electronics

MDPI

Article

A Field Programmable Gate Array Placement Methodology for Netlist-Level Circuits with GPU Acceleration

Meng Liu *, Yunfei Wang and Shuai Li

Faculty of Information Technology, Beijing University of Technology, Beijing 100025, China; shuai@emails.bjut.edu.cn (S.L.)
* Correspondence: liumeng@bjut.edu.cn

Abstract: Field Programmable Gate Arrays (FPGAs), renowned for their reconfigurable nature, offer unmatched flexibility and cost-effectiveness in engineering experimentation. They stand as the quintessential platform for hardware acceleration and prototype validation. With the increasing ubiquity of FPGA chips and the escalating scale of system designs, the significance of their accompanying Electronic Design Automation (EDA) tools has never been more pronounced. The placement process, serving as the linchpin in FPGA EDA, directly influences FPGA development and operational efficiency. This paper introduces an FPGA placement methodology hinging on the Verilog-to-Routing (VTR) framework. We introduce a novel packing approach grounded in the weighted Edmonds' Blossom algorithm, ensuring that the CLB generation strategy aligns more closely with load-balanced distribution. Furthermore, we enhanced the electric field-driven resolver placement process for CLB locations and leverage GPU-accelerated design. Experimental results demonstrate substantial improvements over the traditional VTR algorithm, with an average optimization of 28.42% in the packing process runtime, an average acceleration ratio of 2.85 times in the placement phase, and a 39.97% reduction in total packing and placement runtime consumption.

Keywords: FPGA; EDA; packing; placement; GPU acceleration; methodology

Citation: Liu, M.; Wang, Y.; Li, S. A Field Programmable Gate Array Placement Methodology for Netlist-Level Circuits with GPU Acceleration. *Electronics* **2024**, *13*, 37. https://doi.org/10.3390/electronics13010037

Academic Editor: Alexander Barkalov

Received: 23 November 2023
Revised: 19 December 2023
Accepted: 19 December 2023
Published: 20 December 2023

1. Introduction

In recent years, the rapid growth of emerging applications, including the Internet of Things, intelligent computing, and medical electronics, has driven increased market demand for application-specific integrated circuits (ASICs) [1–3]. Field Programmable Gate Arrays (FPGAs), known for their reconfigurable nature, offer flexible configuration and cost-effective engineering experimentation. They stand as an ideal development platform for hardware acceleration applications and prototype validation [4,5]. The numerous advantages mentioned above have led to their widespread adoption. The design of FPGAs heavily relies on computer-aided tools (EDA). Within the entire FPGA EDA process, placement represents a pivotal and computationally intensive phase. This step determines the physical placement of various instance units in the design, directly impacting FPGA development and utilization efficiency.

FPGA placement is an NP (nondeterministic polynomial time)-complete combinatorial optimization problem. Early research in FPGA placement in both industry and academia often relied on heuristic algorithms based on simulated annealing [6,7], with a notable example being the Versatile Place and Route (VPR) tool [6,8]. These approaches generated initial solutions in a random manner and perturbed the current solution through global or local exchanges, substitutions, movements, duplications, and other operations to generate new solutions, while these methods can yield relatively desirable placement results, they are associated with longer execution times when dealing with large-scale circuit placement problems [9]. In the subsequent development, both industry and academia transitioned towards minimum-cut algorithms for placement research, with a prominent example being

PPFF [10]. These methods employ a recursive netlist partitioning approach, subject to specific constraints, iteratively breaking down the larger placement problem into smaller sub-placement problems. This step-wise process reduces the computational complexity of circuit design but may introduce delays in critical paths. By incorporating graph algorithms and hypergraph partitioning concepts [11,12], minimum-cut algorithms can handle designs comprising thousands of gate units. However, as the gate counts of FPGAs have grown to several million, analytical methods have consistently outperformed the minimum-cut techniques in terms of runtime efficiency and the quality of physical designs. In contrast to the methods mentioned above, analytical techniques describe the placement problem as a more intricate optimization challenge and leverage a diverse set of gradient descent methods for solving it. Analytical methods can be further categorized into quadratic methods [13,14] and non-linear methods [15–17]. Quadratic solving entails objective functions composed of quadratic polynomials. Optimal placement results are obtained by solving equations where the first-order derivatives equal zero. Non-linear methods, conversely, use objective functions based on higher-order non-linear equations. Prominent examples of analytical methods include QPF [18], CAPRI [19], StarPlace [20], UTplaceF [14], elfPlace [16], and commercial FPGA placement solvers. Presently, commercial FPGA placement solutions predominantly rely on analytical methods and are recognized for their superior overall performance. In recent years, within the realm of large-scale digital integrated circuit placement research, placement tools based on machine learning methods have begun to emerge [21,22]. These include Google's EDA achievements based on reinforcement learning [23], the RLPlace placement tool based on reinforcement learning [24], and the DREAMPlace and DREAMPlaceFPGA frameworks with GPU acceleration [25,26]. However, at this stage, addressing the entire process from the circuit netlist to placement remains a challenging endeavor, and the exploration of FPGA placement using artificial intelligence methods is still in its nascent stages [21,27].

This paper presents a novel framework-integrated placement methodology for commercial FPGA processor chip platforms. Firstly, the proposed placement method covered all instance types, and the CLB generation process was constructed using a weighted Edmonds' Blossom algorithm. Secondly, GPU-based global placement was achieved for industrial-grade netlist circuits, and the entire packing and placement process was completed. Finally, placement experiments were conducted on general netlist circuits, validating the engineering application value of the proposed EDA methodology. Experimental results reveal an average 28.42% optimization in the packing process time, an average acceleration ratio of 2.85 times in the placement phase, and a 39.97% reduction in total packing and placement time compared with traditional VTR.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries of this work. Section 3 presents details of our methodology and algorithm details. Section 4 demonstrates the experimental results. Finally, Section 5 concludes the paper.

2. Related Work

FPGAs are programmable logic chips with a dynamically configurable logic circuit structure that offers increased flexibility and customizability for diverse applications. Despite being manufactured by different vendors, FPGA chips share a similar architecture, differing only in minor local hardware configurations. Figure 1 provides a consolidated overview of the FPGA placement and routing architecture, including the configurable logic block (CLB), digital signal processor (DSP), block random access memory (BRAM), buffer (BUF), and I/O (IO) components. The right side of the figure also depicts the internal structure of the CLB, which consists of several basic logic elements (BLEs), where BLEs are composed of look-up tables (LUTs) and flip-flops (FFs).

