the wirelength degradation between the baseline and BSDE.

Table 1: An example of the initial expansion factor of 0.021 for a 98.5% coverage

Bench.	Δ_C	array	$\frac{\Delta_C}{\sqrt{\text{array}}}$	coverage
diffeq2	1	34×34	0.029	100%
mkDelayW.	1	48×48	0.021	99.8%
blob_mer.	1	51×51	0.020	100%
mkPKtMer.	1	58×58	0.017	100%
or1200	1	65×65	0.015	100%
LU8PEEng	1	53×53	0.019	99.3%
bgm	2	73×73	0.027	99.5%
mcml	2	101×101	0.020	98.7%
average	-	-	0.021	-

mation is based on the statistical data from existing routed circuits.

An important observation is that for most nets, the size of the bounding box of the net pins is only slightly greater than the bounding box of the final routing tree, as illustrated in Fig. 4. Based on this empirical relation based on existing routed circuits, Corolla statistically estimates the size of the boxes for the routing subgraphs during the initial coverage. The box of a given net in the initial coverage is expanded from the four sides of the bounding boxes of net pins by a distance of Δ_C, as illustrated in Fig. 3(b). The estimation of Δ_C relates to the FPGA size, as well as a user-defined percentage of coverage.

Here we describe an example flow to estimate Δ_C given a few circuits with known routing solutions. The distance Δ_C is the difference in the left, right, top and bottom boundary coordinates between the bounding box of the net pins and the bounding box of its final routing tree, which can be collected from these existing circuits. If the user-defined percentage of coverage is 98.5%, we find the smallest coverage that is not less than 98.5% of the nets in every given circuit. We observe that by dividing this value of Δ_C by the FPGA array size, we obtain a similar ratio, 0.021 on average, among many circuits. We call this ratio the *initial expansion factor*. Examples of this initial expansion factor are shown in Table 1.

Therefore, given a new circuit, we can multiply the FPGA array size by the initial expansion factor and round it up to the next integer to obtain Δ_C for the initial coverage. By applying this rule, we can estimate how much we should expand the bounding box to be the initial coverage when constructing of the initial routing resource subgraphs.

4.3 Dynamic Expansion

Though the initial coverage provides enough routing resources for most nets, there are still some outliers. A simple fix is to expand the subgraphs continuously to ensure sufficient routing resources eventually. However, such strategy will increase the routing time due to some unnecessarily large subgraphs. In Corolla, we use the *dynamic expansion* strategy, which contains a detection method to expand a subgraph only when necessary.

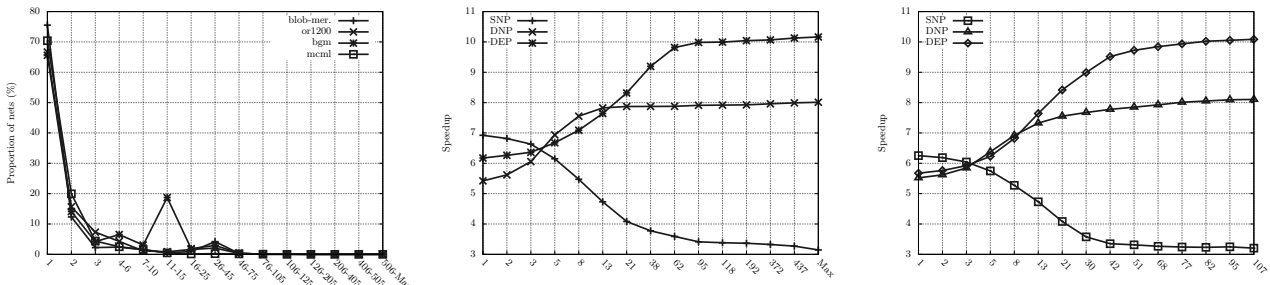
The dynamic expansion is based on a boundary detection strategy, which is used to decide whether Corolla continues to expand the size for the routing subgraph of a net. A net is likely to use more routing resources when its routing tree occupies a node on the boundary of the current routing subgraph. Once detected, Corolla expands the box of its routing subgraph on the four sides by a distance of Δ_D[2] in the next iteration. With this boundary detection strategy in dynamic expansion, Corolla can converge using a similar number of iterations as the original PathFinder algorithm.

