applied sciences

MDPI

Article

A Scheduling Method for Heterogeneous Signal Processing Platforms Based on Quantum Genetic Algorithm

Yudong Li, Jinquan Ma *, Zongfu Xie, Zeming Hu, Xiaolong Shen and Kun Zhang

Information System Engineering College, PLA Strategic Support Force Information Engineering University, Zhengzhou 450001, China; 15248073482@163.com (X.S.); 13147978871@163.com (K.Z.)
* Correspondence: ma7q@163.com

Abstract: Currently, many problems such as variable signal resources, complex execution environments, and low efficiency of scheduling algorithms are faced by heterogeneous signal processing platforms. The task scheduling algorithm is one of the key factors that directly affect the performance of the processing platform. In order to solve the problems of low efficiency of task scheduling algorithms and high computational cost of processors, a heterogeneous platform scheduling algorithm based on the quantum genetic algorithm is proposed in this paper. The algorithm constructs a task scheduling model by using a directed acyclic graph. This paper quantifies the mapping relationship between the quantum genetic algorithm and task scheduling. It corresponds qubits to binary, chromosomes to processor numbers, and individuals to processor scheduling strategies. In this paper, a new way of coding chromosomes using quantum coherence properties is designed to reduce the population size and increase population diversity. Crossover operations are performed on all individuals using full-interference crossover to avoid the results falling into local optimal solutions. The population of slow convergence is solved by implementing mutation operations on populations through quantum rotation gates. In addition, a task pre-ordering stage is designed based on the table scheduling algorithm. The task scheduling priority developed at this stage is used as the reference value for the initial encoding of the population, so that the search space for solutions is reduced. Finally, experiments are conducted using randomly generated task graphs. The algorithm is compared with improved genetic algorithms and existing intelligent scheduling algorithms. The results show that the algorithm can still obtain better results when the number of populations and iterations is small. It is more appropriate for heterogeneous platforms and computation-intensive tasks.

Keywords: heterogeneous platform; qubit; full-interference crossover; quantum rotation gate; quantum genetic algorithm; computation-intensive

Citation: Li, Y.; Ma, J.; Xie, Z.; Hu, Z.; Shen, X.; Zhang, K. A Scheduling Method for Heterogeneous Signal Processing Platforms Based on Quantum Genetic Algorithm. *Appl. Sci.* **2023**, *13*, 4428. https://doi.org/10.3390/app13074428

Academic Editor: Vincent Cicirello

Received: 7 March 2023
Revised: 28 March 2023
Accepted: 29 March 2023
Published: 30 March 2023

1. Introduction

The heterogeneous signal processing platform is a comprehensive processing platform that integrates processing units from a variety of different architectures. Due to its rich hardware and software resources, high degree of heterogeneity, parallelism, and efficiency, it is widely used in signal processing. How to rationally allocate tasks to processing units with different computational power to achieve the shortest schedule length is one of the key issues affecting the performance of heterogeneous systems. This problem has been proven to be NP-hard [1]. Currently, task scheduling techniques are mainly modeled by the directed acyclic graph (DAG), which simulates tasks with different communication dependencies and different computational requirements. However, with the rapid development of wireless communications and the Internet, the size of the signal continues to grow. As a result, the number and complexity of tasks after DAG abstraction is increasing. Therefore, designing a reasonable and effective scheduling algorithm will greatly improve the computational performance of the heterogeneous platform.

