symmetry

MDPI

Article

GRMD: A Two-Stage Design Space Exploration Strategy for Customized RNN Accelerators

Qingpeng Li †, Jian Xiao † and Jizeng Wei *

College of Intelligence and Computing, Tianjin University, Tianjin 300300, China; 2022244073@tju.edu.cn (Q.L.); xiaojian@tju.edu.cn (J.X.)
* Correspondence: weijizeng@tju.edu.cn
† Current address: Beiyang Campus, No. 135 Yaguan Road, Haihe Education Park, Tianjin 300350, China.

Abstract: Recurrent neural networks (RNNs) have produced significant results in many fields, such as natural language processing and speech recognition. Owing to their computational complexity and sequence dependencies, RNNs need to be deployed on customized hardware accelerators to satisfy performance and energy-efficiency constraints. However, designing hardware accelerators for RNNs is challenged by the vast design space and the reliance on ineffective optimization. An efficient automated design space exploration (DSE) strategy that can balance conflicting objectives is wanted. To address the low efficiency and insufficient universality of the resource allocation process employed for hardware accelerators, we propose an automated two-stage design space exploration (DSE) strategy for customized RNN accelerators. The strategy combines a genetic algorithm (GA) and a reinforcement learning (RL) algorithm, and it utilizes symmetrical exploration and exploitation to find the optimal solutions. In the first stage, the area of the hardware accelerator is taken as the optimization objective, and the GA is used for partial exploration purposes to narrow the design space while maintaining diversity. Then, the latency and power of the hardware accelerator are taken as the optimization objectives, and the RL algorithm is used in the second stage to find the corresponding Pareto solutions. To verify the effectiveness of the developed strategy, it is compared with other algorithms. We use three different network models as benchmarks: a vanilla RNN, LSTM, and a GRU. The results demonstrate that the strategy proposed in this paper can provide better solutions and can achieve latency, power, and area reductions of 9.35%, 5.34%, and 11.95%, respectively. The HV of GRMD is reduced by averages of 6.33%, 6.32%, and 0.67%, and the runtime is reduced by averages of 18.11%, 14.94%, and 10.28%, respectively. Additionally, given different weights, it can make reasonable trade-offs between multiple objectives.

Keywords: recurrent neural networks; customized accelerators; design space exploration; reinforcement learning

Citation: Li, Q.; Xiao, J.; Wei, J. GRMD: A Two-Stage Design Space Exploration Strategy for Customized RNN Accelerators. *Symmetry* **2024**, *16*, 1546. https://doi.org/10.3390/sym16111546

Academic Editor: Jie Yang

Received: 25 October 2024
Revised: 7 November 2024
Accepted: 14 November 2024
Published: 19 November 2024

1. Introduction

In recent years, the performance of recurrent neural networks (RNNs) has been steadily improving in tasks concerning the field of sequential data processing, such as machine translation and speech recognition. RNNs are artificial neural networks with internal circular connections, allowing data and information to circulate within these networks [1]. This enables RNNs to remember and process long-term sequential information. However, due to this internal recursion process, RNNs exhibit temporal sequence dependencies, where the output produced at each time step depends on the output of the previous time step. These dependencies generally make RNNs more difficult to train and less computationally efficient than classic convolutional neural networks (CNNs). Additionally, to address the vanishing and exploding gradient issues that can occur when processing long data sequences, different RNN models, such as long short-term memory (LSTM) [2] and gated recurrent units (GRUs) [3], have been proposed. This has led to larger and

more complex network structures accompanied by continuous computational complexity increases. Traditional processors are unable to efficiently handle these computational tasks, making the design of hardware accelerators for RNNs an important research direction. H. Cho et al. developed a hybrid architecture based on a GPU and a field-programmable gate array (FPGA) to accelerate RNNs [4]. Z. Zaghloul et al. proposed a new GRU hardware implementation method [5]. M. Wasef et al. accelerated a completely reconfigurable RNN via a system-on-a-chip approach on an FPGA [6].

Despite the progress achieved in terms of RNN hardware, several challenges remain. First, most optimization works focus on specific RNN structures, making them inherently limited by their network details and unable to be generalized to other RNNs. This means that even minor network structure changes can force the hardware resource configurations to be redesigned, highlighting the lack of a general design space exploration (DSE) method. Second, due to the large and complex RNN structures, the microarchitecture of hardware accelerators has become more complex, calling for additional hardware resources to be considered and allocated. These parameters of hardware accelerators, such as their bandwidth and memory, form a high-dimensional and vast design space. For example, the design space of a general matrix multiply (GEMM) accelerator is in the order of 10^9 [7], and evaluating the design of a single accelerator is often time-consuming. Exploring and comparing all possible design points is impractical. Additionally, developers often need to adjust their design spaces on the basis of various performance constraints that involve trade-offs between different performance metrics, and the manual exploration process may converge to local optima. Therefore, an automated and efficient DSE method is crucial for designing RNN hardware accelerators.

Recently, various DSE methods for hardware accelerators have been proposed. For example, T. Pacini et al. determined the configurations of RNN hardware accelerators by evaluating multiple user-defined constraints (such as accuracy and resource usage), achieving a high level of control over the final design [8]. J. Jiang et al. performed acceleration, fixed-point quantization, loop tiling, and piecewise linear approximation on activation functions to achieve high parallelism, accelerating RNNs while reducing their hardware resource usage levels and thereby achieving a throughput of 1223.53 GOP/s [9]. C. Zhang et al. quantified the computational throughputs and required memory bandwidths of potential design solutions and proposed a roofline model-based approach to explore the optimal solutions among multiple CNN design schemes [10]. M. Motamedi et al. developed a CNN analysis model to balance and utilize various parallelisms, achieving on-chip resource load balancing, generating the most efficient architecture for the target platform, and maximizing the throughput [11]. S. Yang et al. proposed a new strategy for searching within a hierarchical processing element (PE) mapping scheme, enabling layer fusion and configuration splitting to optimize the PE mapping process [12]. H. Miomandre et al. proposed a DSE strategy that selects appropriate bit widths and storage types for buffers to satisfy imposed constraints, reduces the memory usage of CNNs while ensuring their output quality, and reduces the memory usage of the SqueezeNet CNN by 58.6% [13]. M. R. Ahmed et al. used FPGA resources such as DSPs and LUTs as optimization objectives and integrated early failure prediction networks into a Bayesian optimization (BO) algorithm to accelerate the DSE process, reducing the search time 70-fold [14]. The current state-of-the-art studies include those of L. Mei et al., who used a genetic algorithm (GA) to explore deep neural networks (DNNs) and find 64% more energy-efficient solutions [15]; L. Zaourar et al., who achieved a great reduction of design space optimization by around 30% on the basis of an efficient implementation of a multi-objective GA [16]; Z. Xie et al., who proposed a DSE framework based on the GA and reduced the exploration time by up to 73.7× [17]; A. Raj et al., who used particle swarm optimization (PSO) to solve complex optimization problems [18]; and H. Kuang, who used BO to find Pareto-optimal designs in the vast configuration space of high-level synthesis [19].

Although these DSE methods have achieved some success, several issues remain. Some methods are time-consuming, rely on experience, and may converge to local optima,

whereas others fail to effectively balance multiple objectives. We summarize the existing problems related to DSE for customized RNN accelerators as follows:

1. Most RNN accelerator designs rely on manual optimization, but the hardware design space is vast, and manual explorations are time-consuming and inefficient. An efficient automated DSE strategy is lacking.
2. Hardware DSE usually involves multiple conflicting objectives. These objectives may have different levels of importance. Given the user-defined weights of objectives, most strategies are unable to adjust their exploration strategies and solutions based on the weights. They lack the ability to make reasonable trade-offs between them. No strategy is capable of finding the optimal balance between different objectives.
3. In DSE, a modeling framework that maps the hardware accelerator design space to the input space of the exploration algorithm to achieve efficient automated DSE is lacking.

To address these challenges, this paper proposes a GA- and reinforcement learning (RL)-based multi-objective DSE (GRMD) strategy for customized RNN accelerators, aiming to achieve efficient hardware resource allocation, find the optimal hardware design scheme in a high-dimensional design space, attain improved design efficiency, and achieve approximately optimal energy efficiency. The main contributions of this paper include the following.

1. We propose an automated two-stage DSE strategy in which the latency, power, and area of the target accelerator are used as optimization objectives. The strategy offers an effective and efficient hardware DSE solution.
2. We employ a GA-based approach for the initial exploration process to narrow the design space and reduce the impact of the area on the exploration results.
3. A subspace optimization scheme based on RL is designed to develop optimal resource allocation strategies, simplifying the trade-offs between multiple objectives and attaining improved exploration efficiency. The RL algorithm learns the relationship between resource allocation and accelerator performance through its actor and critic networks then adjusts its output strategy and solutions to achieve multi-objective trade-offs under imposed user constraints.
4. Experiments demonstrate that, compared with similar DSE algorithms, the proposed strategy generates better solutions and can achieve latency, power, and area reductions of 9.35%, 5.34%, and 11.95%, respectively.