The subgraph dynamic expansion consists of subgraph extraction, initial coverage, and dynamic expansion. Its purpose is to reduce the problem size to remedy the worst-case time complexity of the Bellman-Ford algorithm, which is GPU-friendly and has efficient GPU-based implementations. Before discussing the GPU acceleration in the next section, here we evaluate the impact of subgraph dynamic expansion on the routing quality and the runtime, as well as the impact of replacing the Dijkstra's algorithm by the Bellman-Ford algorithm. Two approaches are evaluated and compared, including: the original PathFinder algorithm (baseline), and the modified PathFinder algorithm using the Bellman-Ford algorithm with subgraph dynamic expansion (BSDE).

Fig. 5 show the normalized runtime of these two approaches. Though the worst-case time complexity of the Bellman-Ford algorithm is greater than the Dijkstra's algorithm, the runtime of BSDE is obviously better than the baseline with a negligible impact on the routed wirelength. Fig. 6 reports the normalized routed wirelength of the two approaches, where the routed wirelength is increased by about 2.7% on average, and detailed explanation will be presented in Section 6.

Fig. 5 and Fig. 6 illustrate the effectiveness of the Bellman-Ford algorithm combined with subgraph dynamic expansion for FPGA routing. This computational kernel is GPU-friendly and is, therefore, a good candidate for the GPU-accelerated FPGA routing.

[2]This parameter is empirically set to one, which is sufficient to find a legal routing solution according to our experiments.



(a) Percentage of nets with different sink counts

(b) Speedup with respect to the number of sinks

(c) Speedup with respect to the HPWL

Figure 7: Hybrid approach by combining SNP, DNP, and DEP.

5. GPU-ACCELERATED ROUTING

In this section, we present the details of acceleration techniques in Corolla.

5.1 Dynamic Parallelism

Dynamic parallelism is an important and useful technique to dynamically exploit the parallelism in irregular computations such as graph algorithms. Moctar and Brisk [13] demonstrate the effectiveness of the dynamic parallelism with multi-threading by using the operator formulation [30] for FPGA routing.

The general idea of the operator formulation to implement dynamic parallelism is to apply a *compute* operator iteratively on a subset of nodes in a graph. At each iteration, the active nodes perform useful computations, and the rest inactive nodes are idle. A *check* operator determines whether a node is active or inactive. The compute operator often accesses neighboring nodes and can activate inactive nodes for further processing. Execution completes when all nodes are inactive and will not be activated again.

For example, in the Bellman-Ford algorithm, a compute operator updates the known shortest path of a node, and a check operator checks whether an upstream of a given node has an updated known shortest path. The operator formulation is a framework that automatically parallelizes a program where the compute and check operators are defined.

In Corolla, we explore the dynamic parallelism of routing resource nodes and edges to accelerate GPU-based FPGA routing.

5.2 Parallelization Exploration

Considering the specific structure of routing resource graphs, the previous experiences of parallelization strategies for general graphs cannot be directly applied to routing resource graphs. Thus, it is necessary to explore and examine the effectiveness of three different kinds of parallelism in the GPU-based Bellman-Ford algorithm for FPGA routing in Corolla:

1. Static node parallelism (SNP), where every node, no matter active or not, is assigned to a thread to process in parallel.

2. Dynamic node parallelism (DNP), where only the active nodes, i.e., the nodes whose known shortest distances to the source node have recently been changed, are assigned to threads to process in parallel.

3. Dynamic edge parallelism (DEP), which is similar to DNP but considers assigning active edges to threads instead of active nodes.

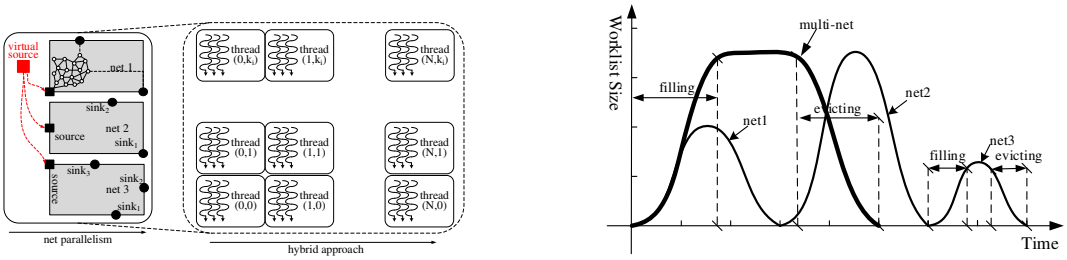
In SNP, every thread first checks whether its responsible node is active. If active, it then applies the compute operator to update the known shortest path of the active node in each superstep. Every kernel execution on the GPU forms a superstep, and the kernel is invoked again in the next superstep when active nodes exist. All nodes, including active and inactive, are statically assigned to the threads through a block decomposition during the parallelization in every superstep.

