

RouterAcc: FPGA Acceleration for VLSI Detailed Router via Hierarchical Storage Mapping

Ruiyuan Guo^{1,†}, Zexu Zhang^{2,†}, Chang Liu¹, Da Tang¹, Weiqi Shen¹, Haodong Lu^{1,*}, Xiqiong Bai³,
Kun Wang^{1,*}, Jianli Chen¹, and Jun Yu¹

¹State Key Lab of Integrated Chips & Systems, and School of Microelectronics, Fudan University, Shanghai, China

²College of Computer and Big Data, Fuzhou University, Fuzhou, China

³School of Integrated Circuit Science and Engineering, Nanjing University of Posts and Telecommunications, Nanjing, China
Email: 24112020080@m.fudan.edu.cn and kun.wang@ieee.org

Abstract—Detailed routing constitutes a critical phase in the very large-scale integration (VLSI) physical design, widely regarded as the most time-consuming and computationally intensive step in the back-end design process. Due to its iterative nature and strong data dependencies, conventional parallel acceleration techniques often suffer from limited scalability and effectiveness. To address these challenges, we propose RouterAcc, an FPGA-based software–hardware co-design acceleration framework tailored for VLSI detailed routing. RouterAcc incorporates an access analysis mechanism and a termination condition strategy to accelerate convergence. Furthermore, we employ a hierarchical storage mapping scheme and a flexible dimension-partitioning architecture to alleviate memory bottlenecks and enhance data locality. Additionally, RouterAcc leverages a hierarchical comparison pipeline with fully parallelized computing units and a data preprocessing strategy to maximize computational efficiency. Experimental results on the ISPD’18 benchmarks demonstrate that RouterAcc achieves consistent speedups of $2.1\times$ – $2.3\times$ over TritonRoute with less than 1% quality degradation. With further co-optimization, RouterAcc attains speedups of $2.7\times$ – $11.8\times$ while maintaining routing quality comparable to TritonRoute and surpassing Dr.CU 2.0 as well as the state-of-the-art (SOTA) FPGA-based approaches.

I. INTRODUCTION

In the workflow of integrated circuit (IC) physical design, routing constitutes a critical stage, establishing functional electrical connections among components while adhering to strict manufacturability constraints. Given the increasing complexity of modern design, the routing process is conventionally divided into two phases: global routing, which provides coarse-grained path guidance, and detailed routing, which executes fine-grained wire and via assignments under stringent design rules. While global routing has been extensively studied with optimization strategies to constrain the search space and reduce computational overhead [1]–[5], detailed routing remains a major bottleneck [6]. This phase typically consumes the majority of total runtime due to the complexity of path exploration, parasitic resistance-capacitance (RC) delay modeling, crosstalk analysis, and frequent rule-checking requirements.

Accelerating detailed routing is particularly challenging due to iterative dependencies and memory-intensive algorithms, which make traditional parallel computing approaches ineffective. Existing algorithms, *e.g.*, TritonRoute [7] and Dr.CU 2.0 [8], face significant performance bottlenecks due to

irregular memory access patterns, especially within the maze routing kernel built upon A* or Dijkstra search. A primary bottleneck lies in the management of the open list, closed list, and priority queue during the search process. Operations on the open list dominate execution time in single-threaded implementations [9], and parallelization efforts are frequently hindered by the fundamentally memory-bound characteristics of the algorithm.

To accelerate routing algorithms, prior work has explored heterogeneous platforms such as GPUs. However, in VLSI design, GPU acceleration has been applied primarily to global routing [2] [3] [10] [11], as detailed routing presents greater challenges regarding problem sizes and rule complexity. Recently, Field-Programmable Gate Arrays (FPGAs) have emerged as a promising platform for detailed routing acceleration [12], [13]. Unlike CPUs and GPUs, FPGAs offer abundant on-chip memory and fine-grained parallelism, which are critical for reducing memory latency and enabling efficient large-scale data interactions. By leveraging on-chip cache, FPGAs can mitigate the bottleneck of external memory access, thereby reducing both latency and power consumption. Several studies have investigated FPGA acceleration for detailed routing. For example, [13] introduced predictive prefetching and crossbar-based parallel cost calculation, achieving over $2\times$ speedup with minimal quality degradation. However, this approach did not fully exploit FPGA programmability and customizability to solve the fundamental memory bottleneck of dynamic lists.