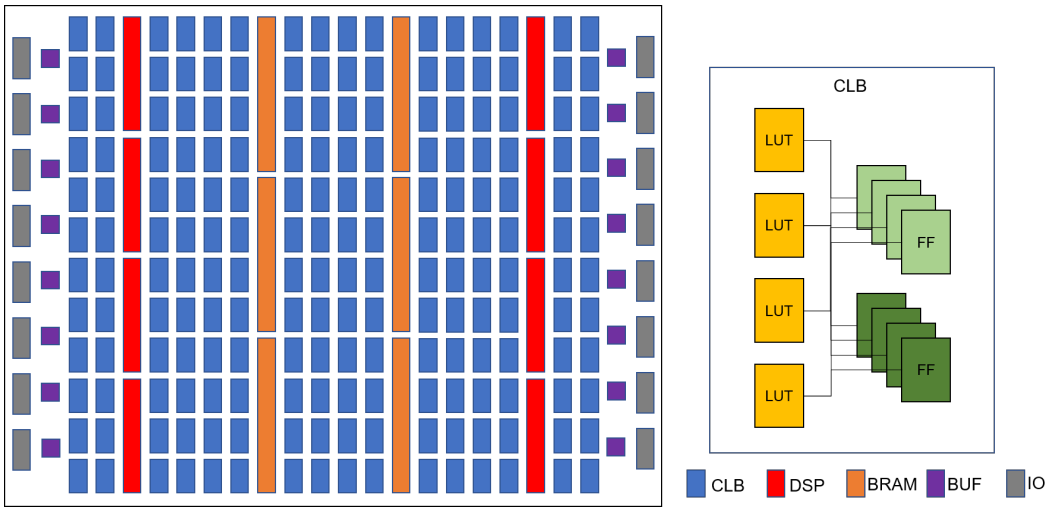

Figure 1. A representative FPGA placement architecture schematic.

VTR (Verilog to Routing) encompasses a suite of computer-aided design (CAD) tools adaptable to diverse FPGA architectures. Figure 2 outlines the tool process from netlist-level circuits to bitstream generation. Given that the quality of wiring results is predominantly influenced by the placement positions of all instances, the placement phase becomes a pivotal process impacting the entire FPGA EDA tool. Previous research efforts have typically addressed specific issues within the placement process for optimization. It receives the netlist file produced by the synthesis tool, carrying out packing, placement, and performance analysis operations. Eventually, it maps the bitstream file onto the FPGA. The placement process depends on the strategy outcomes of the packing phase. The FPGA packing phase needs to address the allocation of resources to be placed, including the usage of CLB resources and the allocation of resources within the CLB. Subsequently, a predefined number of BLEs are placed within a specific CLB, adhering to predetermined value constraints and optimization conditions. The internal configuration of the CLB remains generally unchanged throughout this process. In the VTR flow, the packaging process logically assigns the circuit netlist to the available resource blocks of the FPGA device, and the resulting information is saved to a .net file. This .net file comprehensively describes the CLB and its associated logic, incorporating details like interconnections, inputs, outputs, and clock signals. Upon completing the packing phase, a placement operation is executed on the circuit. The placement process allocates the resource blocks generated during packing to valid grid locations, optimizing paths and timings. This results in the output of a .place format description file, which details the physical location of all CLB instances. Despite the widely employed simulated annealing algorithm in the placement phase of VTR as a heuristic method, its essence lies in the iterative placement optimization of macrocell modules, resulting in a low granularity of the placement solution. Furthermore, the random placement algorithm for initialized placement struggles to guarantee the placement quality for large-scale circuits, with challenging control over the algorithm's convergence time. Confronting the characteristics and limitations of the VTR framework, this study embarked on the optimization of the packaged placement algorithm and methodology.

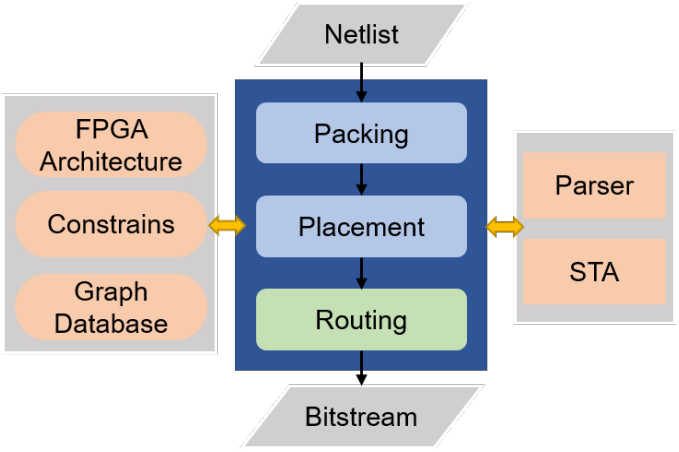

Figure 2. The traditional methodology of FPGA EDA. Among other aspects, the EDA database covers the FPGA architecture, constraint information, and graph database. Plug-in class tools include the parser and static timing analysis (STA) tool.

3. Approach

3.1. Proposed Methodology

The proposed methodology is illustrated in Figure 3. Specifically, a dedicated parser device is crafted for handling input data in eblif format netlists. Leveraging the VTR netlist read function, this parser extracts essential graph information from the netlist, subsequently integrating it into the optimized DREAMPlaceFPGA framework based on the bookshelf format data. To achieve compatibility with the standard netlist data format, we introduced example additions and modified data interfaces during parsing. The overall workflow involves executing a global placement task using GPU acceleration and seamlessly importing the results into the VTR database. A pivotal aspect of our approach is the introduction of a CLB packing process, grounded in Edmond's matching algorithm. This process was followed by the development of a CLB legalization process, informed by the outcomes of the packing strategy and global placement data. The results obtained from this comprehensive process serve as the initial state for our parsing method. Building upon this foundation, we employed a sophisticated parsing technique that not only reuses the placement results but also iteratively enhances them to yield refined CLB placement results. These results were then fed into the VTR router, completing the entire process. The proposed methodology integrates seamlessly into the design flow, optimizing the placement of CLBs within the FPGA architecture for enhanced performance and efficiency.

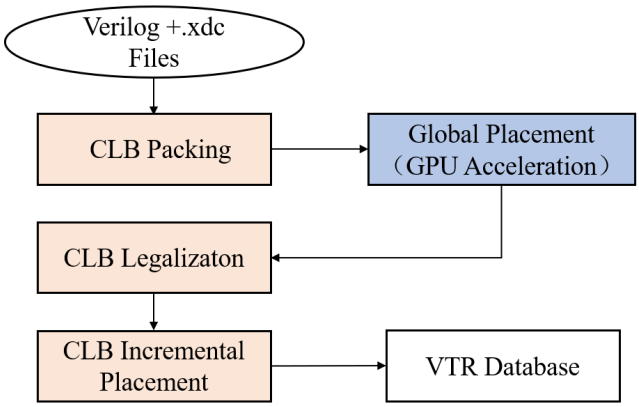

Figure 3. The proposed methodology of FPGA packing and placement.

3.2. Parser Design

A key element of the methodology integration process is the parser, and different database types are used between the VTR and GPU acceleration frameworks. Thus, the

custom database was designed to bridge the following gaps. GPU acceleration makes use of the bookshelf file interface format, which consists of the following files: *.lib* for instance unit properties (pin definitions and directions), *.node* for units to be placed and their types, *.net* for logical wire and unit graph relationships, *.scl* for logical resource distribution in the FPGA chip, and *.pl* for IO instance position information. The third-party parsing libraries Flex and Bison are used by the GPU acceleration framework to automatically generate grammar programs. The sections on declarations, rules, and C code comprise the core programs. The Flex and Bison libraries' source code was changed to adapt the lexicon rules for the synthesized netlist text [28]. For example, there is an instance called $techmap1163296$abc1140382lut$aiger1140381$99963 in the circuit netlist file. The *Bookshelf Scanner.ll* file's rule definition for STRING must be changed to [$A-Za-z][\ \ \ ][A-Za-z0-9_\ / \ [\]\-\$\.\:]* in order to parse it as bookshelf data.