In DNP, a centralized worklist with atomic memory operation (AMO) is used to manage the dynamic parallelism. First, an initialization step pre-checks all the nodes and populates the active nodes into the worklist for parallel processing. For example, to route a net, the worklist is initialized with the source node. Second, every thread pulls an active node from the worklist using AMO and then applies the compute operator to the corresponding active node. The newly activated nodes are pushed onto the worklist with AMOs such that only active nodes will be visited in the next iteration. This process is repeated until the worklist becomes empty. Compared with SNP, DNP exposes more parallelism and improves the efficiency by mapping threads to useful computation work. However, the DNP also present its weakness with high memory contention when accessing a shared worklist.

A better implementation can be obtained by exploring the DEP using Merrill's method [31]. Besides focusing on the edges instead of nodes, DEP use a prefix scan to allocate a chunk of memory for each thread to maintain the active nodes so that it relieves the contention of atomic accesses by avoiding AMO. These chunks of memory are assigned to the CUDA blocks, which work in parallel to check the edges in their assigned chunks, using various heuristics to trade-off time and space for a high throughput.

5.3 Hybrid Approach

Either static or dynamic parallelism has its merits and demerits. Although slower than DNP and DEP for large graphs, the static SNP method achieves greater speedup for the low-fanout nets than the dynamic DNP and DEP methods. Our explanation is that the routing subgraph of a low-fanout net only has a small number of routing nodes, so the static assignment of GPU threads to these nodes is efficiently executed on the hardware. In Corolla, we present a hybrid approach to exploit the merits of both the static and dynamic parallelisms.



(a) Concurrent routing on GPU

(b) Sources of speedup

Figure 8: Multi-net parallelization using our hybrid approach.

To seek higher speedup for FPGA routing, we analyze how different attributes of a net affect the runtime of different methods. Fig. 7(a) shows the percentage of the nets with a different number of sinks on four representative circuits. The number of low-fanout nets is significantly higher than the high-fanout nets. Thus, there is an opportunity to improve the speedup using a net-specific parallelization strategy. Fig. 7(b) shows the speedup using the SNP, DNP, and DEP methods of the nets with a different number of sinks in the *or1200* circuit. For this specific circuit, we observe that the static SNP method is better than the dynamic methods for the nets with fewer than three sinks, and the dynamic DEP method is superior to the others for the nets with more than thirteen sinks. Moreover, we explore the speedups from the SNP, DNP and DEP methods with respect to the HPWL of the nets and observe similar results, as shown in Fig. 7(c). We also observe similar patterns for other circuits by conducting the same set of experiments.

These results reveal the opportunity to combine different methods to improve the speedup. We propose an efficient hybrid approach that uses SNP for the nets with less than or equal to three sinks and uses DEP for the remaining nets. Though there is a possible gain using the DNP method for the nets with a moderate amount of sinks, these nets only contribute to a small percentage of runtime, and we simply apply DEP instead. We will present experimental evaluations in Section 6.2, which will show that our hybrid approach is as efficient as an "optimal" combination of SNP, DNP, and DEP.

5.4 Multi-Net Parallelization

In the previous subsections, we explore the fine-grained node and edge parallelism using the SNP, DNP, DEP, and the hybrid approaches. In this subsection, we will explore the coarse-grained net parallelism to achieve a further speedup on GPU. Specifically, during the multi-net parallelization, we maintain equivalent routing results as the single-net parallelization.

According to previous works [33], the routing sequence of the nets affects the routing quality. To explore the net parallelism while maintaining deterministic results, we impose a restriction that the routing results are equivalent to the single-net routing according to the original net ordering. The parallelization is done in a greedy fashion: we collect as many as nets as possible according to their original order, and make sure that their concurrent routing will not affect their routing results compared to the single-net routing. By using the routing subgraphs of the nets, this collection process can efficiently and precisely check the dependency among the nets in each iteration.