1.1. Research Status

According to literature studies, scheduling algorithms are mainly proposed based on heterogeneous computing systems. It has received widespread attention from the academic community for its ability to perform large-scale scientific computing, information processing, and data services at low cost and high efficiency. Scheduling algorithms can be categorized in a variety of ways, such as task scheduling procedures, types of goals, or mathematical models and research methodologies. Examples include random and non-random scheduling, static and dynamic scheduling, and scheduling with no deadline constraints and with deadline constraints [2]. Traditional scheduling algorithms are mainly list-based scheduling algorithms, clustering-based scheduling algorithms, and task replication-based scheduling algorithms. Heterogeneous earliest finish time (HEFT) [3] and dynamic level scheduling (DLS) [4,5] are the most representative. While traditional scheduling works well for smaller tasks and data handling, the rapid development of 5G and artificial intelligence makes traditional scheduling no longer applicable. Scheduling algorithms based on artificial intelligence are constantly being proposed. For example, to generate a predictive scheduling scheme for the early detection of faulty machines, Paprocka et al. [6] proposed an algorithm based on ant colony optimization. To improve task allocation schemes, Sanabria et al. [7] used deep reinforcement learning. Gao [8] introduced machine learning to heterogeneous computing systems, aiming to accelerate platform execution through neural networks and multilayer perceptron. Although AI-based scheduling algorithms have been extensively researched and applied in various scenarios, their drawbacks include not accurately reflecting the internal connections in the task graph and longer model training time. Due to the limitations of traditional scheduling algorithms and AI-based scheduling algorithms, combinatorial optimization algorithms have gained a lot of attention. This is because the strengths of several algorithms are used to complement one another. Among them, due to genetic algorithms' mature technology and ease of operation, they are widely used in multi-objective optimization, detection accuracy improvement, job shop scheduling, and cloud computing task scheduling [9–12].

In recent years, the problem that genetic algorithms tend to stick to locally optimal solutions and converge slowly has been investigated using quantum computing. Quantum genetic algorithm (QGA) is an improvement of traditional genetic algorithms that uses qubits, quantum coherence, quantum rotation gate, and more. For example, Chen et al. [13] used chaotic sequences to update the quantum rotation gate. Gandhi et al. [14] improved genetic algorithms based on revolving door refinement to improve scheduling efficiency in distributed systems. Alam et al. [15] improved the efficiency of bi-objective load balancing scheduling based on the quantum genetic algorithm. Konar et al. [16] improved quantum genetic algorithms using random key distribution to improve scheduling efficiency in a multi-processor environment. To speed up the convergence of the algorithm, Guo et al. [17] improved the population structure of the QGA based on small-world theory to improve the algorithm's performance. Teng et al. [18] improved the QGA by using mutative scale chaos optimization, which accelerated the algorithm's convergence speed and improved solution accuracy. Chang et al. [19] used cellular automaton to update the population in the quantum genetic algorithm. This improved the convergence and accuracy of the algorithm for multi-objective functions. Zhu et al. [20] introduced multi-subpopulations to improve the encoding stage of the QGA, which improved the algorithm's performance in optimizing benchmark functions. Ni et al. [21] improved the coding stage of the QGA by using the phases in the niche technique, which improved the algorithm's ability to optimize multi-peak functions. Zhao et al. [22] used QGA to optimize the radio parameters, obtaining a better solution early in evolution and reducing execution time. Pitchai et al. [23] used a discrete quantum walk to replace the cross-mutation process in the quantum genetic algorithm, improving the algorithm's ability to solve the 0-1 quadratic knapsack problem. The QGA is found to have higher performance and faster convergence through research. However, much of the research has only improved the QGA itself, or applied it to solving function optimization or knapsack problems. There has been very little application of the

algorithm to engineering practice. Based on this research, the paper applies the QGA to task scheduling of heterogeneous systems and proposes a scheduling method for heterogeneous signal processing platforms based on the quantum genetic algorithm (HPS-QGA).

1.2. Main Work and Contribution

We first modeled the task scheduling in heterogeneous signal processing platforms. Then the mapping relationship between the quantum genetic algorithm and scheduling strategies in the heterogeneous platform were established. Finally, the algorithm was tested in multiple groups using randomly generated task graphs to confirm the efficiency and applicability of the algorithm.

The main contributions of this paper are as follows:

1. Quantum bits are mapped to the "0" and "1" codes in the binary. We transform the task scheduling problem in the macro to the optimization problem in the micro. The chromosome encoded by quantum bits is mapped to the serial number of the processor, and each individual represents a processor scheduling strategy.
2. According to the quantum coherence, a new type of crossover is introduced—the "full- interference crossover". It achieves crossover operations for all individuals in the population and solves the problem of getting stuck in a local optimum solution.
3. Based on the data flow of the components in the task scheduling model, we have designed a task pre-sorting stage. The initial chromosome is encoded in conjunction with the task scheduling strategy in task preordering. It reduces the search space of the solution and improves performance.