3.3. CLB Packing Design

The VTR tool offers a comprehensive packing function based on a greedy algorithm and seed strategy, featuring intricate constraints to address considerations, making it well-suited for small-scale engineering scenarios. However, the current VTR packing and placement exhibit relative independence, particularly with the packing phase being time-consuming. This presents a significant opportunity for runtime optimization. To address this, our study focuses on refining the CLB merger generation strategy. We propose a CLB generation strategy based on Edmond matching, aiming to enhance the efficiency of the packing phase. The key adjustment involves conducting CLB clustering generation in strict accordance with logical relationships. By increasing the number of CLB clustering instances, we aim to not only improve packing time but also significantly increase the volume of placement tasks. The motivation behind this approach is to exploit GPU acceleration technology more effectively. This involves leveraging the parallel processing capabilities of GPUs to handle a larger number of placement tasks concurrently. Our strategy is illustrated in detail in Figures 4–6. The LUT and FF graphs are established using information from the static timing analysis report (Figure 4 illustrates the initial state relationship of the node graph, encompassing nodes 1–24, utilizing a directed graph to represent time paths between cells with node weights calibrated by static timing results). Subsequently, the hypergraph of the register group is derived through the HCS (Highly Connected Subgraph) algorithm [29], as depicted in Figure 5, forming clustering points H1(2,3), H2(13,14), and H3(18,19). This newly formed hypergraph enables the calculation of the evaluation function and constraint function. The evaluation function, in this study, is expressed as the sum of weights, while the constraint function considers the number of cells accommodated within the CLB and the total number of CLBs. Employing the Edmond maximum weight matching algorithm [30], clustering results were obtained to ensure relative balance within each CLB block. Figure 6 illustrates the formation of division regions, with each region capable of independently performing CLB generation.

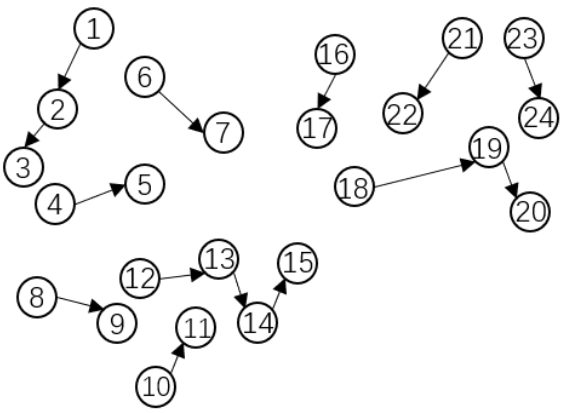

Figure 4. Several LUT and FF relationship graphs.

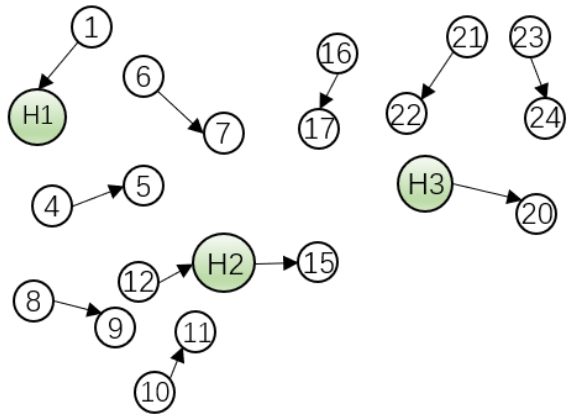

Figure 5. After HCS merging, several LUTs and FFs are hypergraphically interconnected.

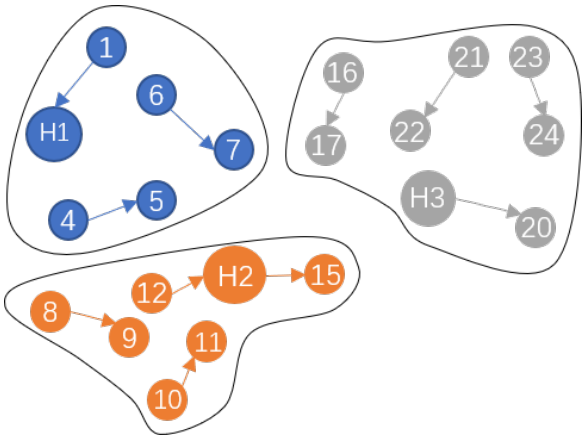

Figure 6. Schematic diagram illustrating the formation of CLB division regions.

We constructed an undirected weighted graph Node = (V; E). The pseudo-code of the weighted Edmonds' Blossom algorithm for CLB packing is summarized in Algorithm 1. Each connected subgraph is constructed by the HCS merging process. Nodes and edges are added into the subgraph in a breadth-first search manner. When the subgraph stops growing, the max-weighted matching is run on the constructed subgraph, and mergings corresponding to matched edges are committed to the graph. The loop of constructing subgraphs, solving max-weighted matching, and committing matched mergings is iterative and stops when no more merging can be performed.

Algorithm 1 Weighted Edmonds' Blossom Algorithm for CLB Packing

Require: CLB division regions are planned.
Ensure: LUTs and FFs relation graph built.
while ture **do**
  Initialize data structure.
  $num\_matching \longleftarrow 0$
  for $v \in V$ **do**
    HCS merging process.
    Edmonds' blossom matching process.
  end for
  if *num_matching* meet constraints **then**
    return
  end if
end while

In summary, our study introduces a novel CLB generation strategy based on Edmond matching, emphasizing the logical relationships between CLBs. This strategic adjustment

is anticipated to optimize the packing phase of VTR, leading to reduced runtime and improved efficiency. The proposed approach holds potential for unlocking the benefits of GPU acceleration technology, enabling more scalable and efficient placement in VTR for a wider range of engineering scenarios.

3.4. Improvements to the GPU Acceleration Framework

The DREAMPlaceFPGA framework is currently in its early stages, offering a significant advantage by presenting a generalized solution for GPU-based EDA problems, albeit primarily focused on global placement challenges [26]. The framework encompasses various operators within its op layer, including adjust_node_area, clustering_compatibility, dct, demandMap, density_map, pin_pos, pin_utilization, place_io, precondWL, rmst_wl, rudy, utility, and weighted_average_wirelength. To enhance its practical applicability, particularly for netlist circuits adapted to general-purpose use (refer to the parser section for specifics), additional considerations should be made to meet the resource requirements of the target FPGA chip. Specific instances on the DREAMPlaceFPGA side, such as the adder unit (CARRY) and data selector (MUX) for the developed FPGA chip, need supplementation in line with the resource demands of the adapted FPGA. The original framework leads to unclear goals of optimization because it considers too trivial instance placement. To optimize DREAMPlaceFPGA placement outcomes, a direct CLB placement method based on IO port anchors was designed. Given DREAMPlaceFPGA's flexibility regarding the number of instances for placement processing, a global placement approach is adopted for all instances, while other tasks are executed within the VTR framework. In the current phase, the DREAMPlaceFPGA framework functions as a subprocess in the methodology, playing a role akin to that of a chip coprocessor.