Table 2: Benchmark summary

Bench.	Arch.	Dim.	Nets	CLBs
ch-intri.	k4_N4_90nm	20x20	788	497
sha	k4_N4_90nm	29x29	1946	866
boundtop	k4_N4_90nm	19x19	2380	724
diffeq2	k4_N4_90nm	34x34	3710	1296
diffeq1	k4_N4_90nm	35x35	3953	1450
mkDelayW.	k4_N4_90nm	48x48	5224	1554
blob_mer.	k4_N4_90nm	51x51	6606	2702
mkSMAdap.	k4_N4_90nm	53x53	7154	3126
mkPKtMer.	k4_N4_90nm	58x58	7474	3767
or1200	k4_N4_90nm	65x65	8078	3648
stereov.0	k6_N10_40nm	39x39	9312	1492
stereov.1	k6_N10_40nm	39x39	13523	1401
LU8PEEng	k6_N10_40nm	53x53	16278	2373
bgm	k6_N10_40nm	73x73	27853	4225
stereov.2	k6_N10_40nm	86x86	36479	2802
mcml	k6_N10_40nm	101x101	81282	7934

The subgraph extraction approach mentioned in Section 4.1 provides a good indicator for this dependency detection. The collection of the subset of concurrent nets starts with the current net to be routed. Then we gradually add the next a few nets according to the original order to this subset, until the routing subgraph of a next net has overlaps with one of the nets in this subset. It is evident that this subset can be routed in parallel without affecting the routing results. Hence, our multi-net parallelization guarantees deterministic solutions.

Fig. 8(a) demonstrates the concurrent routing of multiple independent nets on GPU. To unify the single-net and multi-net routing, a virtual source node is introduced to directly connect to the actual source nodes of the independent nets. Thus, the independent nets can be combined into a single pseudo net, which can be routed on GPU leveraging the single-net acceleration techniques as discussed previously.

Fig. 8(b) explains the sources of speedup when routing multiple nets in parallel. The vertical axis reveals how the size of the worklist, which is the number of edges processed concurrently in the GPU-based SSSP algorithm, varies with respect to the execution time. The speedup comes from the reduction of the filling time of the worklist at the beginning and the evicting time near the end. So that when we route multiple nets as a single pseudo net, some overheads in these two phrases are eliminated to improve the speedup.

6. EXPERIMENTAL EVALUATIONS

In this section, we present and analyze the acceleration results of Corolla. our GPU-accelerated FPGA routing method based on subgraph dynamic expansion.

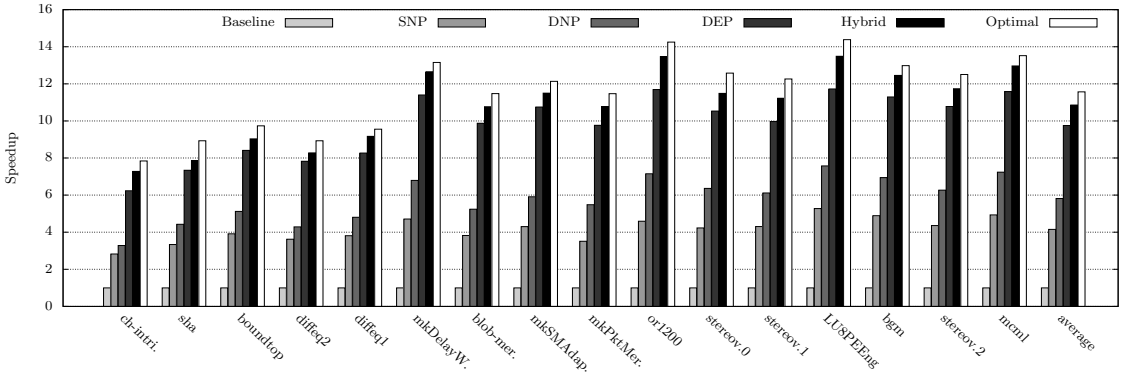


Figure 9: Single-net acceleration on GPU using SNP, DNP, DEP, and Hybrid approach.

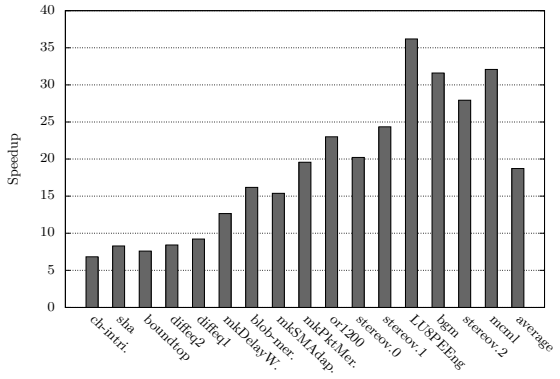


Figure 10: Multi-net acceleration on GPU using the hybrid approach.