The remainder of this article is organized as follows: Section 2 summarizes the background information and explains the motivations of this work. Section 3 introduces the overall framework of the GRMD strategy. Section 4 provides the exploration details of GRMD. Section 5 presents the experimental results and discussion, which are followed by conclusions in Section 6.

2. Background and Motivation

2.1. Design Space Exploration

Hardware DSE involves two strategies: exploration and exploitation. Exploration refers to trying new areas within the space and evaluating different combinations of accelerator parameters to discover potentially superior solutions. This strategy emphasizes exploring unknown regions and reduces reliance on the current optimal solutions, which can prevent the DSE algorithm from getting stuck in local optima. It ensures thorough searching of the design space to identify potential global optimal solutions. Exploitation focuses on further optimizing the currently discovered superior solutions based on existing exploration results. This strategy fine-tunes design points around the current optimal solutions to make solutions approach the optimal direction, thus avoiding resource wastage.

In design space exploration, symmetrical exploration and exploitation are key to finding the global optimal solution. Excessive exploration can consume a significant amount of time, as the hardware design space has multiple dimensions, leading to a vast number of design points, and evaluating each design point often takes considerable

time. Excessive exploitation, on the other hand, may cause exploration to get trapped in local optima.

2.2. Design Space Exploration Algorithms

Traditional DSE usually uses meta-heuristics algorithms, such as GAs or PSO. In terms of ML-based DSE, several studies leverage BO to find solutions. PSO has fewer parameters and faster convergence speed, yet its efficiency is low and may converge to local optima when exploring in high-dimensional hardware design space. BO effectively utilizes the results of each evaluation through a surrogate model, making it suitable for expensive objective function evaluations. However, its effectiveness depends on the selection of model. In the hardware design space, the joint impact of different resources on the resulting performance is not clear. Therefore, the surrogate model may not match well with the hardware design space model, leading to reduced effectiveness. GAs can effectively maintain population diversity through selection, crossover, and mutation operators, and avoid getting stuck in local optima in high-dimensional space. However, in hardware DSE, the process of evaluating an accelerator configuration to obtain its performance values is time-consuming. Multiple iterations of GAs may result in excessively long running time. RL uses Monte Carlo learning (MCL) and neural networks to reduce reliance on models of the design space. The networks can also complete inference in a very short amount of time, significantly reducing runtime. Furthermore, based on the rewards, RL can learn the relationship between accelerator configurations and performance metrics, thereby achieving effective multi-objective trade-offs.

2.3. Spatial Accelerators

This paper uses a custom spatial accelerator to accelerate RNNs. Its microarchitecture is shown in Figure 1. Spatial accelerators have been successfully used to accelerate various DNNs, including CNNs and RNNs. They can support multiple dataflows and improve parallelism through data reuse, thereby accelerating computational tasks. The custom spatial accelerator used in this paper consists of three basic parts: a 2D array of PEs, a storage system, and a controller. The PE array is composed of several computing units that communicate via on-chip interconnections and form a 2D array to attain large-scale parallelism. Each PE contains an ALU and registers to execute computational tasks. The storage system consists of the local memory inside each PE and a scratchpad shared by all PEs. The local memory is optional, while the scratchpad can be divided into banks; more banks allow the scratchpad to support greater levels of concurrent data access. The controller includes a direct memory access controller (DMAC) and an instruction decoder (ID). The DMAC interacts with off-chip caches to handle large data movements between the scratchpad of the accelerator and the external DRAM. The ID retrieves and decodes instructions derived from the CPU core to control the PEs and DMAC. Compared with commercial accelerators possessing more complex memory and control structures, the custom accelerator used in this paper has a simple architecture, high flexibility, and high efficiency, making it suitable for accelerating various RNNs.

The hardware design space is composed of all possible hardware resource configurations contained in the space, where each design point includes the types and values of different hardware resources. Based on the microarchitecture of the examined accelerator, the design space can be represented as shown in Table 1, where WS and OS refer to weight stationarity and output stationarity, respectively. In a dataflow with WS, the weights are stored in PEs, and the calculation results flow between the PEs. In a dataflow with OS, the calculation results are saved in PEs, while the weights flow between the PEs, and the calculation results are updated.

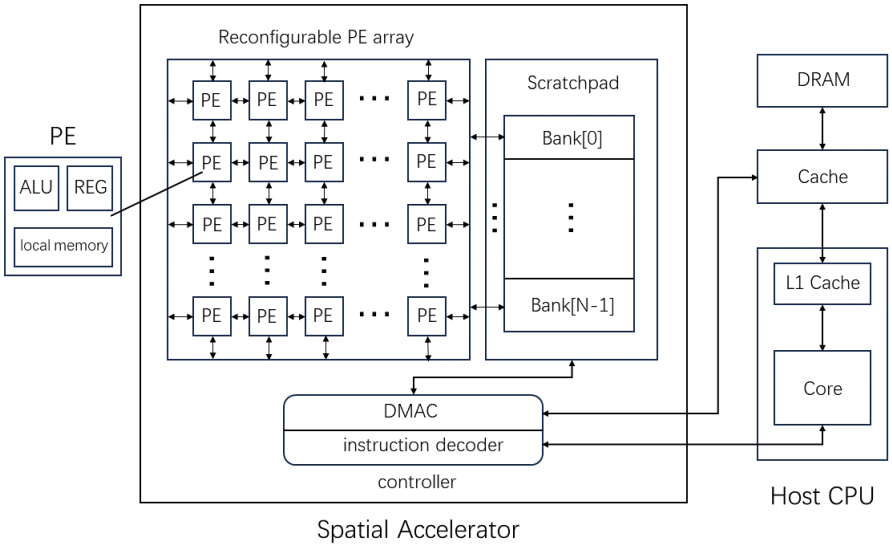

Figure 1. Microarchitecture of our custom spatial accelerators.

Table 1. Hardware design space.

Hardware Resources	Description	Values
x	PE array size	Integers within [4, 32]
local_capacity	Capacity of the local memory	Integers within [64, 256] (KB)
dataflow	PE array dataflow	"WS", "OS"
sp_capacity	Capacity of the scratchpad	Integers within [128, 512] (KB)
sp_banks	Number of scratchpad banks	Integers within [1, 8]
dma_buswidth	Bus width of the DMA	Integers within [64, 128] (bps)
dma_maxbytes	Maximum number of DMA bytes	Integers within [64, 128]
dtype	data type	"int8"

2.4. Motivational Case Study

In this case study, we compare the latency, power, and area values of four GEMM accelerators with different parameters to demonstrate the complex trade-offs between different objectives. These accelerators have 4×4, 8×8, 16×16, and 32×32 PE arrays. Other hardware parameters have similar impacts, but the PE array size has the most significant effect, so we choose to compare the changes observed in this parameter. The results are shown in Figure 2. As the PE array size gradually increases from 4 to 8, 16, and 32, the latency decreases, whereas the power and area significantly increase. Specifically, the latency decreases by 50.6%, 30.5%, and 58.3%, while the power increases by approximately 3.1, 3.4, and 4.8 times and the area increases by 4.6, 7.4, and 11.2 times, respectively. The order of magnitude of the area is much larger than those of the latency and power, and its variation is the most significant. This finding indicates that when hardware accelerators are evaluated, area has the greatest impact on the results, highlighting the complex trade-offs among latency, power, and area.

Therefore, when exploring the hardware design space, we adopt a divide-and-conquer approach. First, we take area as the objective and partially explore the design space. Then, we take latency and power as objectives to explore the obtained subspace, generating a Pareto solution set. By decomposing the DSE task into two parts, we can reduce the impact of the area on the exploration results, thus simplifying the trade-offs between multiple objectives and improving the efficiency of the proposed strategy.

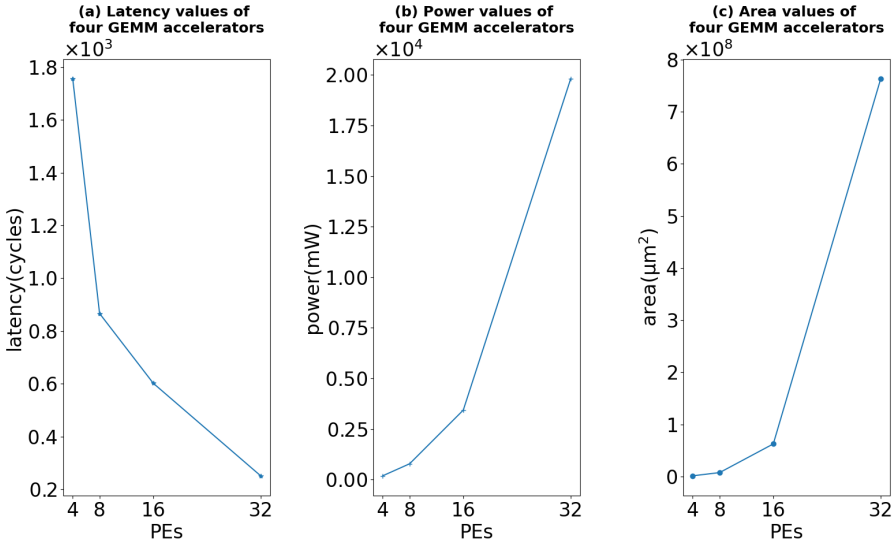

Figure 2. Latency, power, and area values of four GEMM accelerators with different PE arrays.