In our project, the global placement algorithm of the GPU acceleration framework is extracted as a sub-process within the fusion framework. The global placement method, as illustrated in Figure 7, initiates with a CLB initialized placement phase, distributing cells within the placement area's center and arranging them randomly. Subsequently, the core computation of global placement is seamlessly integrated into the deep learning framework. Equation (1) characterizes the classical optimization objective function for global placement, where $D(x, y)$ denotes the density statistics of instance objects at the position (x, y) and $We(x, y)$ represents the HPWL statistics of net e in the set of nets E. The unit density control factor is indicated by λ. A more sophisticated nonlinear global placement algorithm is presented in elfPlace [16]. The Weighted Average (WA) model can be introduced based on the definition of electrostatic placement. Equation (2) expresses the HPWL (Half-Perimeter Wirelength) statistics of net e, Equation (3) expresses the WA statistics in the x direction, and Equation (4) describes the overall WA wirelength statistics. The solution objective of the nonlinear placement problem is corrected based on the Augmented Lagrange Method (ALM) [31] by comparing the potential density of the electrostatic system and the WA wirelength model, as shown in Equation (5). Here, the device set $s \in S = \{CLB, IO, DSP, RAM\}$, and λ_s and ϕ_s indicate the density control factor and the expression for electrical energy, respectively. The expression for the electrical energy calculation can be based on the Poisson distribution for column writing, where c_s stands for the quadratic control factor. The α and β then denote the tuning factors to obtain more optimized results. The discrete trigonometric function transformation method is used to solve the electrostatic Poisson equation. The neural network training problem in Equation (1) is converted into a GPU-accelerated framework for FPGA placement. In order to create deep learning solution mapping, coordinate data y are used as the label quantity when coordinate data x are input into the neural network. Different device units can be regarded as distinct objects for placement calculations in the FPGA placement process. The overall score of the current distribution is derived from a combination of HPWL and timing estimation results. The iterative process is controlled by evaluating the incremental percentage, as illustrated in Figure 7. Upon successful DSP/RAM legalization verification, the CLB undergoes adjustment for legalized placement and subsequent incremental optimization.

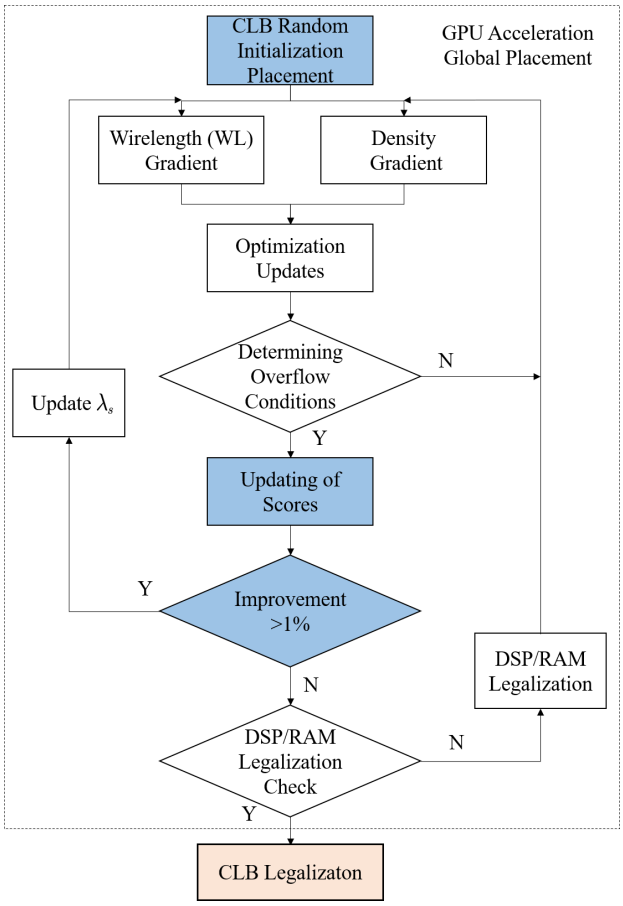

Figure 7. The GPU acceleration framework for CLB global placement based on DREAMPlaceFPGA.

$$\min_{x,y} f(x,y) = \lim(\sum_{e\in E} We(x,y) + \lambda D(x,y)) \tag{1}$$

$$W(x,y) = \sum_{e\in E}(\max_{i,j\in E}|x_i - x_j| + \max_{i,j\in E}|y_i - y_j|) \tag{2}$$

$$\widetilde{We}_x = \frac{\sum_{i\in e} x_i exp(\frac{x_i}{\gamma})}{\sum_{i\in e} exp(\frac{x_i}{\gamma})} - \frac{\sum_{i\in e} x_i exp(\frac{-x_i}{\gamma})}{\sum_{i\in e} exp(\frac{-x_i}{\gamma})} \tag{3}$$

$$\widetilde{W}(x,y) = \sum_{e\in E}(\widetilde{We}_x + \widetilde{We}_y) \tag{4}$$

$$\min_{x,y} f(x,y) = \alpha \cdot \widetilde{W}(x,y) + \beta \cdot \sum_{s\in S} \lambda_s(\phi_s(x,y) + \frac{c_s}{2}\phi_s(x,y)^2) \tag{5}$$

3.5. Incremental Optimization

CLB legalization is required based on the GPU-accelerated global placement results and the CLB clustering generation strategy. The goal is to give each CLB instance a reasonable location that is closest to the results of the GPU-accelerated global placement. Algorithm 2 presents the comprehensive algorithm design for the placement of CLB legalization. The main concept of discrete search in the surrounding area is incorporated into the overall algorithm. The discretized vector data type *bin_source*, which completely encapsulates the FPGA's physical resource information, is used to represent the layoutable resources in the GPU-accelerated global placement results. The legalized locations of CLB instances, indicated by the variable *clb_legal_location*, are the output data. In Step 1, instance unit weights are sorted by criticality based on CLB cluster/class score statistics and physical constraints; the array of CLBs pending legalization is denoted by *sort_clb*. The storage identifier in each CLB database is represented in Step 2 by *blk_id*. The horizontal

and vertical coordinates are calculated using the nearest neighbor algorithm, and the associated solving function is designated as *nearest_location*(*blk_id*). As explained in Algorithm Step 3, the computation results are kept in the *clb_location* structure database. Step 4 verifies the legality of *bin_source* and *clb_location* based on placement rules, in accordance with the patterns of physical resource allocation in domestic FPGAs. This is achieved by verifying whether the location is occupied by other instances, whether resource conflicts exist, and whether it corresponds with the physical characteristics of the corresponding instances. If the constraints are satisfied, the legal placement information can be output. Otherwise, incremental adjustments to the location coordinates in the neighboring region are made to achieve legal placement for CLBs.