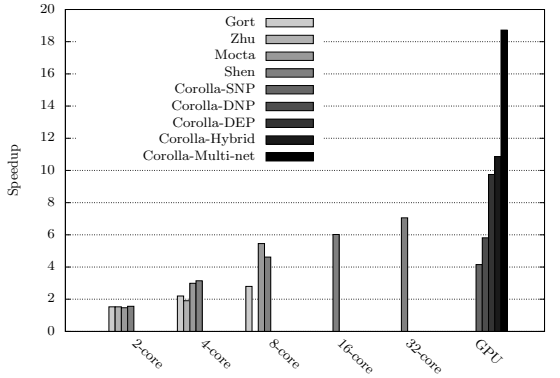


Figure 11: Speedup of SNP, DNP, DEP, Hybrid, and Multi-net routing compared to existing works.

6.1 Experimental Setup

All experiments are performed on a Linux server with a 6-core Intel Xeon E5-2620 CPU at 2.2GHz and 32 GB shared memory, equipped with a Tesla K40c GPU having 2880 cores in 15 streaming multiprocessors and 12 GB video memory. The baseline for comparison is the original VPR 7.0 router [38], which is a sequential program implemented in C. Some of the GPU implementations of the SSSP algorithm are adapted from the source code in the LonestarGPU collection [29].

All experiments are run with sixteen representative circuits from the VTR benchmarks [38] commonly used in FPGA CAD research. Table 2 summarizes the characteristics of these benchmarks, including the array size of the routing region generated, the number of nets, and the number of configuration logic blocks (CLBs) used. We use ABC [39] for logic synthesis and technology mapping, T-VPack for packing, and VPR placer [38] for placement, respectively. Across all runs, each benchmark is routed using a channel width of 1.3× the minimum channel width needed by VPR, following the same configurations as in the previous works [12, 13, 14].

6.2 Speedup Analysis

Fig. 9 shows the speedups using four different parallelization techniques in Corolla. The speedups of each benchmark are shown in a cluster of bars. The leftmost bar is the baseline, and the next four bars show the speedups of SNP, DNP, DEP, and the hybrid approach in Corolla. To illustrate the effectiveness of our hybrid approach, we include the speedups of an optimal hybrid approach in the last bar. The average speedups over all benchmarks are shown in the last cluster. On average, we achieve a speedup of 4.15×, 5.82×, 9.75×, and 10.86× with the SNP, DNP, DEP, and Hybrid, respectively. The runtime of the optimal hybrid approach is estimated by summing up the fastest possible runtime of each net using either SNP, DNP, or DEP, assuming there is an oracle to predict the optimal selection. The 10.86× speedup of our hybrid approach in Corolla is only slightly less than the speedup of the optimal hybrid approach, 11.57× on average.

We can observe in Fig. 9 that the hybrid approach in Corolla achieves more speedup than the parallel methods of SNP, DNP, and DEP. The reason is that our hybrid approach invokes SNP to route a significant number of low-fanout nets, and routes the timing-consuming multi-sink nets using DEP. Moreover, the hybrid approach is compatible with the coarse-grained parallel methods [12, 14] to achieve a further speedup.

Fig. 10 presents the results of the acceleration of multi-net routing on GPU using the hybrid approach. It can be seen that this approach produces an average speedup of 18.72× using a single GPU. This is a 3.43× improvement over the recent fine-grained parallel router [13], and a 2.67× enhancement over the recent coarse-grained parallel router [14]. We do obtain notable speedups for the four largest benchmarks on the right in the figure, owing to that more independent nets can be combined into a single pseudo net and result in more acceleration on GPU. Moreover, this approach is promising to be extended with the multi-GPU parallelization to improve the speedup greatly [31].

Finally, we list the speedups of previous coarse-grained and fine-grained parallel FPGA routers, compared to the sequential VPR router in Fig. 11. By taking advantage of GPU acceleration, Corolla achieves significant speedups. It is the first work to accelerate FPGA routing using GPU. It is also the first work to achieve near 20× speedup for the FPGA routing problem.