3. Overall Framework

Figure 3 shows the overall framework of GRMD. The DSE and optimization processes for customized RNN accelerators are completed in three steps.

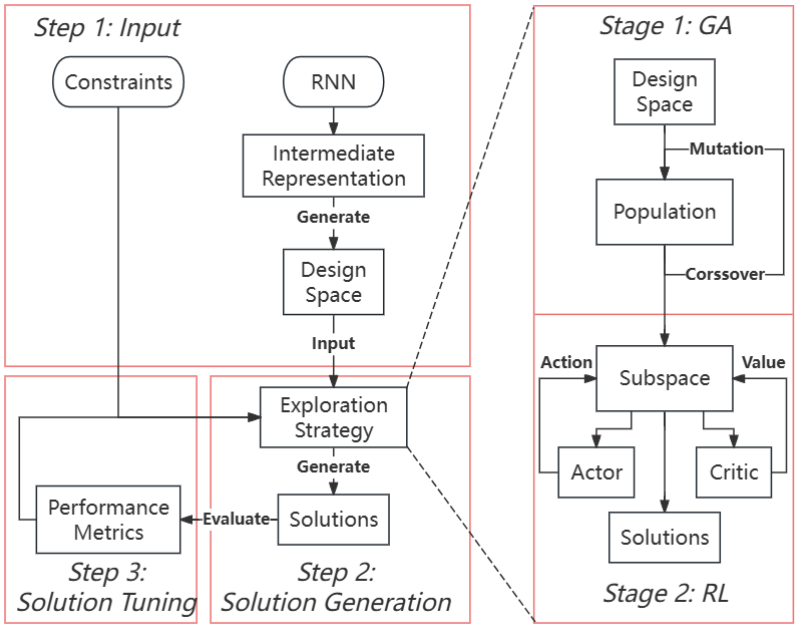

Figure 3. Overall framework of the DSE method.

Step 1: Input. The user specifies the workload and constraints to be imposed on the RNN in the input module. In this paper, the FPGA resource constraints are the limits of the available resources during DSE, whereas the constraints imposed on the latency, power, and area of the hardware accelerator are the limits of performance metrics. The RNN workloads are then converted into intermediate representations (IRs) to generate hardware design spaces. Some RNN subworkloads are handled by the accelerator, whereas the remaining parts are handled by software, which repeatedly calls the accelerator. This paper focuses mainly on hardware DSE. The IRs and hardware/software partitioning scheme are not discussed here.

Step 2: Solution Generation. After generating the hardware design space, the design space is explored to generate diverse solutions. GRMD is a two-stage hardware DSE strategy, with a divide-and-conquer approach adopted. In the first stage, GRMD uses a GA to partially explore the design space with the accelerator area as its objective, thus reducing the design space. Then, in the second stage, it uses an RL algorithm with latency and power as its objectives to optimize the subspace generated by GA. In contrast, strategies based on a single algorithm directly take latency, power, and area as objectives and explore the complete and vast design space. The divide-and-conquer approach aims to help GRMD prioritize the most impactful objectives, thereby simplifying multi-objective trade-offs. This does not bring GRMD to find the Pareto optimal solution. The Pareto optimal solutions are obtained through continuous exploration and exploitation of the design space, along with comparisons of non-dominated relationships between solutions. GRMD accepts the hardware design space generated in the first step and the FPGA resource constraints as inputs and outputs the resource configurations of the hardware accelerator. During exploration, GRMD checks whether the FPGA resource constraints are satisfied after performing resource allocation. If not, it adjusts the resource allocation scheme accordingly. The detailed process of DSE is described in Section 4.

Step 3: Solution Tuning. After generating solutions, the framework evaluates these solutions in order to obtain the performance metrics of the accelerator. It then iteratively adjusts the solutions on the basis of these metrics to acquire Pareto optimal solutions. Additionally, if the metrics violate the user-imposed constraints, GRMD attempts to regenerate a new accelerator configuration. If GRMD cannot generate a configuration that satisfies the performance constraints, it outputs the best solution among the constraint-violating solutions for the user to choose from.

4. Design Space Exploration Strategy

In this work, the goal of hardware DSE is to generate hardware resource configurations and minimize their latency, power, and area while satisfying the given constraints. The hardware resource configurations include the PE array size, scratchpad capacity, local memory capacity, hardware dataflow, number of scratchpad banks, DMA bandwidth, maximum number of DMA bytes, and hardware data type parameters.

Traditional heuristic algorithms have achieved good results in hardware DSE scenarios. For example, NSGA2 can effectively solve multi-objective optimization problems (MOPs) and introduce randomness to avoid local optima. However, GAs still have several limitations. First, a basic assumption of a GA is that the offspring generated via crossover and mutation are usually better than their parents. This is not always the case in hardware DSE because the relationships and impacts of resource allocation decisions are not clear, which reduces the effectiveness of such algorithms. Second, a GA requires multiple iterations and evaluations for each new problem to re-explore the design space and is unable to utilize previous experiences, leading to long exploration times and reduced efficiency. Additionally, the multiple objectives involved in hardware DSE often have different importance levels, necessitating trade-offs between the objectives. Although GAs can effectively address MOPs via nondominated sorting and crowding distances, they cannot explicitly utilize the trade-offs between multiple objectives, making them more suitable for general scenarios rather than targeted applications with user constraints.

To address these issues, we introduce a DSE method based on the actor–critic RL algorithm. RL is a machine learning method used for decision-making purposes. Since hardware DSE can be formulated as a sequential decision-making problem, where values are assigned to each hardware resource in sequence, it naturally falls into the realm of RL. An RL problem consists of an agent, an environment, states, actions, and rewards. The agent continuously learns and forms a policy that generates actions to maximize the received rewards. Compared with GAs, RL has the following advantages and features:

- Effectiveness: RL algorithms can actively learn resource allocation strategies through balanced exploration and exploitation processes, addressing the issue that GAs may not always be effective.
- Efficiency: After training, RL algorithms can quickly infer solutions within a few seconds, even in vast design spaces, solving the inefficiency problem faced by GAs.
- Flexibility: With neural networks, RL can actively learn and leverage the trade-offs between multiple objectives, generating hardware resource configurations on the basis of user requirements and making them more suitable for targeted applications.

As mentioned before, the divide-and-conquer approach can be adopted when exploring the hardware design space, thereby decomposing the exploration task into two parts to reduce the impact of the accelerator area and simplify multi-objective trade-offs. Therefore, the DSE strategy proposed in this paper combines a GA and RL to leverage the advantages of both types of algorithms. When taking the area as the objective, the GA is used to partially explore the space, which makes the RL training process easier and maintains the diversity of the solutions by introducing randomness to avoid local optima. When latency and power are taken as objectives, RL is used to explore the obtained subspace, generating Pareto optimal solutions. RL can actively learn resource allocation strategies and multi-objective trade-off strategies, quickly inferring solutions after completing the training process and thus compensating for the inefficiencies of the GA.

4.1. Initial Design Space Exploration Based on a Genetic Algorithm

The hardware DSE problem based on a GA can be modeled as follows:

Input: Design space, X,; network structure S; FPGA resource limits L_{LUT}, L_{DSP}, L_{BRAM}, $L_{bandwidth}$; maximum number of iterations T; population size N.

Output: A set of hardware resource configurations with the area as the objective.

Optimization Objective: Area.

Constraints: area $\leq$ $area_{max}$.

where "area" refers to the area of the accelerator. The optimization objective is to sort the areas in ascending order and output the top-N hardware resource configurations. $Area_{max}$ is a user constraint that represents the upper limit of the accelerator area. The FPGA resource limits include the number of LUTs L_{LUT}, the number of DSPs L_{DSP}, the amount of BRAM L_{BRAM}, and the on-chip bandwidth $L_{bandwidth}$.

The mapping between the DSE process and the GA is as follows:

Gene G: The value of each hardware resource. For example, G_1 = 8 indicates that the PE array size of the accelerator is 8×8.

Chromosome C: A combination of hardware resource values (a combination of genes). For example, C = {8, 128, 64, "WS", 2, 64, 64, "int8"} represents a resource allocation scheme for the accelerator in which the PE array size is 8×8, the scratchpad capacity is 128 KB, the local memory capacity is 64 KB, the dataflow exhibits WS, the scratchpad is divided into 2 banks, the DMA bandwidth is 64 bps, the maximum number of DMA bytes is 64, and the data type is int8.