1.3. Paper Organization

The rest of this article is organized as follows: Section 2 presents heterogeneous platform scheduling models. Section 3 details the methods for establishing the mapping relationship between quantum genetics and task scheduling, including operations such as encoding and decoding, cross mutation, etc. Section 4 presents the results of the comparison experiments using the HSP-QGA algorithm. Section 5 presents the conclusions, which include the limitations of the study and further work.

2. Platform Scheduling Model

The heterogeneous signal processing platform scheduling is greatly affected by the correlation between components, and the communication between heterogeneous processors, so its model mainly includes the task model and the target platform hardware model. A reasonable and effective scheduling model can greatly improve the performance of the target platform [24].

2.1. Heterogeneous Signal Processing Platform

The heterogeneous signal processing platform is a common platform consisting of modular, standardized, and universal hardware units connected by bus or switch. Various signal processing functions are implemented by loading reusable, portable, scalable, and easily upgradable standardized software modules on the platform. Heterogeneous signal processing platforms adopt common high-performance hardware platforms in hardware architecture, such as ATCA, CPCI, VPX, etc. A large number of different types of processors in the platform, including FPGA, DSP, GPU, GPP, etc. In the software architecture, we build a standardized and regulated hierarchical new software architecture. The difficulty of the cross-platform operation and portability of components in heterogeneous platforms is solved by shielding the differences between the underlying layers through system abstraction.

The software architecture of a heterogeneous signal processing platform is mainly divided into a hardware platform layer, an operating system layer, a driver abstraction layer, a core framework layer, a management service layer, and an application layer. The operating system layer and the driver abstraction layer belong to the operation support service layer, which can shield the underlying processing unit from hardware differences

in the operating system, communication, memory, and file operation, and provide a unified and standardized interface for the upper layer. The core framework layer is based on the container technology (including FPGA container, process container, thread container, etc.) to shield differences in processor task scheduling and implement application component-based scheduling services. The management service layer includes system operation management environment and visual development management to improve the visibility and portability of application development. The application layer includes the component library and the application console, which are responsible for completing the signal processing function. The software architecture of the heterogeneous signal processing platform is shown in Figure 1.

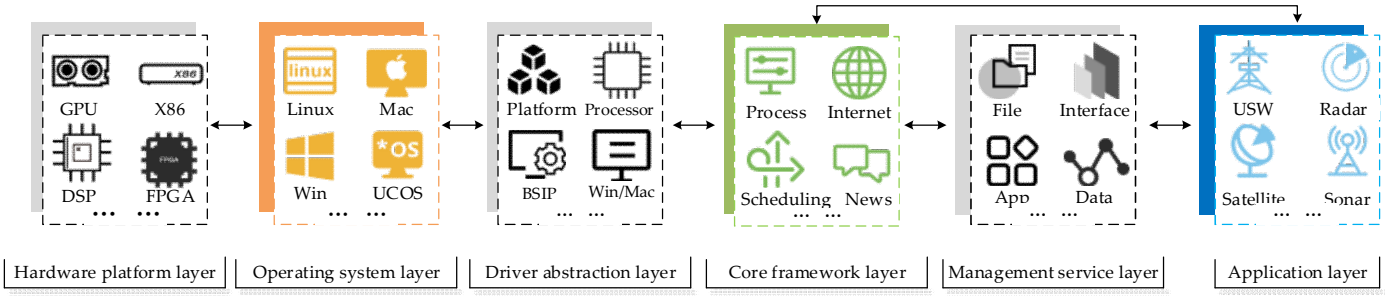

Figure 1. The software architecture of the heterogeneous signal processing platform.