6.3 Quality Analysis

In Fig. 12, we compare the routing quality of the multi-net parallelization in Corolla with the sequential VPR regarding the routed wirelength. The degradation mainly comes from the multi-sink nets. The original VPR router performs the Dijkstra's algorithm many times to obtain a final routing tree. And Corolla directly combines the shortest paths from the single source to the multiple sinks in a single round of the Bellman-Ford algorithm into a full routing tree.

Corolla only introduces a 2.73% degradation in wirelength on average, compared to the original VPR router. This impact is negligible for the scenarios such as the synthesis of reconfigurable FPGA accelerators in datacenters and fast design iterations in early design stages. In the former case, the delay degradation is insignificant compared to the orders of magnitude speedups introduced by FPGA acceleration. And in the latter case, the design quality is not as important as the design productivity.

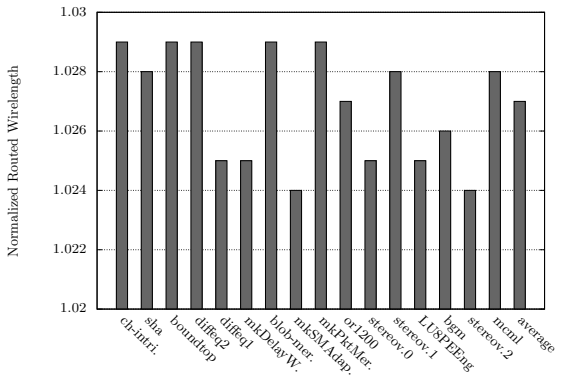


Figure 12: Impacts on the routed wirelength using the multi-net parallelization in Corolla.

6.4 Scalability Analysis

With the FPGA integration density scales, the routing resource graph will continue to grow every generation. Thus, a scalable routing algorithm becomes essential to provide similar speedup when the size of routing graph grows. Here, we evaluate the multi-net parallelization in Corolla on large-scale routing graphs.

We construct synthetic designs on large-scale routing graphs based on given benchmarks. The locations of the sources and sinks of the nets in a benchmark are linearly stretched when we extend the FPGA array size from 100×100 to 1000×1000. The routing resource graph for the 1000×1000 array size is at the same scale as the largest benchmark in Titan [6].

Fig. 13 gives the speedups obtained on three representative benchmarks over the sequential VPR router. Corolla achieves the best speedup for the FPGA array size around *500×500*. While the speedup decreases slowly when the FPGA array size grows, Corolla still scales well. The scalability of Corolla is attributed to two reasons: 1) The subgraph dynamic expansion strategy effectively reduces the problem size. 2) The GPU-based SSSP is a variant of the Bellman-Ford algorithm, which makes use of the worklist that behaves similarly to a queue. Actually, in many practical cases, the Bellman-Ford algorithm converges faster than the worst-case analysis [35, 36], and there exist examples [37] that the queue-based Bellman-Ford algorithm spends less computation than the Dijkstra's algorithm. Here, Corolla provides another example that the variant of queue-based Bellman-Ford algorithm is practical for FPGA routing. We observe the same trends for other benchmarks and therefore, these results indicate that Corolla is promising to maintain a similar speedup for large-scale routing graphs.

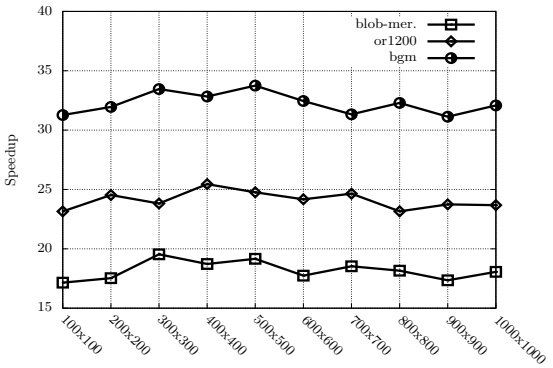


Figure 13: Scalability Analysis with different FPGA array sizes.

7. RELATED WORK

Most of the previous works on parallelizing FPGA routing are motivated by the acceleration of the overall synthesis time. The first parallel PathFinder algorithm is proposed by Chan and Schlag [10]. By modeling the routing problem into a graph or hypergraph matching problem, they analyze how and when the history and present congestion cost of the PathFinder routing algorithm should be synchronized across the processors to ensure convergence while improving parallelism [32]. Although their method is highly sensitive to the order of the nets to be routed, it is still an open problem to determine the best net ordering [33]. Quite noticeable is that a speedup of 2.5× is attainable using three processors on a distributed cluster.