Population P: A set of chromosomes, such as P = {C_1, C_2, . . ., C_N}.

Elite E: A set of high-performance chromosomes.

Fitness F: The negation of the hardware evaluation metric, $F = -area$. The structure of the accelerator is determined by the RNN structure S, and the accelerator is then evaluated on the basis of the hardware resource values.

Algorithm 1 presents the overall DSE procedure based on the GA. It first initializes a population P (line 1). Generating a diverse population is fundamental for conducting hardware DSE. Therefore, a reasonable random probability density function must satisfy the following two requirements: First, the function should be a discrete distribution that conforms to the discrete values of hardware resources. Second, the function should generate a subspace that covers as many solutions as possible. Therefore, we choose the discrete uniform distribution as the random probability density function for initializing the population. The discrete uniform distribution variable is discrete, thus fitting the characteristics

of hardware resource values (e.g., the size of the PE array can only be a positive integer). Moreover, the probability value at each variable value is equal, ensuring the diversity of the population genes without significant biases and thus avoiding convergence to local optima.

Algorithm 1 DSE process based on a GA

1: Initialize a population $P \in X$
2: **for** $t = 1$ **to** T **do**
3: Randomly select chromosomes from P to obtain P'
4: Obtain the offspring Q through crossover and mutation operations
5: **while** the number of required resources $> L$ **do**
6: Reduce hardware resources
7: **end while**
8: Merge Q into P
9: Sort the chromosomes based on their fitness
10: Divide P into a Ps that satisfies the user constraints and a Pv that does not
11: Select chromosomes from Pv and Ps to obtain the new P
12: **end for**
13: **if** $Ns \geq N$ **then**
14: **return** $x = \{\text{The first } N \text{ chromosomes in } Ps\}$
15: **else**
16: **return** $x = Ps \cup \{\text{The first } N - Ns \text{ chromosomes in } Pv\}$
17: **end if**

The algorithm then iteratively explores the design space until it reaches the maximum number of iterations T (lines 2–12). In each iteration, the algorithm first randomly selects a subset of chromosomes P′ from the population P (line 3) to perform an initial exploration of the design space. For each chromosome C in P′, crossover and mutation operations are implemented to obtain offspring populations Q (line 4). Crossover is a key step for improving the quality of the genetic exploration process. Generally, it combines two chromosomes (called "parents") by exchanging their genes to form new chromosomes (called "offspring"). By continuously applying crossover operations, superior genes are expected to appear more frequently in the population and eventually converge to the optimal solution. The available crossover methods can be divided into small-granularity crossover methods based on genes and large-granularity crossover methods based on chromosomes, as shown in Figure 4. Since the hardware resource values are discrete and independent of each other, this paper uses small-granularity crossover based on genes and classifies the hardware resources into two categories, applying different crossover operators accordingly. First, for "Choice"-type resources that are completely discrete, such as the PE array size or the dataflow (e.g., the possible dataflow values are "WS" and "OS"), the crossover operator can be defined as follows:

$$G_i^{child} = G_i^{parent1}\gamma_i + G_i^{parent2}(1 - \gamma_i) \tag{1}$$

$$\gamma_i = \begin{cases} 1, & u_i \leq 0.5 \\ 0, & u_i > 0.5 \end{cases} \tag{2}$$

where u_i is a random number within [0, 1]. Second, for "Integer"-type resources such as the number of scratchpad banks, which can be any integer within a range (e.g., any integer from 1 to 8), the crossover operator is defined as shown below:

$$G_i^{child} = \left\lfloor G_i^{parent1}\gamma_i + G_i^{parent2}(1 - \gamma_i) \right\rfloor \tag{3}$$

where γ_i is a random number within [0, 1]. The advantage of small-granularity crossover is that the genes of offspring can automatically adjust the crossover proportion of the parent genes. Large-granularity crossover uses chromosomes as its crossover units, so

the offspring chromosomes are similar to those of the parent with a higher crossover rate. In contrast, small-granularity crossover allows for the use of different crossover rates for different genes, enabling the offspring chromosomes to inherit varying proportions of their parental characteristics. Therefore, small-granularity crossover can significantly increase the diversity of gene variations and better maintain the overall stability of the chromosomes. For example, for hardware configurations C1 = {8, 128, 128, "WS", 2, 64, 256, "int8"} and C2 = {32, 256, 64, "OS", 6, 128, 64, "int8"}, the offspring chromosomes are {15, 130, 87, "OS", 2, 69, 136, "int8"} and {15, 157, 113, "WS", 2, 78, 211, "int8"}, respectively, assuming the gene crossover rates of small-granularity crossover are {67%, 98%, 36%, 0%, 93%, 91%, 38%, 100%} and the chromosome crossover rate of large-granularity crossover is 77%. It can be seen that the offspring generated by large-granularity crossover is more similar to C1, while the offspring generated by small-granularity crossover have higher diversity. The hardware resource values are independent of each other, and different resources can be classified into different types. This makes small-granularity crossover more suitable for DSE problems in this paper. Moreover, small-granularity crossover exchanges genes between two parents to produce new chromosomes, helping to retain superior genes within the population while also allowing for the exploration of new gene combinations, thus achieving symmetry between exploration and exploitation.

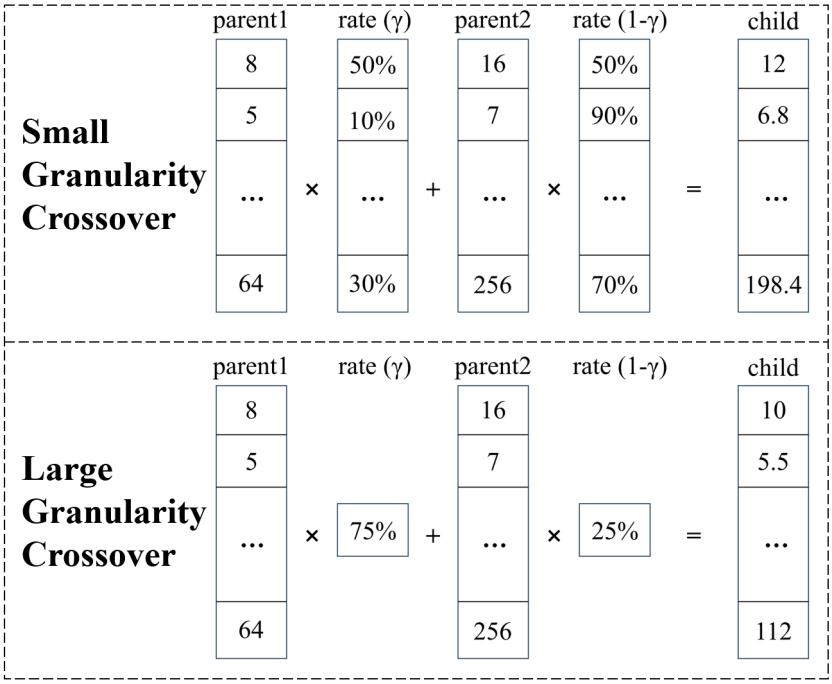

Figure 4. An overview of small-granularity and large-granularity crossover.

Crossover operations enable offspring to inherit genes from their parents, which may lead to the chromosomes in the population becoming similar and their gene sequences tending to be the same. Therefore, after the crossover operations, mutation operations are needed to reintroduce genetic diversity to the population. This effectively prevents the DSE process from converging to a local optimum. Different mutation operators are used for different categories of hardware resources. For "Choice"-type resources, the genes randomly mutate to other possible values in the design space. For "Integer"-type resources, polynomial mutation is adopted. Equation (4) shows the formula employed for gene mutation:

$$G_i^{child} = G_i^{parent} + \delta(G_u - G_l) \tag{4}$$

where G_u and G_l are the upper and lower bounds of the examined resource value, respectively. δ is calculated as follows:

$$\delta = \begin{cases} [2u + (1-2u)(1-\delta_1)^{\eta+1}]^{\frac{1}{\eta+1}} - 1, & u \leq 0.5 \\ 1 - [2(1-u) + 2(u-0.5)(1-\delta_2)^{\eta+1}]^{\frac{1}{\eta+1}}, & u > 0.5 \end{cases} \quad (5)$$

where $\delta_1 = \frac{G_i^{parent} - G_l}{G_u - G_l}$ and $\delta_2 = \frac{G_u - G_i^{parent}}{G_u - G_l}$ represent the proportion of the distance (from the gene value to the lower bound) to the total length of the range and the proportion of the distance (to the upper bound) to the length of the range, respectively. η is a user-defined distribution index (a nonnegative real number) that is used to indicate the intensity of the mutation exhibited by an individual. u is a random number within (0, 1) that is used to determine the gene value mutation range. By introducing the user-defined distribution index and a random number, the value of a gene can be reasonably mutated, improving the stability of the algorithm. Compared with traditional mutation operators (such as step mutation), polynomial mutation helps improve the exploration efficiency and mutation quality of the GA.