In this paper, we propose RouterAcc, a software–hardware co-design framework for FPGA-accelerated detailed routing. RouterAcc integrates pin accessibility analysis and an intelligent termination mechanism to reduce redundant rip-up-and-reroute iterations. To alleviate memory access bottlenecks, we introduce a memory-efficient architecture with hierarchical storage mapping and reconfigurable dimension partitioning. The main contributions are summarized as follows:

- We propose RouterAcc, an FPGA-based software–hardware co-design acceleration framework tailored for the detailed routing phase of VLSI physical design.
- We integrate pin access analysis and present a termination condition to prevent redundant rip-up-and-reroute iterations, and adapt the routing algorithm to FPGA hardware.
- We present hierarchical storage mapping and flexible dimension partition architecture to overcome the detailed

*Corresponding authors. †These authors contributed equally to this work.

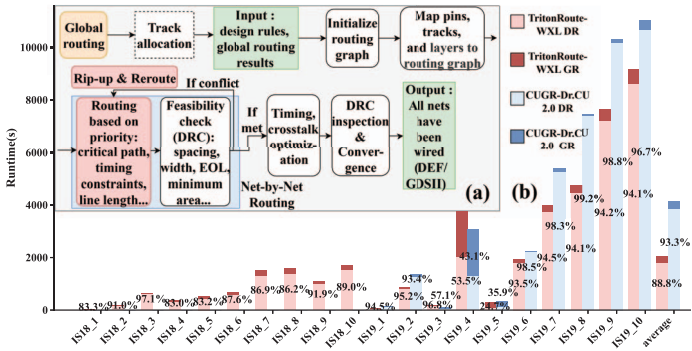


Fig. 1: (a) Classic industrial detailed routing flow. (b) Runtime proportion of global routing and detailed routing throughout the end-to-end routing process.

routing acceleration bottleneck. Moreover, we introduce a hierarchical comparison pipeline and fully parallelized computing units to improve computational efficiency.

- We demonstrate that RouterAcc achieves consistent $2.1\times\text{--}2.3\times$ acceleration of TritonRoute with negligible quality loss. In addition, a co-optimization strategy further improves performance to $2.7\times\text{--}11.8\times$, delivering quality comparable to TritonRoute while outperforming Dr.CU 2.0 and prior FPGA accelerators.

II. BACKGROUND AND MOTIVATION

A. Detailed Routing Flow

Detailed routing is the final routing stage that converts global-routing guides into DRC-clean geometric wires and vias on specific metal layers and tracks. The flow, as Fig. 1(a) shows, typically starts with technology-driven resource modeling, including track grids, layer directions, and blockage or keepout mapping. Next, pin-access analysis generates legal access points and resolves local infeasibilities near standard-cell pins and macros. Nets are then prioritized (e.g., clocks and timing-critical or high-fanout nets) and routed using fast pattern routing followed by maze search when needed, with cost terms for wirelength, vias, congestion, and timing or SI weights. Conflicts and violations are mitigated through iterative rip-up-and-reroute, often guided by negotiated congestion and history costs to promote convergence. A dedicated repair phase fixes residual DRC issues such as spacing, EOL, minimum area, via enclosure, antenna, and multi-patterning constraints via local detours, layer swaps, and patching. Finally, post-route optimization inserts redundant vias, applies NDRs, spacing or shielding for SI, and hands the result to signoff DRC and PEX.

B. Bottleneck

1) *Urgency of Accelerating Detailed Layout:* In advanced full-process global-detailed routing (GR-DR) flows, e.g., TritonRoute-WXL [6] and the CUGR [5]-and-Dr.CU 2.0 [8] flow, detailed routing accounts for approximately 90% of the total runtime, while global routing contributes only about 10%, as shown in Fig. 1(b). This statistic is based on the acceleration of detailed routing using CPU multi-threading. Under standard single-threaded operations, the proportion of

detailed routing is higher. Due to its vast path search space, iterative operations, and frequent design rule checks on high-resolution databases, detailed routing dominates runtime. While global routing acceleration has been extensively studied [1], [2], [4], [11], detailed routing remains difficult to accelerate because of its strong data dependencies.