Algorithm 2 CLB Legalization

Input: *global_location*, FPGA source, *bin_source*.
Output: CLB Legalization Location: *clb_legal_location*.
Instance unit weights sorted to form initial to-be-legalized array *sort_clb*.
for (*autoblk_id* : *sort_CLB*) **do**
  clb_location nearest_location(*blk_id*)
  if *bin_source*, *clb_lacation*, Legalization rule determination **then**
    push_back(*clb_location*)
  else
    Incremental adjustment of the calculated value of the coordinates.
  end if
end for

Based on the VTR API, an incremental placement is achieved by reducing the process iteration to optimize the placement quality. The specific algorithm design is illustrated in Algorithm 3, where the input is the legal CLB placement, *clb_legal_location*, and the output is the optimized CLB placement, denoted as *clb_optimized_location*. The VTR parsing API function *AP_run*( ) is utilized for calculations, considering constraints. The iteration count *N* is adjusted to control the runtime overhead of detailed placement, where HPWL needs to adhere to local optimal constraints, and timing results should meet design requirements.

Algorithm 3 CLB Incremental Placement

Input: CLB Optimized Location.
Output: CLB Legalization Location: *clb_legal_location*.
while Number of iterations *N* && HPWL local optimal solution && Timing result satisfied **do**
  AP_run( )
  Collect statistical HPWL and timing results.
  N- -
end while

4. Experimental Results

All experiments presented in this paper were conducted using the FPGA general-purpose interface development board. The FPGA chip model employed is part of the Zynq UltraScale+ series developed by Xilinx Corporation [32]. Figure 8 illustrates a schematic diagram of the development board along with the corresponding FPGA chip sample. The XCZU7EV-FFVC1156AAZ is fabricated using the 16 nm FinFET process node and boasts specifications including 504K logic cells, 230.4K LUTs, 1728 DSPs, and 11Mb BRAM. It adopts a stacked modularized placement, reinforcing the hierarchical characteristics of the overall architectural design.

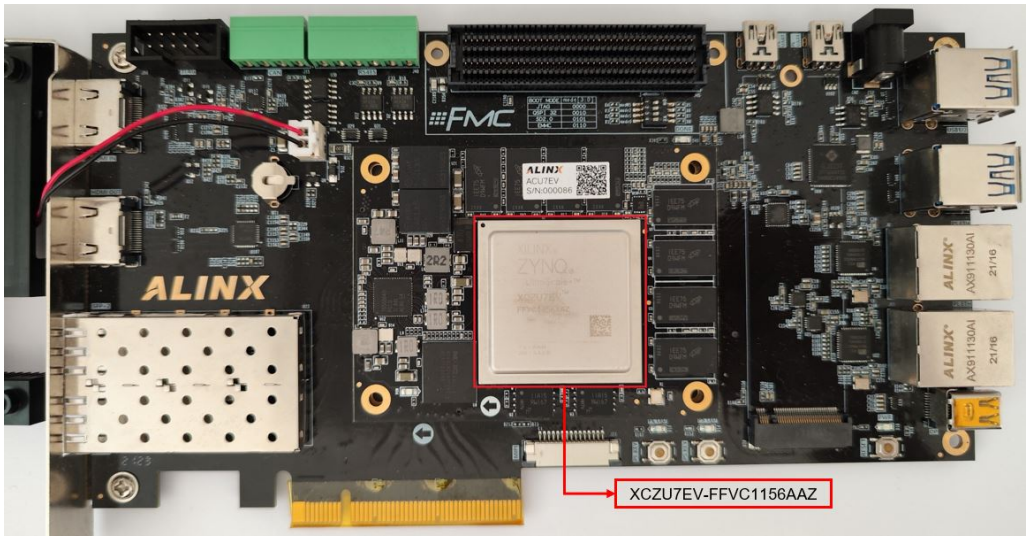

Figure 8. The schematic diagram of the development board and corresponding XCZU7EV-FFVC1156AAZ chip.

The benchmark circuit test set was selected from open-source test benchmarks in VTR and OpenCores [33,34]. The baseline program for comparative experiments utilizes the VTR source code. Our methodology, which combines the proposed algorithm and GPU acceleration with the VTR framework, focuses on comparing the time required for the packing and placement processes. The GPU acceleration methodology extends the PyTorch and DREAMPlace frameworks [25,26,35]. The included operator (op) layers consist of adjust_node_area, clustering_compatibility, dct, demand_map, density_map, pin_pos, pin_utilization, place_io, precondWL, rmst_wl, rudy, utility, and weighted_average_wirelength. The algorithm prototypes are fitted with a flexible combination of op and GPU adaptation. The hardware platform for the entire experiment consists of an Intel Core i5-10500 CPU, 36GB of RAM, and an NVIDIA A10 GPU running Linux Ubuntu 18.04.

Figures 9 and 10 illustrate the runtime statistics for selected test cases under various platform conditions using GPU acceleration, comparing the time between CPU- and GPU-accelerated implementations. On average, the packing process time is reduced by 28.42% due to the decreased computational complexity of the optimization algorithm. The experimental results demonstrate that the utilization of GPUs accelerates the placement program. The runtime for the CPU group ranges from 3.32 s to 171.78 s, while the GPU group's runtime varies from 1.32 s to 61.50 s. The speedup ratio ranges from 1.10 times to 5.55 times, with an average of 2.85 times. Regarding the initial CLB half perimeter wire length (HPWL), the original VTR process shows a minimum HPWL of 22,139 and a maximum of 27,125,48. In our proposed methodology, the minimum HPWL is 21,304, and the maximum is 26,856,911. As the number of HPWL is associated with the scale and characteristics of the benchmark circuits, Figure 11 visually depicts the percentage reduction in the initial HPWL compared to the original VTR process. The minimum reduction percentage is 0.19%, and the maximum is 11.50%, with an average HPWL reduction of 3.73%.

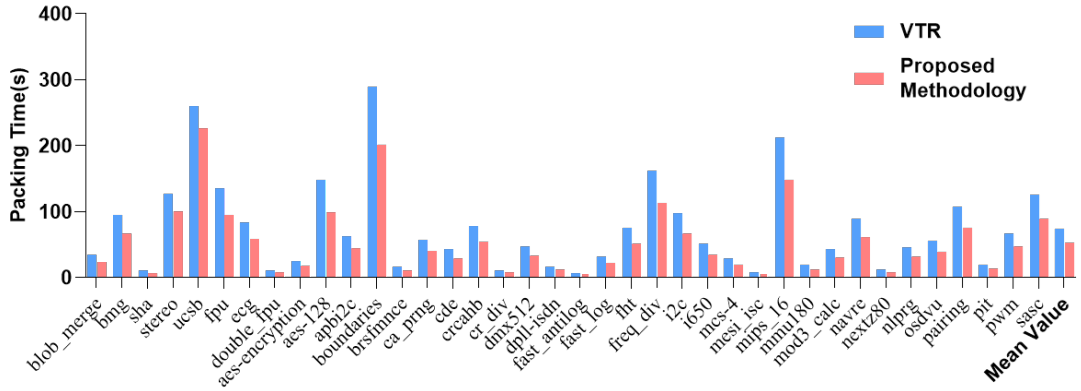

Figure 9. The comparison of packing time between VTR and the proposed methodology.