After the crossover and mutation operations, i.e., the completion of resource allocation, the algorithm checks if the obtained solutions can satisfy the constraints imposed on the FPGA resources (lines 5–7). If the required resources exceed the FPGA limits, then the hardware resource values are gradually reduced to satisfy the constraints. These adjustments are made based on the types of FPGA resources with three cases:

1. If the required LUTs exceed the limit L_{LUT} or the required DSPs exceed the limit L_{DSP}, then the PE array size is reduced to decrease the number of PEs.
2. If the required BRAM exceeds the limit L_{BRAM}, then the scratchpad capacity or the local memory capacity of the Pes is reduced.
3. If the required bandwidth exceeds the on-chip bandwidth limit $L_{bandwidth}$, then the DMA bandwidth is reduced.

After applying crossover, mutation, and constraint checking, i.e., at the end of each iteration, different chromosomes need to compete with each other. The superior individuals survive and reach the next iteration. To expand the sampling space, the offspring populations Q are merged into the parent population P (line 8) to compete together via an elite strategy. This helps the algorithm retain the superior individuals from the parent population. By continuously selecting the fittest individuals, the best chromosomes emerge at the end of the exploration process. Therefore, the chromosomes need to be sorted so that superior individuals rank higher while inferior individuals rank lower. In this work, the smaller the hardware area is, the greater the fitness of the corresponding chromosome, and the higher its rank.

The new population P is obtained by sorting the chromosomes in descending order of their fitness values (line 9). If the user specifies constraints to be imposed on the area of the accelerator, then the sorted population P is divided into a part Ps that satisfies these constraints and a part Pv that does not. If the number of chromosomes Ns contained in Ps is not less than N, then the top N chromosomes in Ps are selected as the new population P. Otherwise, after selecting individuals from Ps, the remaining N−Ns chromosomes are selected from Pv to maintain the population size at N (lines 10–11).

After the maximum number of iterations is reached, the top N solutions that satisfy the constraints, i.e., the hardware accelerator resource configurations with the smallest objective function values, are returned. If the number of solutions that satisfy the constraints Ns is less than N, then the solutions derived from Ps and the top N−Ns solutions acquired from Pv are returned (lines 13–17). In this way, the solutions that do not satisfy the user constraints are returned to the RL algorithm, even if no solutions do so, ensuring the generation of solutions for the user reference and selection processes and thus enhancing the stability of the algorithm.

We achieve symmetrical exploration and exploitation by combining various operators and strategies. The crossover operator exchanges genes between the current good solutions, leveraging existing beneficial traits while potentially generating new combinations. The mutation operator randomly changes some genes in chromosomes, providing the algorithm

with an opportunity to escape the current local optimum, thus avoiding early convergence and promoting exploration. The elite strategy merges parents and offspring, then selects the chromosomes with higher fitness, thereby directly preserving the current optimal solutions and promoting exploitation.

4.2. Subspace Optimization Based on an Reinforcement Learning Algorithm

The hardware DSE problem based on an RL algorithm can be modeled as follows:

Input: Design space X; network structure S; FPGA resource limits L_{LUT}, L_{DSP}, L_{BRAM}, $L_{bandwidth}$; maximum number of episodes E; maximum number of timesteps T; objective weights A and B; and network parameters θ and ω.

Output: Hardware resource configuration x that maximizes the received rewards.

Optimization Objective: $F = A \times latency + B \times power$

Constraints: latency $\leq$ latency_{max}, power $\leq$ power_{max}.

where "latency" and "power" refer to the latency and power of the accelerator, respectively. The optimization objective is the weighted sum of these two parameters, where A and B are the weights corresponding to the objectives. The latency $_{max}$ and power $_{max}$ are user constraints that represent the upper limits of latency and power, respectively. The FPGA resource limits include the number of LUTs L_{LUT}, the number of DSPs L_{DSP}, the amount of BRAM L_{BRAM}, and the on-chip bandwidth $L_{bandwidth}$. θ and ω are hyperparameters for representing the network parameters.

The mapping between DSE and the RL algorithm is as follows:

States: The set of all possible states S. In this paper, the state S_t at time step t is a tuple representing the allocation of hardware resources and the current hardware resource under consideration. The allocated hardware resource values are stored in an array, whereas the unallocated values are 0. For example, a possible state is {[8, 128, 0, "", 0, 0, 0, ""], "dataflow"}, where the array refers to the allocation of hardware resources, showing that the PE array size is 8×8 and that the scratchpad capacity is 128 KB, whereas other resources such as the dataflow and bandwidth are not allocated. The "dataflow" is the current hardware resource under consideration: the dataflow.

Actions: The set of all possible actions A. In this paper, given the current state and the hardware resource under consideration, action A_t taken at time step t allocates a value to that hardware resource. For example, in the state {[8, 128, 0, "", 0, 0, 0, ""], "dataflow"}, where the current resource under consideration is the dataflow, and the possible actions include "WS" and "OS", which represent dataflows with WS and OS, respectively.

State Transition: The probability distribution of the next state given a current state and an action. In this paper, the next state is randomly selected from all possible states.

Reward: The immediate reward R for taking an action in a state. In this paper, the intermediate rewards for all actions are 0, whereas the reward for the final action is the negative weighted sum of the hardware latency and power values.

Algorithm 2 presents the overall procedure of DSE, which is based on the RL algorithm. It first initializes the actor and critic networks (line 1). The algorithm uses weights θ and ω to initialize the parameters of the actor and critic, respectively. In DSE, which is based on sequence decisions, to obtain resource configurations with latency and power values that are as small as possible, a policy function is needed to input the state and output the action for completing the resource allocation process. Additionally, a value function is needed to evaluate the resource allocation results. However, as mentioned before, the relationships between resource allocation decisions are unclear, and the impact of allocating a resource may not always be good. It is not possible to determine the exact policy function and value function. Therefore, we use "actor" of the neural network to approximate the policy function and the "critic" of the neural network to approximate the value function. Both the actor and critic are 2-layer multilayer perceptrons (MLP) that include convolutional layers for extracting feature vectors and fully connected layers. The actor network outputs the probabilities of different actions on the basis of the current state, whereas the critic

network evaluates the output of the actor. The actor then modifies its action output strategy according to the evaluation of the critic.

Algorithm 2 DSE based on RL

1: Initialize actor and critic networks
2: **for** $e = 1$ **to** E **do**
3: Initialize state S
4: **for** $t = 1$ **to** T **do**
5: Obtain the outputs of the actor $\pi_\theta(A|S)$ and the critic $V_\omega(S)$
6: Select an action A_t with the ϵ-greedy strategy
7: Obtain reward R_{t+1}
8: Obtain the next state S_{t+1} based on the state transition function
9: **end for**
10: Check resources constraints
11: Update the parameters of the actor and critic network θ, ω
12: **end for**
13: **return** Pareto solutions inferred by the actor

The algorithm then iteratively explores the design space until it reaches the maximum number of episodes E. In each episode, the algorithm initializes the state S by randomly selecting a hardware resource from all possible resources to initially consider and learns the appropriate hardware resource allocation strategy based on time steps (lines 2–12). At time step t, the algorithm calculates $\pi_\theta(A|S)$ and $V_\omega(S)$ (line 5). π is the probability distribution of the possible actions under state S, and V is the value of state S, which is used to evaluate the state. These are provided by the actor network and critic network, respectively. Next, the algorithm takes action A_t at the current time step based on the probability distribution $\pi_\theta(A|S)$ (line 6). To introduce randomness, thereby enabling the algorithm to achieve symmetry between its exploration and exploitation of the design space, we use the ϵ-greedy strategy and the decaying exploration rate strategy to guide the action selection procedure. This means that the action with the highest π value is selected with a probability of $1 - \epsilon$ (exploitation); an action is randomly chosen from all possible actions (including the action with the highest π value) with a probability of ϵ (exploration). The decay exploration rate strategy sets a larger ϵ value during the early stage in order to encourage more exploration and gradually reduces ϵ as training progresses in order to increase the emphasis on exploitation. This ensures that all possible actions have a certain chance of being selected under state S, thus continuously exploring and exploiting the design space.

When action A_t is taken, the algorithm calculates the associated reward R_t and outputs the next state S_{t+1} based on the state transition function (line 7). The reward is critical for RL, as it serves as the optimization goal. A higher reward indicates a smaller and closer-to-optimal objective. The reward also evaluates the impact of the corresponding action, helping the actor learn the optimal resource allocation strategy. Therefore, designing an appropriate reward is crucial. Since the optimization goal of RL is to maximize the reward, while the goal of hardware DSE is to minimize the function $F = A \times latency + B \times power$, i.e., to maximize $F' = -F = -A \times latency - B \times power$, considering the user constraints, we define the reward as shown in (6).

$$R_t = \begin{cases} -A(latency - latency_{max}) - B(power - power_{max}), & t = T \\ 0, & 0 < t < T \end{cases} \tag{6}$$

where A and B are hyperparameters that represent the relative importance levels of different performance metrics. The larger the weight is, the more emphasis is placed on the corresponding objective during exploration, and vice versa. On the one hand, by defining the reward as the negative weighted sum of the latency and power values and introducing user constraints, we can map the optimization goal of hardware DSE to the optimization

goal of the RL algorithm. This also enables automatic constraint checking. By changing the weights of different objectives, the algorithm can find various trade-offs without violating the imposed constraints and achieve flexible multi-objective optimization. On the other hand, when $t < T$, the resources are not fully allocated, and setting the reward to 0 can minimize the unknown impact of allocating a resource.