2) *Memory Access Bottleneck:* Classical detailed routers rely primarily on two data structures: the open list and the closed list [14]. The closed list, typically implemented as a linked hash table, prevents redundant state expansion, while the open list, generally managed via a binary heap, prioritizes states using heuristic cost values. As reported in [9], operations on the open list account for approximately 95% of execution time in single-threaded implementations, and even with parallelization across eight threads, this proportion only decreases to 88%. The persistent dominance of memory operations highlights a fundamental architectural limitation, i.e., computational optimizations cannot overcome bottlenecks arising from memory-bound execution and inefficient resource utilization. This challenge is further exacerbated in complex routing with high obstacle density, where frequent cache misses significantly degrade performance predictability and efficiency.

C. Motivation

Recent studies have explored FPGA acceleration for detailed routing [12] [13], employing predictive prefetching, parallel computing units, and crossbar structures to achieve over $2\times$ speedup with minimal quality degradation. However, such straightforward parallel designs have not fully exploited the programmability and customizability of FPGAs. By leveraging these features together with real-time pipeline operations, it becomes possible to tailor and optimize back-end detailed routing algorithms and flows, thereby solving the fundamental memory access bottleneck problem, unlocking greater acceleration potential.

III. ALGORITHM OPTIMIZATION

A. Data Structure on FPGA

In CPU-based maze routing, data structures are inefficient for FPGA accelerators. To address this, we redesign the data layout. As shown in Fig. 2(a), the on-chip RAM stores compact node information, enabling fast access for parallel expansion. Fig. 2(b) shows the DDR organization, where grid groups record the coordinates and state of nodes, allowing efficient large-scale data access and supporting fine-grained parallelism.

B. Detailed Routing Flow Optimization

The overall framework of our detailed routing algorithm is presented in Algorithm 1 and Fig. 3(a). The routing flow begins with database initialization on the CPU, followed by the construction of the grid structure (GCells) [16] and pin accessibility analysis [6]. Pre-routed nets are initialized, and pin access points are collected to facilitate subsequent routing operations (Line 1). During the rip-up-and-reroute stage (Lines 2-15), the CPU sends batches of routing instructions to the FPGA, which executes parallelized search operations

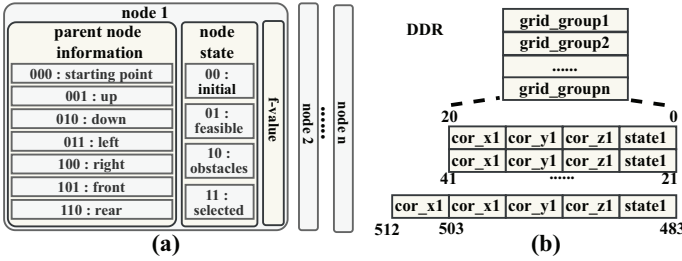


Fig. 2: FPGA data structure. (a) Information stored in a RAM block. (b) Data organization in DDR.

Algorithm 1 Iterative Optimization of Detailed Routing

Input: benchmark files
Output: full connected nets

- 1: InitDb(), InitPreRouteNet(), PinAccessAnalysis()
- 2: **for each** SearchRepair stage **do**
- 3: **Select** all *drcNets* from *nets*
- 4: **Partition** *drcNets* into batches $\{batch_1, \dots, batch_n\}$
- 5: **for** $net \in batch_i$ **in Parallel do** \triangleright per search parallel
- 6: $ucPins \leftarrow GetPins(net)$
- 7: **while** $ucPins \neq \emptyset$ **do**
- 8: $tpin \leftarrow \arg \max_{pin \in ucPins} pin.cost$
- 9: $cPins \leftarrow cPins \cup tPin, ucPins \leftarrow ucPins \setminus tPin$
- 10: $G \leftarrow PrepareGraph(cPins, ucPins)$
- 11: FPGA-OptimizedRoutingKernel(G)
- 12: **end while**
- 13: UpdateDRC(), UpdateNets()
- 14: **end for**
- 15: **end for**
- 16: UpdateDb()

and returns results to the CPU via the top-level module. This iterative process progressively reduces design rule violations (DRCs) and converges toward an optimized detailed routing solution. To further improve efficiency, we integrate the state-of-the-art pin access analysis [15], which jointly accounts for track resources, pin accessibility, and design rules, thereby enhancing congestion estimation and layer allocation. In addition, when the DRC count is already low or its reduction stagnates, repeated iterations tend to re-examine only a small number of violations while incurring up to 50% runtime overhead. To avoid unnecessary computation, we introduce an early-termination condition to prevent redundant iterations. For each main iteration loop (Lines 2-14), nets with unresolved violations are divided into batches and routed in parallel. Within the inner loop (Lines 7-12), the algorithm repeatedly selects the pin with the highest routing cost from the set of unconnected pins. The FPGA-optimized routing kernel (Line 11) computes paths between pins while updating the routing graph G (Line 10), which maintains connectivity and cost metrics. The routing graph is also designed to capture routing stats, including the grid graph, source pin, connected components, and tracking metrics (pathData, perTime, and stepCnt). Once all SearchRepairs