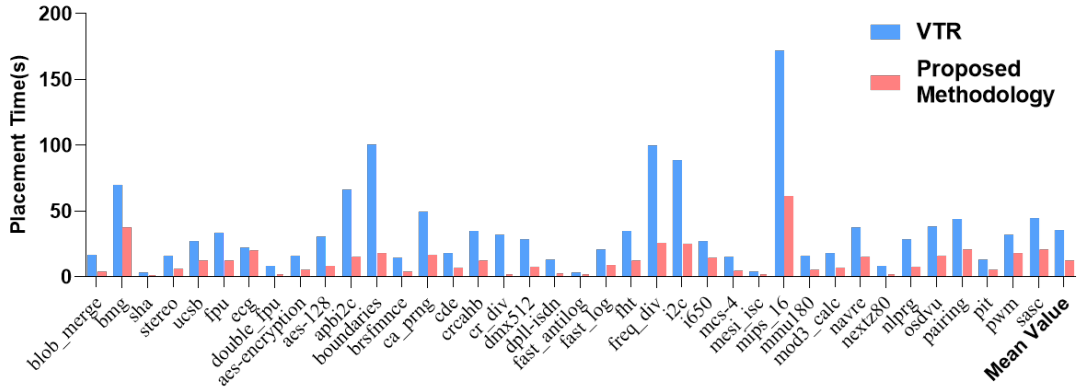

Figure 10. The comparison of CLB HPWL between VTR and the proposed methodology.

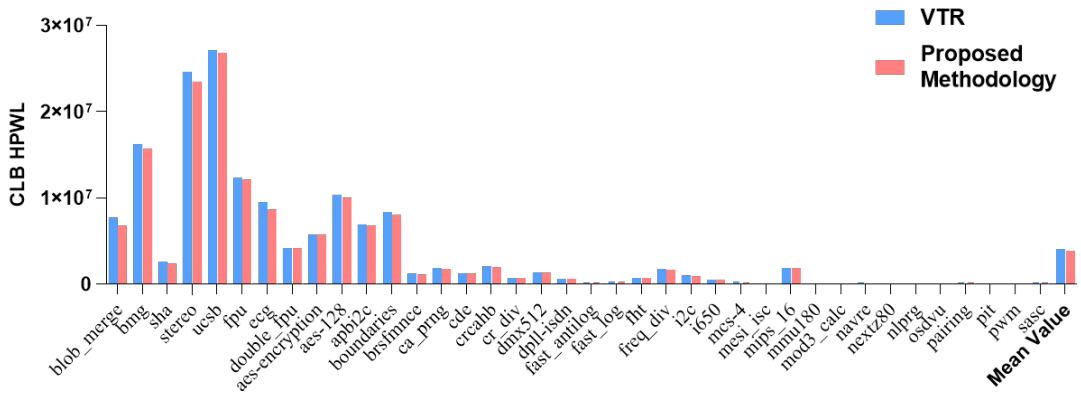

Figure 11. The comparison of placement time between VTR and the proposed methodology.

Table 1 offers a comprehensive statistical comparison between the original VTR and our proposed methodology concerning the packing time, placement time, CLB HPWL, and the total runtime of the packing and placement process. The experimental results demonstrate that, in comparison to the traditional algorithms used by VTR, the total packing and placement time is optimized by an average of 39.97%. The adjustment of the CLB merge generation strategy, conducting CLB clustering based on logical relationships, and appropriately increasing the number of generated CLBs effectively reduce the packing time overhead. Furthermore, GPU acceleration can significantly enhance the efficiency of distributed computing in the placement phase. Moreover, leveraging the weighted

Edmonds' Blossom algorithm, we attain an improved combination of CLB logic. This not only results in a higher HPWL gain but also optimizes timing outcomes, leading to an average maximum frequency increase of 5.08 MHz. The detailed data can be referenced in Freq. (frequency) columns of Table 1. Notably, all experiments were conducted using the open-source synthesizer Yosys tool, with a synthesis phase maximum frequency set to 300 MHz. In addition, the CLB resource utilization of the hardware remains at the same level, with an average percentage reduction of 2.09% compared to the VTR, as depicted in the CLB Util. (utilization) columns of Table 1.

Table 1. Comparisons between the VTR flow and our proposed methodology.

Benchmarks [1]	VTR						Proposed Methodology					
	Packing Time (s)	Placement Time (s)	Total Time (s)	CLB HPWL	Freq. (MHz)	CLB Util.	Packing Time (s)	Placement Time (s)	Total Time (s)	CLB HPWL	Freq. (MHz)	CLB Util.
blob_merge	35.19	16.65	51.84	7706102	282.62	13.10%	23.91	4.06	27.97	6819560	286.40	10.35%
bmg	94.55	69.73	164.28	16182815	271.43	28.37%	67.17	37.67	104.84	15711471	277.75	25.48%
sha	10.77	3.63	14.40	2568701	286.15	4.21%	6.52	1.32	7.84	2378427	297.58	2.17%
stereo	127.15	15.81	142.96	24659527	265.89	35.82%	100.64	6.07	106.71	23485264	266.74	33.24%
ucsb	260.39	27.50	287.89	27125480	261.83	43.16%	226.45	12.38	238.83	26856911	262.19	40.99%
fpu	135.54	33.20	168.74	12329764	268.97	27.53%	95.28	12.70	107.98	12088004	269.60	24.84%
ecg	83.64	22.50	106.14	9484434	273.78	26.21%	58.64	20.40	79.04	8701315	275.89	23.21%
double_fpu	10.54	8.21	18.75	4123667	285.63	7.12%	7.40	1.89	9.29	4115436	296.52	4.97%
aes-encryption	24.64	16.32	40.96	5773133	278.36	11.86%	17.46	5.76	23.22	5699046	284.69	9.17%
aes-128	148.54	30.43	178.97	10391640	269.27	27.93%	99.23	8.53	107.76	9991962	278.05	26.07%
apbi2c	63.28	66.50	129.78	6927760	282.38	14.78%	45.12	15.05	60.17	6725981	286.02	13.48%
boundaries	289.47	100.45	389.92	8313312	263.45	42.32%	201.75	18.10	219.85	8039954	263.72	40.83%
brsfmnce	16.91	14.50	31.41	1187616	287.86	10.51%	11.68	3.90	15.58	1100830	297.12	8.03%
ca_prng	57.82	49.78	107.60	1900186	283.21	13.75%	40.22	16.95	57.17	1881187	286.19	10.79%
cde	42.99	18.43	61.42	1187616	280.43	11.40%	29.95	7.29	37.24	1096672	283.68	9.28%
crcahb	78.13	34.54	112.67	2018947	271.39	18.65%	54.06	12.66	66.72	1927769	273.10	17.31%
cr_div	11.26	32.08	43.34	747758	287.44	8.34%	7.83	1.83	9.66	681233	297.36	6.01%
dmx512	47.53	28.81	76.34	1345965	279.62	12.54%	33.02	7.67	40.69	1238855	282.89	11.05%
dpll-isdn	17.42	13.42	30.84	611802	286.87	9.16%	12.14	2.93	15.07	565370	295.25	6.63%
fast_antilog	6.88	3.32	10.20	197356	281.92	3.25%	4.79	1.98	6.77	184318	299.75	1.78%
fast_log	31.77	20.70	52.47	276298	280.45	12.93%	22.07	9.10	31.17	276256	283.94	11.64%
fht	75.22	34.70	109.92	663115	278.93	14.87%	51.96	12.36	64.32	620420	280.50	13.32%
freq_div	162.33	100.11	262.44	1724098	267.39	28.36%	112.98	25.90	138.88	1698974	268.25	27.19%
i2c	97.09	89.03	186.12	1014175	277.34	25.32%	67.39	25.05	92.44	977281	279.11	22.33%
i650	51.08	26.91	77.99	533776	278.53	13.83%	35.52	14.63	50.15	504092	282.07	10.93%
mcs-4	28.75	15.63	44.38	242626	283.89	8.56%	20.08	4.85	24.93	242039	289.91	7.32%
mesi_isc	7.44	4.20	11.64	56425	285.12	5.75%	5.18	1.90	7.08	53854	297.27	3.87%
mips_16	213.02	171.78	384.80	1884581	257.28	40.43%	148.00	61.50	209.50	1761781	257.91	38.15%
mmu180	18.99	16.36	35.35	31410	283.26	10.77%	13.16	5.57	18.73	29221	290.67	9.40%
mod3_calc	43.21	18.43	61.64	59678	281.51	11.65%	30.11	7.29	37.40	57425	286.88	9.15%
navre	89.07	37.37	126.44	137260	278.67	14.36%	62.02	15.25	77.27	127944	282.03	12.68%
nextz80	12.56	8.46	21.02	22139	288.45	7.67%	8.73	2.15	10.88	21792	297.98	5.22%
nlprg	46.33	28.90	75.23	79700	280.96	12.86%	32.28	7.80	40.08	76753	285.94	11.17%
osdvu	55.78	38.56	94.34	103609	279.16	13.43%	39.00	16.15	55.15	94854	283.83	10.75%
pairing	107.26	43.95	151.21	165775	275.39	26.18%	75.00	20.90	95.90	154530	277.10	24.42%
pit	19.84	13.36	33.20	31278	285.29	9.07%	13.72	5.50	19.22	28968	293.94	7.51%
pwm	67.55	32.31	99.86	106346	277.46	14.21%	46.88	18.45	65.33	97470	281.38	12.01%
sasc	126.34	44.54	170.88	175471	274.23	27.17%	88.68	21.22	109.90	166121	275.66	25.13%
Mean Value	74.11	35.56	109.67	4002404	277.94	17.56%	53.05	12.49	65.55	3849456	283.02	15.47%