For the next state, the algorithm needs to select a hardware resource from all possible resources as the next considered resources (line 8). Since different resources are relatively independent, we use the function $f(S_{t+1}|S_t) = \frac{1}{m}$ as the state transition function, where m is the number of resources yet to be allocated in state S_t. This ensures the independence of the state transitions, thus satisfying the requirement of relatively independent resources.

At the end of an episode, MCL is used to update the parameters of the actor and critic networks (line 11). MCL relies on learning from experience rather than on an environmental model (e.g., state transition functions and reward functions) to improve its policy. It is a model-free method that effectively leverages previous experiences, allowing the algorithm to quickly generate suitable solutions even in a large design space or in cases when the model is unclear. This enhances the efficiency of the algorithm. The parameters are updated by (7) and (8) only at the end of an episode.

$$\Delta\omega \propto \sum_{i=1}^{T} \delta_i \nabla_\omega V_\omega(S_i) \tag{7}$$

$$\Delta\theta \propto \sum_{i=1}^{T} \delta_i \nabla_\theta log \pi_\theta(A_i|S_i) \tag{8}$$

where δ_i is the cumulative advantage of the state, indicating the quality of the state. It is calculated as follows:

$$\delta_i = \gamma^{T-i} R_T - V_\omega(S_i) \tag{9}$$

After reaching the maximum number of episodes, the actor learns the optimal policy that maximizes the reward. Then, hardware configurations can be inferred by the actor in a short period. Pareto solutions are found among these configurations and returned (line 13).

5. Experiment

5.1. Experiment Setup

Benchmarks. To evaluate the DSE strategy proposed in this paper, we use three different network models as benchmarks: a vanilla RNN, an LSTM, and a GRU [20]. LSTM and the GRU are both representatives of RNN hardware accelerators. LSTM is an advanced variant of RNNs that address the issue of capturing long-term dependencies. Compared to standard RNN, LSTM models have proven to be more effective at retaining and utilizing information over longer sequences. The GRU is another variant of the RNN architecture that addresses the short-term memory issue. It offers a simpler alternative to LSTM with fewer tensor operations, allowing for faster training. Both architectures have their advantages and disadvantages and are suitable for different tasks. In this paper, all three models are character prediction networks, and both the inputs and outputs are one-hot encoded vectors of size 29×1 that represent characters. The hidden layer size is set to 100 for all the models. The RNN has 3 layers, including an input layer, a hidden layer, and an output layer. The LSTM has 8 components, including a forgetting gate, an input gate, an output gate, candidate memory, 2 memory cell outputs, a hidden layer output, and an output layer output. The GRU has 7 components, including a reset gate, 3 candidate hidden states, an update gate, a hidden layer output, and an output layer output.

Methodology. We compare our strategy with a GA [15], PSO [18] and BO [19] from four perspectives, their performance metrics, hypervolumes (HVs), solution speeds, and performance metric trade-offs, to evaluate and illustrate the performance of our design. The performance metrics are the optimization objectives in this paper, i.e., the latency, power, and area of the hardware accelerator. The HV measures the overall performance of

an algorithm, including the quality and diversity of its solutions. The performance metric trade-offs illustrate the algorithms' ability to adjust their solutions under imposed user constraints. Experiments are performed on a Linux host with an Intel Core i7-10700 CPU (2.90 GHz), Nvidia RTX 2060 GPUs and 16 GB memory. As for the parameters of the algorithms, the size of population is set to 15 for GA. The cognitive coefficient and social coefficient of PSO are 1.5 and 1.0, respectively. BO uses 5 samples as its prior.

Metrics. This paper uses the open-source Maestro microarchitecture [21] to model the accelerator based on scratchpad memory, local PE memory, PE arrays, and PE interconnections. Maestro analyzes data reuse in terms of the temporal, computational, and memory aspects to evaluate the latency, power, and area of the accelerator.

5.2. Experimental Results and Analysis

5.2.1. Performance Metrics Evaluation

To demonstrate the capabilities of GRMD, it is compared with the GA, PSO, and BO. The GA and PSO are metaheuristic-based approaches that are widely applicable to many DSE problems. Specifically, the GA is an evolutionary approach. PSO makes use of swarm intelligence. BO uses a Gaussian process (GP) to estimate the objective function. Figure 5 shows the accelerator latency, power, and area values of different RNN models. The area values of most solutions are concentrated within the range of $[7.25 \times 10^6, 7.75 \times 10^6]$. GRMD can always provide solutions with lower latency and power for the three RNN models when the area differences are not significant. Specifically, compared with those produced by the GA, PSO, and BO for the vanilla RNN, the latency is reduced by averages of 5.19%, 9.35%, and 3.30%; the power is reduced by 2.63%, 4.98%, and 1.91%; and the area is reduced by 6.93%, 11.08%, and 2.34%, respectively. For LSTM and the GRU, which have more components, GRMD is generally better than the other three approaches in terms of power and area, with power reductions of 2.77%, 5.34%, and 2.00% for LSTM and 3.71%, 5.03%, and 2.39% for the GRU, respectively. GRMD also provides area reductions of 8.59%, 11.19%, and 2.56% for the LSTM and 11.95%, 11.18%, and 1.69% for the GRU, respectively. The minimum values are also lower than those given by the GA and PSO. This indicates that for the three different RNNs, GRMD can provide solutions that are superior to those of the other three algorithms. However, complex trade-offs are evident between the latency and area and between the latency and power. For example, when the number of PEs and the number of scratchpad banks possessed by the accelerator increase, the degree of parallelism improves, leading to a significant reduction in latency but also a substantial area increase. Conversely, when the area decreases, the latency increases. This is determined by the characteristics of the hardware design space, where resources are often independent but collectively influence the performance of the accelerator, making it challenging to provide a nondominated solution.

These results demonstrate the great potential of RL for use in hardware DSE tasks. GRMD interacts with hardware resources and performance metrics to gradually learn the impacts of resources on the resulting performance. It uses actor and critic networks to approximate the policy and value functions during the hardware resource allocation process. By balancing the exploration and exploitation schemes, GRMD actively learns suitable resource allocation strategies, reducing the uncertain impacts of various allocations; thus, it can provide better solutions than those of the competing methods. In contrast, the GA assumes that the offspring of two individuals in a population are generally better than their parents, which is not the case in hardware DSE, leading to reduced algorithmic effectiveness. Similarly, PSO is also a population-based stochastic optimization technique, and the main difference between PSO and the GA is that PSO does not have evolution operators. During the iterative process, the velocities and directions of the particles are adjusted in accordance with their own historical optimal positions and that of the entire population. However, this can lead to particles relying too much on historical optimal solutions, causing them to become stuck in local optima, especially in hardware design spaces, which are high-dimensional. BO uses models such as the stochastic process and

GP to estimate the objective function, and its accuracy depends on the chosen model and the input prior information. However, in the hardware design space, the joint impact of different resources on the resulting performance is not clear; thus, BO is not always reliable.