Algorithm 2 FPGA-Optimized Routing Kernel Algorithm

Input: graph batch G_k
Output: connected nodes set

- 1: **Init** Hierarchical RAM of existing nodes
- 2: **while** $node \in$ feasible nodes **do**
- 3: Update $node$ information in RAM
- 4: $p_nodes \leftarrow$ **Expand** $node$ in six directions
- 5: Upload $node \rightarrow$ astar_top
- 6: **for** $n \in p_nodes$ **in Parallel do** \triangleright per direction parallel
- 7: **if** $v \in G_k.end_points$ **then**
- 8: Upload $p_node \rightarrow$ astar_top **and Exit**
- 9: **end if**
- 10: Calculate f_cost_value and g_cost_value
- 11: Compare intra_RAM $nodes$ cost_min
- 12: Update n information intra_RAM
- 13: **end for**
- 14: Compare inter_RAM cost_min
- 15: **end while**

stages are completed or convergence is reached, the final results are committed, and the routing process is terminated.

C. FPGA-Optimized Routing Kernel Algorithm Flow

The conventional maze routing algorithm is adapted to the characteristics of the FPGA, as illustrated in Algorithm 2. The grid map is first initialized on the CPU, and then the data is transmitted to the DDR on the FPGA for storage. The routing kernel begins by initializing the hierarchical RAM with node data. From each node, potential routing paths are expanded in six directions, *i.e.*, up, down, left, right, forward, and rear (Lines 2-5). These six expanded nodes are processed simultaneously within a computation module that supports six degrees of parallelism (Lines 6-13). If a target endpoint is reached during the expansion (Lines 7-9), the corresponding node is immediately passed to astar_top, and the search for that net is terminated. For each neighbor node, the algorithm evaluates the heuristic cost (Line 10) and performs a two-stage cost comparison. 1) Intra-RAM comparison (Line 11): row-level comparators within each storage unit first identify local minima, and a depth-first tree arbiter then selects the optimal candidate. 2) Inter-RAM comparison (Line 14): the global minimum across RAMs is chosen. If the updated cost is smaller than the previous result from the binary-tree comparator, it is directly selected; otherwise, the prior node is retained and the new value is inserted for a partial tree update. Finally, the updated node states are written back to RAM (Line 12), maintaining the address-to-coordinate mapping required to sustain hardware-based data flow.

IV. HARDWARE ARCHITECTURE

A. Overview of RouterAcc Framework

An overview of the CPU-FPGA architecture of RouterAcc is shown in Fig. 3. The CPU dispatches batches of routing instructions to the FPGA, which executes parallelized search operations and returns results via the top-level module. Before

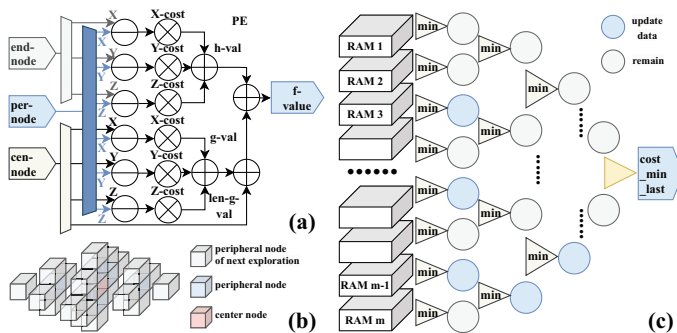


Fig. 5: (a) The PE of `cost_calculator` module. (b) The memory manager prefetches from DDR. (c) Workflow of the `cost_compare` module.

feasibility (01), a hardware comparator updates the entry only when the new f -value improves path quality. This state-driven conditional write strategy reduces redundant memory access and overwriting, while ensuring preservation of path optimality.

C. Hardware-Optimized Execution on FPGAs

Algorithm 2 and Fig. 3(b) illustrate the detailed routing algorithm adapted for FPGA-based hardware-optimized execution.