[1] blob_merge: image processing; bmg: finance calculation; sha: cryptography; stereo: computer vision; ucsb: educational CPU module; fpu: floating point unit; ecg: signal processing; double_fpu: double-precision floating point unit; aes-encryption: cryptography; aes-128: cryptography; apbi2c: bus protocol; boundaries: clock boundaries; brsfmnce: synchronous FIFO; ca_prng: cellular automata; cde: codec module; crcahb: bus protocol; cr_div: cached reciprocal divider; dmx512: communication controller; dpll-isdn: digital phase locked loop; fast_antilog: anti-logarithm calculation; fast_log: logarithm calculation; fht: fast Hartley transform; freq_div: adjustable frequency divider; i2c: bus protocol; i650: data storage module; mcs-4: data processing module; mesi_isc: coherency intersection controller; mips_16: educational CPU module; mmu180: memory management unit; mod3_calc: divider module; navre: multimedia application module; nextz80: combinatorial logic; nlprg: pseudo-random generator; osdvu: UART component; pairing: pairing algorithm module; pit: programmable interval timer; pwm: pulse width modulator; sasc: asynchronous serial controller.

To underscore the versatility of the GPU acceleration framework in handling very large-scale netlisting circuits, we conducted a methodology validation using a million-gate scale System-on-Chip (SoC) netlisting, specifically BJUT-RISC [36]. Table 2 details the FPGA resource overhead statistics, with the utilization rate of LUTs exceeding 90%. Notably, the benchmarked BJUT-RISC faced challenges completing placement experiments on VTR due to memory constraints (limited to 36GB RAM on the experimental machine). In contrast, our proposed methodology not only accomplished the entire placement process but also capitalized on GPU acceleration, demonstrating its effectiveness in overcoming memory limitations. Figure 12 illustrates the runtime overhead statistics for benchmarking BJUT-RISC. The GPU-accelerated step consumes a mere 0.23 h, constituting only 8% of the overall process. This highlights the optimization impact achieved through GPU acceleration, with the detailed placement time of CLBs standing at a commendable 0.97 h.

Table 2. FPGA resource overhead statistics for a multi-million gate netlist circuit.

Site Type	Available	Used	Utilization
LUTs	230400	217254	0.94
FFs	460800	165580	0.36
BRAM	11	8	0.73
DSP	1729	1340	0.78

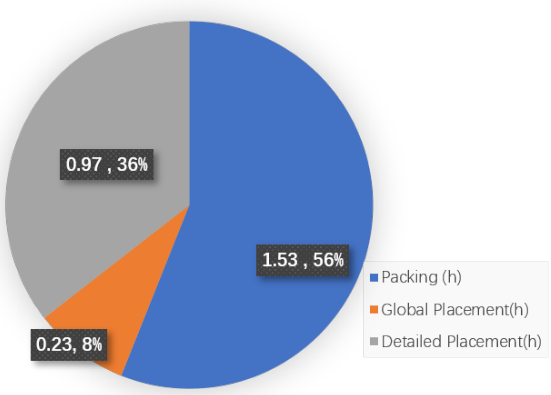

Figure 12. Runtime overhead statistics for benchmarking BJUT-RISC.

The outcomes of the global placement, when processed through the devised legalization algorithm, proficiently detect suboptimal solutions during the placement stage. Subsequent local optimization through the detailed placement algorithm results in a decreased number of iterations, thereby optimizing the placement process time. Hence, the proposed FPGA placement methodology is well-suited for the Xilinx FPGA processor chip XCZU7EV-FFVC1156AAZ, efficiently accomplishing the packing and placement tasks.

5. Conclusions

In this paper, we present an enhanced EDA methodology for FPGA packing and placement processes by leveraging the VTR and GPU acceleration frameworks. By optimizing algorithms and fine-tuning process parameters, we enhance the efficiency of traditional VTR methodology's packing and placement algorithms. The practicality of the GPU acceleration framework is extended, and a novel approach to FPGA placement is introduced by integrating different frameworks. Experimental results reveal an average 28.42% optimization in the packing process time, an average acceleration ratio of 2.85 times in the placement phase, and a 39.97% reduction in total packing and placement time compared to traditional algorithms, demonstrating engineering feasibility. Furthermore, we showcase the applicability of the proposed methodology to ultra-large-scale industrial-grade

netlist circuits. In future work, we plan to explore multi-objective placement optimization problems and streamline the CUDA programming framework to improve the quality of packing and placement results, along with enhancing the usability and practicality of the programming interface.