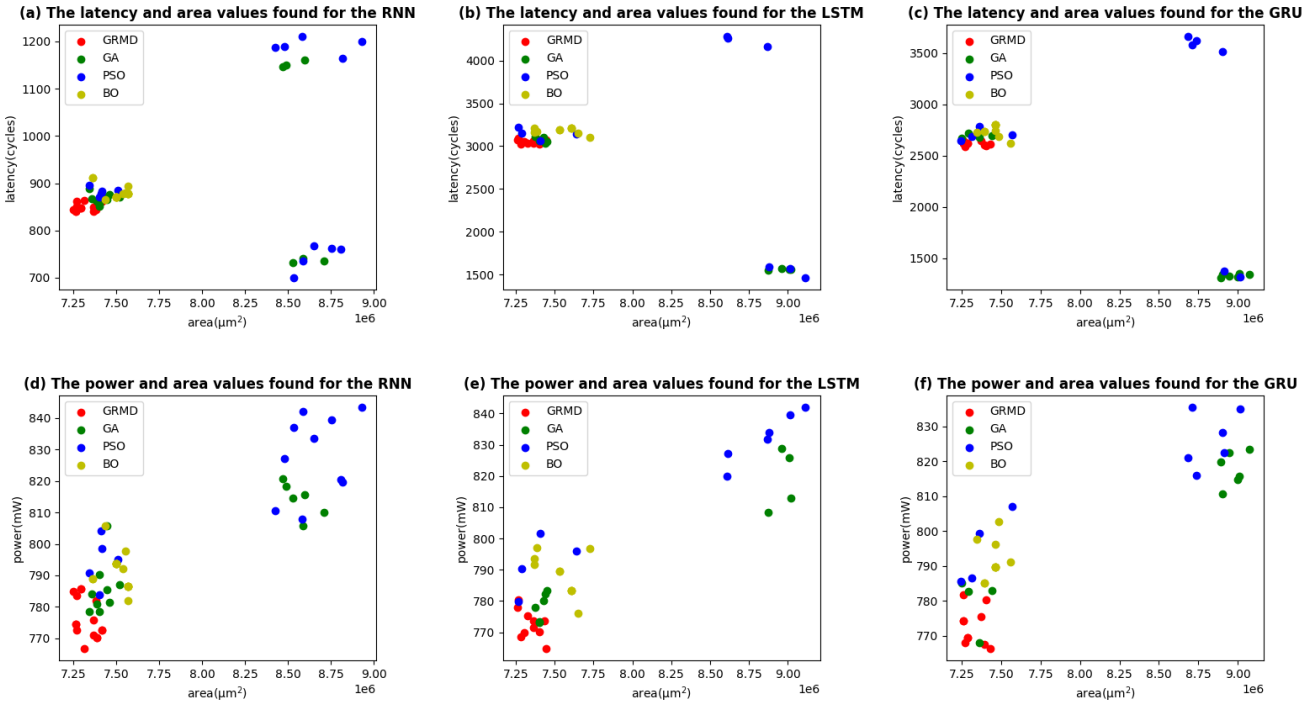

Figure 5. Solutions found by GRMD, the GA, PSO, and BO for the vanilla RNN, LSTM, and the GRU.

To summarize, for the three different RNNs, GRMD can provide solutions that are superior to those of the other three algorithms, achieving latency, power, and area reductions of 9.35%, 5.34%, and 11.95%, respectively. This fully demonstrates the effectiveness of GRMD in hardware DSE. Users can obtain hardware accelerators with lower latency, lower power, and smaller area.

5.2.2. Overall Algorithmic Performance Evaluation

We then compute the HVs and runtimes of the four algorithms to demonstrate their overall performance. In MOPs, the Pareto optimal accelerator latency, power, and area solutions form the Pareto front in space. When a solution moves along the Pareto front toward a reduction in one metric, at least one other metric increases. By selecting a reference point away from the front, a space between this point and the solutions is formed. The smaller the latency, power, and area are, the closer the corresponding solution is to the front, and the larger the space. We use the HV to represent the size of this space. The HV measures the overall performance of an algorithm, including the quality and diversity of its solutions. Table 2 shows the HVs and runtimes yielded by the four algorithms for three RNNs. Compared with the GA, PSO, and BO, GRMD loses a small amount of HV, but its runtime is significantly reduced. Specifically, the HV is reduced by averages of 6.33%, 6.32%, and 0.67%; the runtime is reduced by averages of 18.11%, 14.94%, and 10.28%, respectively.

The HV is influenced not only by the distance from the design points to the Pareto front but also by the distribution distances between the design points, i.e., the diversity of the solutions. The solutions generated by the GA have greater diversity because this algorithm actively introduces more randomness through evolution operators, allowing more design points to be explored, whereas GRMD uses RL to learn hardware resource allocation strategies, leading to less randomness when the neural network generates solutions after completing the training process, which translates to lower diversity. PSO relies on the historical optimal position to adjust the velocities and directions of the particles,

which can result in them becoming trapped in local optima. However, GRMD requires less time to generate better solutions because once trained, the neural networks used by the RL algorithm can effectively utilize previous experience and infer solutions in a few seconds without needing to re-explore and re-evaluate the space, significantly reducing the solution time. Furthermore, GRMD uses MCL methods during training, making it model-independent and capable of quickly generating solutions even when the model changes, ensuring high solution quality. It is important to note that upon having the Pareto front, it is not obligatory to choose one solution from it because GRMD aims to provide diverse solutions for users to choose from. The selection of solutions depends on the specific use case and problem at hand. For example, smart homes represent a typical low-power scenario, while autonomous driving has strict constraints on latency.

In general, compared to GA, PSO, and BO, GRMD significantly reduces runtimes by averages of 18.11%, 14.94%, and 10.28% due to the RL algorithm, thereby decreasing time costs for users. Additionally, GRMD generates solutions more efficiently with an acceptable loss of diversity on different models. This allows users to more easily apply GRMD to other models.

Table 2. HVs and runtimes of GRMD, the GA, PSO, and BO with the vanilla RNN, LSTM, and the GRU.

Benchmarks	Hypervolume ($\times 10^{15}$)				Running Time (s)			
	GRMD	GA	PSO	BO	GRMD	GA	PSO	BO
RNN	5.5	5.6	5.6	5.5	1856.0	2194.1	2071.6	1943.3
LSTM	4.5	5.0	5.1	4.6	3779.7	4689.4	4488.3	4271.8
GRU	4.8	5.2	5.2	4.8	3085.5	3766.9	3693.7	3505.1

5.2.3. Multi-Objective Trade-Off Evaluation

As mentioned before, the trade-offs between different optimization objectives are complex. We thus compare the abilities of RL and the GA to form a trade-off between latency and power. In most cases, a change in either latency or power will result in an opposite change in the other. For example, when the number of PEs increases, the computing power of the accelerator improves, leading to a significant reduction in latency but also a substantial power increase. Therefore, different objectives have different importance depending on the specific use case. By setting different weights for the latency and power parameters, we aim to achieve latency-first and power-first exploration. Ideally, as the weight of latency increases, the latency values of the solutions decrease, and vice versa. The same principle applies to power. The upper part of Figure 6 shows the latency and power values yielded by GRMD for the three RNN models. When exploring the LSTM and GRU, the latency-first solutions are concentrated in the lower right of the space, representing lower latency but higher power, whereas the power-first solutions are concentrated in the upper left, denoting lower power but higher latency. The lower part of Figure 6 shows the latency and power values yielded by the GA for the different RNNs. Compared with the latency-first solutions, the power-first solutions generally have much lower latency values but higher power values, and the distribution of the solutions does not reflect the impact of the weights on the resulting latency and power.

This finding indicates that when assigning different importance values to performance metrics, the proposed DSE strategy can appropriately balance latency and power values, whereas the GA does not effectively utilize the trade-offs between multiple objectives. This is due to the RL algorithm used in GRMD, which learns the relationship between resource allocation and accelerator performance through its actor and critic networks, achieving multi-objective trade-offs under imposed user constraints. When GRMD explores the vanilla RNN, the distribution of its solutions shows that the weights do not significantly impact the latency and power. To explain this, we conduct ablation experiments to study the impacts of different accelerator resource values on the achieved performance. The

results are shown in Table 3. Under the area-first scenario, increasing the capacity of the scratchpad in Group 2 or increasing the DMA bandwidth in Group 6 does not significantly reduce the latency or power. This is because, with the vanilla RNN as the baseline, the latency and power reach their optimal values when errors are considered. Our design finds optimal or approximately optimal solutions, so the latency and power cannot decrease significantly further. Therefore, the impact of the weights on the trade-offs between metrics is minimal. This further verifies the performance of the hardware DSE strategy proposed in this paper.