1) *Low-Latency Cost Calculation*: Each routing direction is computed in parallel by separate processing elements (PEs), as shown in Fig. 5(a), thereby accelerating wavefront expansion. Within the `Data_ctr` module, peripheral nodes in six directions are prefetched together with the nodes of the next iteration, as shown in Fig. 5(b). This pipelined prefetching ensures that the required data is already available during computation, effectively hiding memory access latency.

2) *Hierarchical Comparison Pipeline Architecture*: In distributed storage architectures, selection of the least-cost node is achieved through a hierarchical self-updating comparison mechanism. Each storage unit integrates a hardware minimum-value tracker, which triggers parallel comparison whenever a new f -value is detected. Register groups are employed to dynamically update the minimum cost and corresponding 3D coordinates, while a valid status flag is sent to the arbitration controller. As illustrated in Fig. 5(c), a double-buffer register is employed to decouple computing and storage units, ensuring continuous pipeline operation during data updates. This distributed arbitration mechanism transforms exhaustive search into multi-level pipelining, mitigating resource contention in centralized comparison and enabling real-time path planning.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

Hardware Implementation. The acceleration framework is implemented on a Linux system with an 8-core Intel(R) Xeon(R) Gold 5218R CPU@2.10GHz with 64 GB memory, and a Xilinx XCVU9P FPGA with 1.18M LUTs, 2.36M FlipFlops, 2.16K Block RAMs, and 960 Ultra RAMs.

Benchmarks. We evaluate the performance of our framework using the ISPD’18 detailed routing benchmarks [17]. To ensure fairness, TritonRoute [7] and Dr.CU 2.0 [8] are re-executed

on our server, and their detailed routing kernel runtimes are recorded. We also report the performance of the state-of-the-art FPGA-based accelerator in [13]. Since its implementation is not open source, we could not re-evaluate it on our platform. Therefore, we compare acceleration ratios rather than raw runtimes to avoid bias caused by differences in CPU platforms.

As shown in Fig. 6(a), the runtime of the batched maze routing kernel with 8 CPU threads approaches the peak performance of a multicore CPU. Accordingly, we test TritonRoute and Dr.CU 2.0 with 8 threads on our CPU server and configure 8 parallel pipelines on the FPGA for acceleration. It is worth noting that [13] adopted 12 PEs for acceleration, while also using the 8-thread CPU runtime as the baseline.

B. Acceleration Efficiency

Table I summarizes the performance comparison among TritonRoute (TR) [7], Dr.CU 2.0 (CU) [8], the SOTA FPGA-based accelerator ([13]-CU), our FPGA-accelerated TritonRoute(RA-TR), and our software-hardware co-optimized framework (CoRA-TR), which first optimizes the detailed routing algorithm and then performs hardware acceleration. ISPD’18 quality scores indicate that TR generally achieves superior routing quality compared to CU and its accelerated variants, consistent with using multiple rip-up-and-reroute iterations to refine solutions. Our FPGA-accelerated version, RA-TR, maintains quality close to TR, with variations typically within 1%, attributed to minor precision loss during hardware execution. CoRA-TR, which further integrates pin access analysis, achieves the best overall results, maintaining quality comparable to TR across different benchmarks. After accelerating the routing process, our results outperform [13]-CU in routing quality on most test cases.

Under 8-thread CPU parallelism, TritonRoute achieves strong performance. We accordingly configured our FPGA accelerator with eight parallel pipelines for fair comparison. This setting yields a stable acceleration of 2.1-2.3 \times across all test cases, with negligible quality degradation. Given that the CPU frequency (2.10 GHz) is approximately 14 \times higher than that of our FPGA (150 MHz), the obtained acceleration clearly demonstrates the efficiency of our design. The iteration count of TritonRoute is approximately 3 \times that of Dr.CU 2.0 [8]. To further improve acceleration, we incorporate pin access analysis from TritonRoute-WXL [6] to optimize the selection of source and sink nodes. Through such software-hardware co-optimization, CoRA-TR surpasses Dr.CU 2.0 in speed, achieving additional improvements in acceleration ratio. Although our exploration iterations remain nearly twice those of Dr.CU, the vast majority of cases still outperform [13], achieving 2.7 \times -11.8 \times speedups. For comparison, we chose to conduct the test on the same ISPD’18 test sets as [13]-CU, which is on a smaller scale. On larger-scale test sets such as ISPD’19, our advantages would be more obvious.