Author Contributions: Methodology, M.L.; software, Y.W.; original draft preparation, Y.W. and M.L.; review and editing, M.L.; analysis and validation, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Beijing Natural Science Foundation (No. 4234089).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Przybył, A. FPGA-Based Optimization of Industrial Numerical Machine Tool Servo Drives. *Electronics* **2023**, *12*, 3585. [CrossRef]
2. Li, D.; Feng, X.; Shen, C.; Chen, Q.; Yang, L.; Qiu, S.; Jin, X.; Liu, M. Vector-Based Dedicated Processor Architecture for Efficient Tracking in VSLAM Systems. *IEEE Embed. Syst. Lett.* **2023**, *15*, 182–185. [CrossRef]
3. Zhang, Z.; Liu, M.; Liu, Z.; Du, X.; Xie, S.; Ma, H.; Ding, G.; Ren, W.; Zhou, F.; Sun, W.; et al. Progress in a novel architecture for high performance processing. *Jpn. J. Appl. Phys.* **2018**, *57*, 04FA03. [CrossRef]
4. Hikawa, H. Place-and-Route Analysis of FPGA Implementation of Nested Hardware Self-Organizing Map Architecture. *Electronics* **2023**, *12*, 4523. [CrossRef]
5. Li, R.; Wu, J.; Liu, M.; Chen, Z.; Zhou, S.; Feng, S. HcveAcc: A high-performance and energy-efficient accelerator for tracking task in VSLAM system. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 198–203.
6. Betz, V.; Rose, J. VPR: A new packing, placement and routing tool for FPGA research. In *International Workshop on Field Programmable Logic and Applications, Proceedings of the 7th International Workshop, FPL '97, London, UK, 1–3 September 1997*; Springer: Cham, Switzerland, 1997; pp. 213–222.
7. Chen, G.; Cong, J. Simultaneous placement with clustering and duplication. In Proceedings of the 41st Annual Design Automation Conference, San Diego, CA, USA, 7–11 June 2004; pp. 740–772.
8. Betz, V.; Rose, J. Automatic generation of FPGA routing architectures from high-level descriptions. In Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 10–11 February 2000; pp. 175–184.
9. Yu, L.; Guo, B. Timing-Driven Simulated Annealing for FPGA Placement in Neural Network Realization. *Electronics* **2023**, *12*, 3562. [CrossRef]
10. Maidee, P.; Ababei, C.; Bazargan, K. Timing-driven partitioning-based placement for island style FPGAs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2005**, *24*, 395–406. [CrossRef]
11. Karypis, G.; Kumar, V. Multilevel k-way hypergraph partitioning. In Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, New Orleans, LA, USA, 21–25 June 1999; pp. 343–348.
12. Fiduccia, C.M.; Mattheyses, R.M. A Linear-Time Heuristic for Improving Network Partitions. Papers on Twenty-Five Years of Electronic Design Automation. 1988. pp. 241–247. Available online: https://dl.acm.org/doi/pdf/10.1145/62882.62910 (accessed on 18 December 2023).
13. Gort, M.; Anderson, J.H. Analytical placement for heterogeneous FPGAs. In Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, 29–31 August 2012; pp. 143–150.
14. Li, W.; Dhar, S.; Pan, D.Z. UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2017**, *37*, 869–882. [CrossRef]
15. Chen, Y.C.; Chen, S.Y.; Chang, Y.W. Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs. In Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 2–6 November 2014; pp. 647–654.
16. Meng, Y.; Li, W.; Lin, Y.; Pan, D.Z. elfplace: Electrostatics-based placement for large-scale heterogeneous fpgas. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 155–168. [CrossRef]
17. Zhu, Z.; Mei, Y.; Deng, K.; He, H.; Chen, J.; Yang, J.; Chang, Y.W. High-performance Placement Engine for Modern Large-scale FPGAs With Heterogeneity and Clock Constraints. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2023**. [CrossRef]
18. Xu, Y.; Khalid, M.A. QPF: Efficient quadratic placement for FPGAs. In Proceedings of the International Conference on Field Programmable Logic and Applications, Tampere, Finland, 24–26 August 2005; pp. 555–558.
19. Gopalakrishnan, P.; Li, X.; Pileggi, L. Architecture-aware FPGA placement using metric embedding. In Proceedings of the 43rd Annual Design Automation Conference, San Francisco, CA, USA, 24–28 July 2006; pp. 460–465.
20. Xu, M.; Gréwal, G.; Areibi, S. StarPlace: A new analytic method for FPGA placement. *Integration* **2011**, *44*, 192–204. [CrossRef]

21. Taj, I.; Farooq, U. Towards Machine Learning-Based FPGA Backend Flow: Challenges and Opportunities. *Electronics* **2023**, *12*, 935. [CrossRef]
22. Baig, I.; Farooq, U. Efficient Detailed Routing for FPGA Back-End Flow Using Reinforcement Learning. *Electronics* **2022**, *11*, 2240. [CrossRef]
23. Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.W.; Songhori, E.; Wang, S.; Lee, Y.J.; Johnson, E.; Pathak, O.; Nazi, A.; et al. A graph placement methodology for fast chip design. *Nature* **2021**, *594*, 207–212. [CrossRef] [PubMed]
24. Mallappa, U.; Pratty, S.; Brown, D. RLPlace: Deep RL guided heuristics for detailed placement optimization. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 14–23 March 2022; pp. 120–123.
25. Lin, Y.; Dhar, S.; Li, W.; Ren, H.; Khailany, B.; Pan, D.Z. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In Proceedings of the 56th Annual Design Automation Conference, Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
26. Rajarathnam, R.S.; Alawieh, M.B.; Jiang, Z.; Iyer, M.; Pan, D.Z. DREAMPlaceFPGA: An open-source analytical placer for large scale heterogeneous FPGAs using deep-learning toolkit. In Proceedings of the 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), Taipei, Taiwan, 17–20 January 2022; pp. 300–306.
27. Goswami, P.; Bhatia, D. Congestion Prediction in FPGA Using Regression Based Learning Methods. *Electronics* **2021**, *10*, 1995. [CrossRef]
28. Levine, J. *Flex & Bison: Text Processing Tools*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2009.
29. Xu, D.; Tian, Y. A comprehensive survey of clustering algorithms. *Ann. Data Sci.* **2015**, *2*, 165–193. [CrossRef]
30. Gabow, H.N.; Kaplan, H.; Tarjan, R.E. Unique maximum matching algorithms. In Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, Atlanta, GA, USA, 1–4 May 1999; pp. 70–78.
31. Zhu, Z.; Chen, J.; Peng, Z.; Zhu, W.; Chang, Y.W. Generalized augmented lagrangian and its applications to VLSI global placement. In Proceedings of the 55th Annual Design Automation Conference, San Francisco, CA, USA, 24–29 June 2018; pp. 1–6.
32. Churiwala, S.; Hyderabad, I. Designing with Xilinx® FPGAs. In *Circuits &Systems*; Springer: Berlin/Heidelberg, Germany, 2017.
33. Wolf, C.; Glaser, J.; Kepler, J. Yosys-a free Verilog synthesis suite. In Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip), Linz, Austria, 10 October 2013; p. 97.
34. Meng, L.; Shuai, L.; Yunfei, W.; Huixin, P.; Chunxue, L.; Liang, W. HDLcs1. 0: A Compiler and Simulator Framework for Hardware Description Language. In Proceedings of the 2023 International Symposium of Electronics Design Automation (ISEDA), Nanjing, China, 8–11 May 2023; pp. 93–96.
35. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. In Proceedings of the Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019.
36. Liu, M. A co-design method of customized ISA design space exploration and fixed-point library construction for RISC-V dedicated processor. *IEICE Electron. Express* **2022**, *19*, 20220244. [CrossRef]