The fine-grained parallelization avoids the influences of net ordering in the PathFinder routing algorithm. Dehon et al. [34] propose the design of a hardware accelerator for FPGA routing. Their simulations predict a speedup of up to three orders of magnitude over PathFinder with 5%-25% loss in solution quality. Zhu et al. [11] partition the high-fanout nets into several low-fanout subnets to be routed in parallel. They achieve a speedup of 1.9× on a quad-core processor platform with 2.3% loss in solution quality. And then Moctar and Brisk [13] explore the dynamic parallelism using the Galois API. They achieve a good speedup of 5.4× using eight threads. Recently, Hoo et al. [15] propose a fully parallel router based on Lagrangian relaxation to decompose the original routing problem into independent subproblems. This approach produces an average speedup of 7× using eight threads, compared to its sequential version.

While the coarse-grained parallelism is sensitive to the net ordering, Gort and Anderson [12] propose a deterministic parallel PathFinder routing algorithm. They partition the nets into subsets, and these subsets are routed in parallel with an efficient synchronization scheme to guarantee the deterministic results. Although they did not emphasize the scalability, it is the first deterministic parallel routing algorithm and achieves a 2.8× speedup using eight cores. Another deterministic parallel router is proposed by Shen and Luo [14], using a partitioning-based parallel routing method. They leverage a dynamic programming algorithm to determine the optimal recursive partitioning strategy. Although it degrades the quality of routed wirelength, the parallel router exploits more parallelism and scales to a 32-core cluster with an average speedup of 7×.

8. CONCLUSIONS

In this paper, we present Corolla, which leverages the GPU techniques to accelerate FPGA routing. In Corolla, we first use the approach of subgraph dynamic expansion to obtain convergent routing results with a reduced problem size. This approach enables the efficient application of the GPU-friendly Bellman-Ford algorithm to replace the Dijkstra's algorithm in PathFinder. We then perform systematic experiments and comparisons among different GPU accelerations of the Bellman-Ford algorithm, including the SNP, the DNP, and the DEP approaches. We point out that we can combine the static SNP and dynamic DEP methods for a greater speedup and exploit the multi-net parallelism, where we achieve an average of 18.72× speedup on GPU.

9. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive comments. This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61520106004.

10. REFERENCES

[1] K. Atasu et al. *Accelerating text analytics queries on reconfigurable platforms.* In Workshop. CARL, 2015.

[2] P. Gupta. *Xeon+fpga platform for the data center.* In Workshop. CARL, 2015.

[3] T. Brewer. *Convey Acceleration of the Memcached and Imagemagick Applications.* In Workshop. CARL, 2015.

[4] A. Putnam et al. *A reconfigurable fabric for accelerating large-scale datacenter services.* In Proc. ISCA, 2014.

[5] K. Ovtcharov et al. *Accelerating deep convolutional neural networks using specialized hardware.* In White paper, 2015.

[6] K. E. Murray and V. Betz. *Titan: Enabling large and complex benchmarks in academic CAD.* In Proc. Field Programmable Logic and Appl., 2013.

[7] C. Lee. *An algorithm for path connections and its applications.* In Proc. IRE Trans. on Electronic Computers, 1961.

[8] L. McMurchie and C. Ebeling. *Pathfinder: A negotiation-based performance-driven router for FPGAs.* In Proc. Field Programmable Gate Arrays, 1995.

[9] B. Catanzaro, K. Keutzer, and B. Su. *Parallelizaing CAD: A timely research agenda for EDA.* In Proc. ACM Design Automation Conference 2008.

[10] P. Chan and M. Schlag, *Acceleration of an FPGA router.* In Proc. Field-Programmable Custom Computing Machines, 1997.

[11] C. Zhu, J. Wang, and J. Lai. *A novel net-partition-based multithreaded FPGA routing method.* In Proc. Field Programmable Logic and Appl., 2013.

[12] M. Gort and J. Anderson. *Deterministic multi-core parallel routing for FPGAs.* In Proc. Field Programmable Logic and Appl., 2010.

[13] Y. Moctar and P. Brisk. *Parallel FPGA routing based on the operator formulation.* In Proc. Design Automation Conf., June 2014.