C. Scalability Exploration of Multiple PEs

As shown in Fig. 6(a), the average runtime using 8 threads approaches the peak performance of the multi-core CPU. On the FPGA, performance scales nearly linearly as the number of

TABLE I: Performance comparison of TritonRoute [7], Dr.CU 2.0 [8], and our FPGA-accelerated framework on the batched maze routing kernel using the ISPD 2018 [17] contest benchmarks.

Benchmark	ISPD'18 Quality Score [†]					Detailed Routing Kernel (8 thread*)						
						Runtime(s)				Speed-up		
	TR	CU	[13]-CU	RA-TR	CoRA-TR	TR	CU	RA-TR	CoRA-TR	[13]-CU	RA-TR	CoRA-TR
ISPD18_test1	304934	291305	290099	303483	307257	4.41	1.59	2.03	1.60	2.0×	2.2×	2.7×
ISPD18_test2	4831983	4700581	4685023	4810886	4819019	58.86	18.57	25.93	18.68	2.2×	2.3×	3.1×
ISPD18_test3	5283329	5295331	5291929	5281145	5332504	110.07	25.60	52.40	52.93	2.1×	2.1×	2.1×
ISPD18_test4	15035816	15761114	15808947	15082968	14957868	244.48	120.63	106.29	20.56	2.4×	2.3×	11.8×
ISPD18_test5	16077963	16370540	16406741	16103496	15971928	109.42	86.68	40.39	25.20	2.7×	2.7×	4.3×
ISPD18_test6	21352959	21636718	21666362	21367147	21083855	165.39	125.90	77.33	45.36	2.7×	2.1×	3.6×
ISPD18_test7	38148393	38408621	38523171	38206329	37867997	246.12	234.84	111.98	88.77	2.8×	2.2×	2.8×
ISPD18_test8	38307564	38571533	38717711	38346227	38207086	256.73	212.65	125.29	94.63	2.8×	2.1×	2.7×
ISPD18_test9	32925038	33114610	33342024	33067470	32805346	264.21	158.15	128.19	83.10	2.9×	2.1×	3.1×
ISPD18_test10	40403574	40998097	42820500	40528793	41254471	380.69	345.35	180.45	189.81	2.7×	2.1×	2.0×

* Except for [13]-CU, which adopts 12 degrees of parallelism, all other works adopt 8 degrees of parallelism.

[†] The ISPD'18 Quality Score evaluates routing solution quality, with a lower score indicating better performance, adjusted for penalties like non-determinism.

TABLE II: Hardware resource utilization.

Resource	Utilization	Available	Percent (%)
LUT	291736	1182240	24.67
FF	226588	2364480	9.58
BRAM	1200	2160	55.55
URAM	242	960	25.20

parallel PEs increases from 1 to 16. Furthermore, the speedup can be further enhanced with additional hardware resources, demonstrating the strong scalability of our framework on larger FPGA platforms.

D. Design Space Exploration of the RAM Number

Due to limited resources, we utilize 100 RAM matrices, each of size 10×10 , for the entire experiment. To investigate the impact of RAM quantity on acceleration, we execute the first 300 search processes of `ispd8_test1`. We collect FPGA simulation data, measure runtimes across different RAM configurations, and normalize the results by reporting runtime per 10,000 explored nodes. As shown in Fig. 6(b), the performance of our framework improves as the number of RAM blocks increases. When on-chip resources are sufficient, employing finer-grained RAM partitioning boosts the throughput of storage comparison and update operations, leading to greater acceleration. These results suggest that acceleration performance can be further improved on FPGA boards with more abundant resources. However, unlimited subdivision is impractical due to the inherent limits of on-chip resources. Once the partitioning granularity exceeds a certain threshold, the marginal performance gain diminishes. Therefore, selecting an appropriate RAM partitioning granularity according to the specific routing scale is essential to achieve strong acceleration while avoiding resource waste from excessive subdivision.