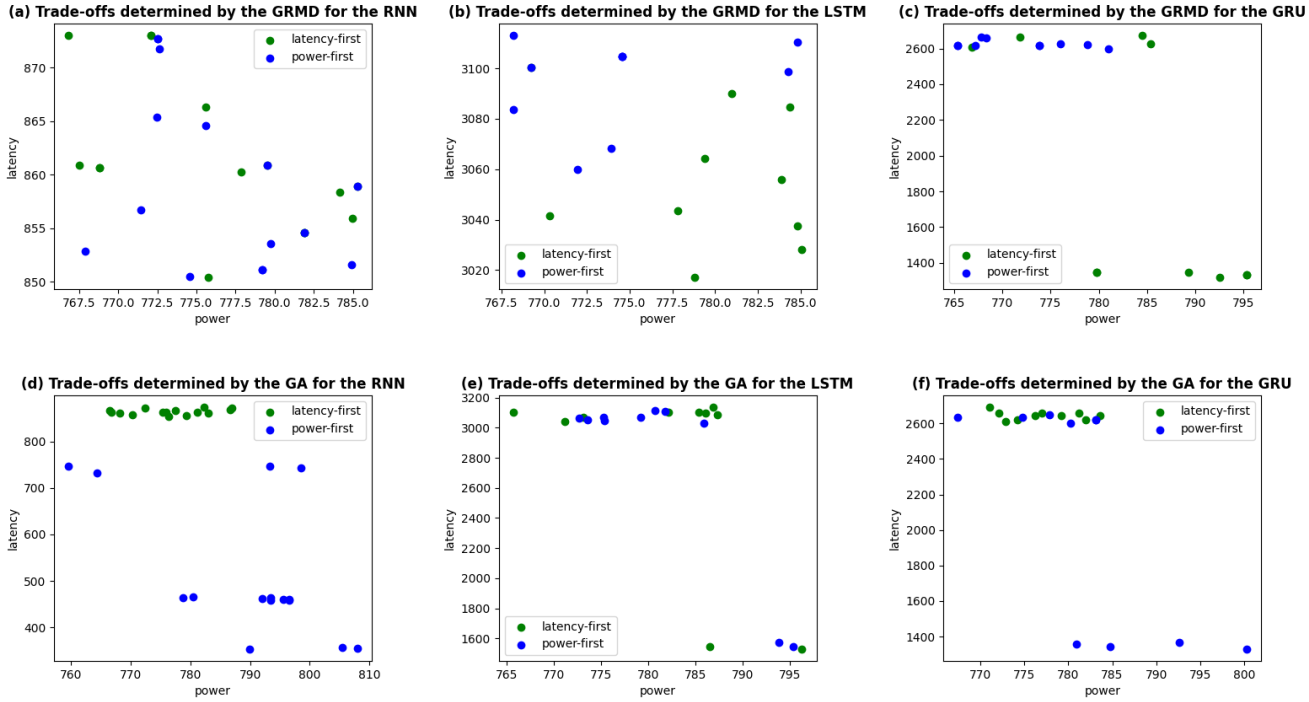

Figure 6. Trade-offs between the latency and power values determined by the GRMD and the GA.

Table 3. Impacts of hardware resource values on accelerator performance.

Hardware Resource		Group							
		Control Group	1	2	3	4	5	6	7
x		8	32	8	8	8	8	8	8
sp_capacity		128	128	512	128	128	128	128	128
local_capacity		64	64	64	256	64	64	64	64
dataflow		WS	WS	WS	WS	OS	WS	WS	WS
sp_banks		1	1	1	1	1	8	1	1
dma_buswidth		64	64	64	64	64	64	128	64
dma_maxbytes		64	64	64	64	64	64	64	128
Metrics	latency	850.92	249.41	873.69	869.06	1172.97	165.59	853.42	852.46
	power	784.77	19,741.69	768.12	771.16	806.43	863.43	785.52	777.26
	area ($\times 10^6$)	7.27	7.78	7.33	7.44	8.43	18.28	7.28	7.44

In conclusion, GRMD effectively balances the latency and power under varying weight settings. When exploring vanilla RNNs, the influence of weight on latency and power is minimal, indicating that our design reaches an optimal solution. GRMD provides users with more flexibility in real-world application scenarios.

6. Conclusions

This paper proposes a DSE strategy for customized RNN accelerators. The strategy first uses a GA to partially explore the design space, with the accelerator area as the objective. Then, it uses RL to find the optimal hardware configuration with latency and power as its objectives, taking the narrowed design space as the input. The proposed strategy can generate better solutions than those of competing methods and can achieve latency, power, and area reductions of 9.35%, 5.34%, and 11.95%, respectively, demonstrating both the effectiveness and efficiency of the hardware resource allocation process. Additionally, it can make reasonable trade-offs between multiple objectives, offering great flexibility. In terms of scalability, GRMD can be used to improve DSE for other complex neural networks that have similar structures or computations. In future studies, we aim to explore how other advanced RL techniques could enhance the GRMD strategy, such as multi-agent system, hierarchical RL, and inverse RL. Additional comparison with recent methods employing hybrid optimization approaches (e.g., a hybrid PSO and GA, a hybrid BO and RL) are also required to show GRMD's performance relative to these methods.

Author Contributions: Conceptualization, Q.L., J.X. and J.W.; methodology, Q.L.; software, Q.L.; validation, Q.L., J.X. and J.W.; formal analysis, Q.L.; investigation, J.X. and J.W.; resources, J.X. and J.W.; data curation, Q.L.; writing—original draft preparation, Q.L.; writing—review and editing, Q.L., J.X. and J.W.; visualization, Q.L.; supervision, J.X. and J.W.; project administration, J.X. and J.W.; funding acquisition, J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Tianjin Natural Science Foundation (No. 23JCYBJC01770) and the National Defense 173 Project (No. 2021JCJQJJ0470).

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

References

1. Elman, J.L. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211. [CrossRef]
2. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 735–1780. [CrossRef] [PubMed]
3. Cho, K. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv* **2014**, arXiv:1406.1078. [CrossRef]
4. Cho, H.; Lee, J.; Lee, J. FARNN: FPGA-GPU Hybrid Acceleration Platform for Recurrent Neural Networks. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1725–1738. [CrossRef]
5. Zaghloul, Z.S.; Elsayed, N. The FPGA Hardware Implementation of the Gated Recurrent Unit Architecture. In Proceedings of the SoutheastCon 2021, Atlanta, GA, USA, 10–14 March 2021; pp. 1–5. [CrossRef]
6. Wasef, M.; Rafla, N. SoC Reconfigurable Architecture for Implementing Software Trained Recurrent Neural Networks on FPGA. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 2497–2510. [CrossRef]
7. Genc, H.; Haj-Ali, A.; Iyer, V.; Amid, A.; Mao, H.; Wright, J.; Schmidt, C.; Zhao, J.; Ou, A.; Banister, M. et al. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv* **2019**, arXiv:1911.09925.
8. Pacini, T.; Rapuano, E.; Fanucci, L. FPG-AI: A Technology-Independent Framework for the Automation of CNN Deployment on FPGAs. *IEEE Access* **2023**, *11*, 32759–32775. [CrossRef]
9. Jiang, J.; Jiang, M.; Zhang, J.; Dong, F. A CPU-FPGA Heterogeneous Acceleration System for Scene Text Detection Network. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 2947–2951. [CrossRef]
10. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; pp. 161–170. [CrossRef]
11. Motamedi, M.; Gysel, P.; Akella, V.; Ghiasi, S. Design space exploration of FPGA-based Deep Convolutional Neural Networks. In Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Macao, China, 25–28 January 2016; pp. 575–580. [CrossRef]
12. Yang, S.; Bhattacharjee, D.; Kumar, V.B.; Chatterjee, S.; De, S.; Debacker, P.; Verkest, D.; Mallik, A.; Catthoor, F. AERO: Design Space Exploration Framework for Resource-Constrained CNN Mapping on Tile-Based Accelerators. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2022**, *12*, 508–521. [CrossRef]

13. Ahmed, M.R.; Koike-Akino, T.; Parsons, K.; Wang, Y. AutoHLS: Learning to Accelerate Design Space Exploration for HLS Designs. In Proceedings of the IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS), Tempe, AZ, USA, 6–9 August 2023; pp. 491–495.
14. Xiao, Q.; Zheng, S.; Wu, B.; Xu, P.; Qian, X.; Liang, Y.HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation. In Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 14–19 June 2021; pp. 1055–1068.
15. Mei, L.; Houshmand, P.; Jain, V.; Giraldo, S.; Verhelst, M. ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators. *IEEE Trans. Comput.* **2021**, *70*, 1160–1174. [CrossRef]
16. Zaourar, L.; Chillet, A.; Philippe, J.-M. A-DECA: An Automated Design Space Exploration Approach for Computing Architectures to Develop Efficient High-Performance Many-Core Processors. In Proceedings of the 26th Euromicro Conference on Digital System Design (DSD), Golem, Albania, 6–8 September 2023; pp. 756–763.
17. Xie, Z.; Dai, K.; Wu, Z.; Wang, J.; Lu, X.; Liu, S. Design Space Exploration of CNN Accelerators based on GSA Algorithm. In Proceedings of the 9th International Conference on Signal and Image Processing (ICSIP), Nanjing, China, 12–14 July 2024; pp. 319–323.
18. Raj, A.; Punia, P.; Kumar, P. A novel hybrid pelican-particle swarm optimization algorithm (HPPSO) for global optimization problem. *Int. Syst. Assur. Eng. Manag.* **2024**, *15*, 3878–3893. [CrossRef]
19. Kuang, H.; Cao, X.; Li, J.; Wang, L. HGBO-DSE: Hierarchical GNN and Bayesian Optimization based HLS Design Space Exploration. In Proceedings of the 2023 International Conference on Field Programmable Technology (ICFPT), Yokohama, Japan, 11–14 December 2023; pp. 106–114.
20. Shiri, F.M.; Perumal, T.; Mustapha, N.; Mohamed, R. A comprehensive overview and comparative analysis on deep learning models: CNN, RNN, LSTM, GRU. *arXiv* **2023**, arXiv:2305.17473.
21. Kwon, H.; Chatarasi, P.; Pellauer, M.; Parashar, A.; Sarkar, V.; Krishna, T. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In Proceedings of the International Symposium on Microarchitecture, Columbus, OH, USA, 12–16 October 2019; pp. 754–768.