[14] M. Shen and G. Luo. *Accelerate FPGA routing with parallel recursive partitioning.* In Proc. Int. Conf. on Computer Aided Design, 2015

[15] C. Hoo, A. Kumar, and Y. Ha. *ParaLaR: A parallel FPGA router based on lagrangian relaxation.* In Proc. Field Programmable Logic and Appl., 2015

[16] J. Croix and S. Khatri. *Introduction to GPU programming for EDA.* In Proc. Int. Conf. on Computer Aided Design, 2009.

[17] N. Kapre and D. Ye. *GPU-accelerated high-level synthesis for bitwidth optimization of FPGA datapaths.* In Proc. Field Programmable Gate Arrays, February 21-23, 2016.

[18] D. Chen and D. Singh. *Parallelizing FPGA technology mapping using Graphics Processing Units.* In Proc. Field Programmable Logic and Appl., 2010.

[19] C. Fobel and D. Stacey. *GPU-accelerated wire-length estimation for FPGA placement.* In Proc. Application Accelerators in High-Performance Computing. 2011.

[20] P. Harish and P. Narayanan. *Accelerating large graph algorithms on the GPU using CUDA.* In Proc. High Performance Computing, pp. 197âĂŞ208, 2007.

[21] U. Meyer and P. Sanders. *Delta-stepping: A parallel single source shortest path algorithm.* In Proc. Annual European Symposium on Algorithms, pp. 393âĂŞ404, 1998.

[22] Y. Deng and S. Mu. *Electronic design automation with graphic processors: a survey.* In Proc. Foundations and Trends in Electronic Design Automation, 2013.

[23] A. Bleiweiss. *GPU accelerated pathfinding.* In Proc. Symposium on Graphics Hardware, pp. 65-74, 2008.

[24] Y. Han, K. Chakraborty, and S. Roy. *A global router on GPU architecture.* In Proc. IEEE International Conference on Computer Design, 2013.

[25] F. Busato and N. Bombieri. *An Efficient Implementation of the Bellman-Ford Algorithm for Kepler GPU Architectures.* In Proc. IEEE Transactions on Parallel and Distributed Systems, 2016.

[26] A. Davidson, S. Baxter, M. Garland, and J. Owens. *Work-efficient parallel GPU methods for single-source shortest paths.* In Proc. IEEE 28th International Parallel and Distributed Processing Symposium, 2014.

[27] A. DeHon et al. *GraphStep: A system architecture for sparse-graph algorithm.* In Proc. Field-Programmable Custom Computing Machines, 2006.

[28] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs.* In Proc. Springer, ISBN 0-7923-8460-1, February 1999.

[29] M. Burtscher, R. Nasre, and K. Pingali. *A quantitative study of irregular programs on GPUs.* In Proc. Int. Sym. on Workload Characterization, 2012.

[30] K. Pingali et al. *The tao of parallelism in algorithms.* In Proc. Programming Language Design and Implementation, 2011.

[31] D. Merrill, M. Garland, and A. Grimshaw. *Scalable GPU graph traversal.* In Proc. Principles and Practice of Parallel Programming, 2012.

[32] P. Chan, M. Schlag, C. Ebeling, and L. McMurchie. *Distributed-memory parallel routing for field-programmable gate arrays.* In Proc. IEEE TCAD, 19(8):850âĂŞ862, 2000.

[33] R. Rubin and A. Dehon. *Timing-driven pathfinder pathology and remediation:quantifying and reducing delays noise in VPR-pathfinder.* In Proc. Field Programmable Gate Arrays, 2011.

[34] A. Dehon, R. Huang, and J. Wawrzynek. *Hardware-assisted fast routing.* In Proc. Field-Programmable Custom Computing Machines, 2002.

[35] S. Zhou, C. Chelmis, and V. Prasanna. *Accelerating large-scale single-source shortest path on FPGA.* In Proc. Parallel and Distributed Processing Symposium Workshop, 2015.

[36] A. Dandalis, A. Mei, and V. Prasanna. *Domain specific mapping for solving graph problems on reconfigurable devices.* In Proc. Parallel and Distributed Processing, 1999.

[37] R. Sedgewick and K. Wayne. *Algorithms* In Proc, Addison-Wesley, 4th edition, ISBN: 032157351X, 2011.

[38] J. Rose et al. *VPR 7.0: Next generation architecture and CAD system for FPGAs.* In Proc. ACM Trans. on RETS, 2014.

[39] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/alanmi/abc/.