E. Consumption of Hardware Resource

Table II summarizes hardware utilization. RAM resources are maximized to construct larger matrix units, leading to relatively

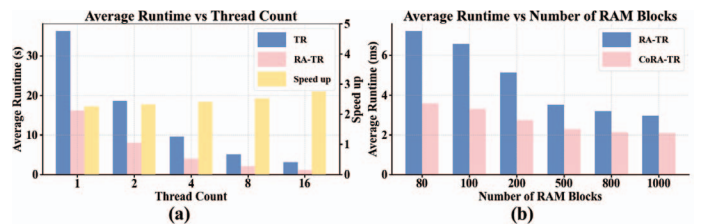


Fig. 6: Impact of RAM block partitioning and thread count on acceleration performance.

high memory utilization. Memory dominates detailed routing and becomes the primary bottleneck as benchmarks scale. By leveraging high on-chip bandwidth, FPGAs effectively reduce latency in data-intensive operations, making them a promising platform for routing acceleration.

VI. CONCLUSIONS

In this paper, we presented RouterAcc, an FPGA-based acceleration framework for detailed routing that addresses the memory access bottleneck through hierarchical storage mapping, state-driven updates, and a hierarchical comparison pipeline. On ISPD'18 benchmarks, RouterAcc achieves a consistent $2.1 \times - 2.3 \times$ speedup of TritonRoute with negligible quality loss, and up to $11.8 \times$ when co-optimized. Compared with Dr.CU 2.0 and prior FPGA accelerators, RouterAcc delivers both higher performance and competitive solution quality. These results demonstrate the feasibility and promise of FPGA-based acceleration for breaking the long-standing runtime bottleneck in detailed routing.

ACKNOWLEDGMENT

This work was financially supported by Project of Shanghai EC (24KXZNA12). The computations in this research were performed using the CFFF platform of Fudan University.

REFERENCES

- [1] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao, "Multi-threaded collision-aware global routing with bounded-length maze routing," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2010.
- [2] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao, "NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2010.
- [3] Y Han and K Chakraborty, and S Roy, "A global router on GPU architecture," in *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, 2013.
- [4] Jiayuan He, Martin Burtscher, Rajit Manohar, and Keshav Pingali, "SPRoute: A scalable parallel negotiation-based global router," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.
- [5] JinWei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline F. Y. Young, "CUGR: Detailed-routability-driven 3D global routing with probabilistic resource model," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2020.
- [6] Andrew B. Kahng, Lutong Wang, and Bangqi Xu, "TritonRoute-WXL: The open-source router with integrated DRC engine," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 4, pp. 1076-1089, April 2022.
- [7] Andrew B. Kahng, Lutong Wan, g and Bangqi Xu, "TritonRoute: The open-source detailed router," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 3, pp. 547-559, March 2021.
- [8] Haocheng Li, Gengjie Chen, Bentian Jiang, Jingsong Chen, and Evangeline F. Y. Young, "Dr. CU 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.
- [9] Yuzhi Zhou, Xi Jin, and Tianqi Wang, "FPGA implementation of A* algorithm for real-time path planning," in *International Journal of Reconfigurable Computing*, 2020, 1-11.
- [10] Han, Y., Ancajas, D. M., Chakraborty, K., and Roy, S, "Exploring high-throughput computing paradigm for global routing," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 155-167, Jan. 2014.
- [11] Shiju Lin, Jinwei Liu, Evangeline F. Y. Young, and Martin D. F. Wong, "GAMER: GPU-accelerated maze routing," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 2, pp. 583-593, Feb. 2023.
- [12] Korolija, Dario, and Mirjana Stojilović, "FPGA-assisted deterministic routing for FPGAs," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019.
- [13] Xun Jiang, Jiarui Wang, Yibo Lin, and Zhongfeng Wang, "FPGA-accelerated maze routing kernel for VLSI designs," in *Proceedings of the IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022.
- [14] Adham Osama, Ahmed Mostafa, Eslam Mamdouh, Mohamed Gamal, Usama Imam, Mohamed Taha, Ahmed Khalil, Islam Ahmed, and Hassan Mostafa, "Fast RTL implementation of A* path planning algorithm," in *Proceedings of the International Conference on Microelectronics (ICM)*, 2021.
- [15] Andrew B. Kahng, Lutong Wang, and Bangqi Xu, "The Tao of PAO: Anatomy of a pin access oracle for detailed routing," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2020.
- [16] Dolgov, Sergei, et al, "2019 cad contest: Lef/def based global routing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.
- [17] Stefanus Mantik, Gracieli Posser, Wing-Kai Chow, Yixiao Ding, and Wen-Hao Liu, "ISPD 2018 initial detailed routing contest and benchmarks," in *Proceedings of the International Symposium on Physical Design (ISPD)*, 2018.