2.2. Platform Task Model

DAG is usually used to describe the task scheduling model in heterogeneous platforms, where each node represents a task component and each directed edge represents a communication link between components. The platform task scheduling model is constructed as $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{W}, \boldsymbol{C}\}$, where $\boldsymbol{V} = \{v_1, v_2, v_3, \cdots, v_n\}$ is the set of all tasks in the scheduling processing system; $\boldsymbol{E} = \{e_{12}, e_{23}, \cdots, e_{ij}\}$ represents the set of communication links between associated tasks, $\boldsymbol{e_{ij}} = (v_i, v_j)$ represents the communication edge between tasks v_i and v_j, and v_i is the parent task of v_j; $\boldsymbol{W} = \{w_1, w_2, \cdots, w_n\}$ represents the set of computing costs for any executed task; $\boldsymbol{C}(v_i, v_j)$ represents the communication cost of data transmission between tasks. If any two tasks (v_i and v_j) are assigned to the same node (on the same processor), the communication cost $\boldsymbol{C_{ij}}$ is 0. Figure 2a shows a DAG with 10 components. Numbers in nodes represent tasks' serial numbers, orange numbers next to nodes represent tasks' computing overheads, and blue numbers on communication edges represent tasks' inter-task communication overheads.

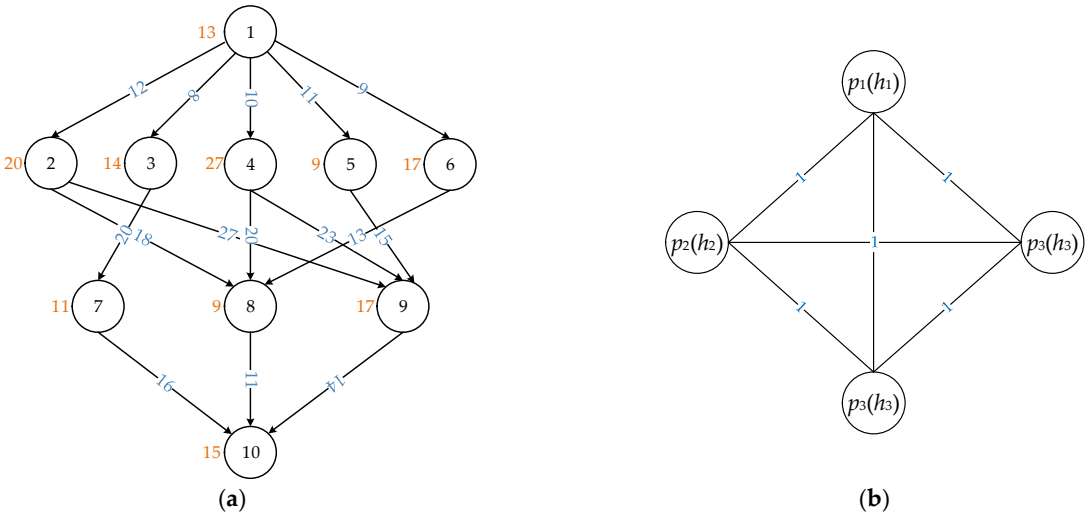

Figure 2. Platform scheduling model: (**a**) task scheduling model; (**b**) processing system model.

The hardware resource architecture is abstracted as $\boldsymbol{N} = \{\boldsymbol{P}, \boldsymbol{H}, \boldsymbol{R}, \boldsymbol{T}\}$ by an undirected graph, where $\boldsymbol{P} = \{p_1, p_2, \cdots, p_k\}$ is the set of processors; $\boldsymbol{H}$ is the feature of the processor, including the main frequency, cache, communication bandwidth of the processor; $\boldsymbol{R} = \{r_1, r_2, \cdots, r_g\}$ is the processing ability of processors; $\boldsymbol{T}$ is the communication bandwidth between processors. This paper simplifies the communication bandwidth between processors to be 1. Figure 2b shows a processing system model with four heterogeneous processors.

3. HSP-QGA Algorithm Design

3.1. Algorithm Analysis

The traditional genetic algorithm is based on Darwin's theory of biological evolution. It is a mathematical optimization technology based on a biological model that enables the evolution of populations through reproduction, variation, competition, and selection. The operation flow is shown in Figure 3.

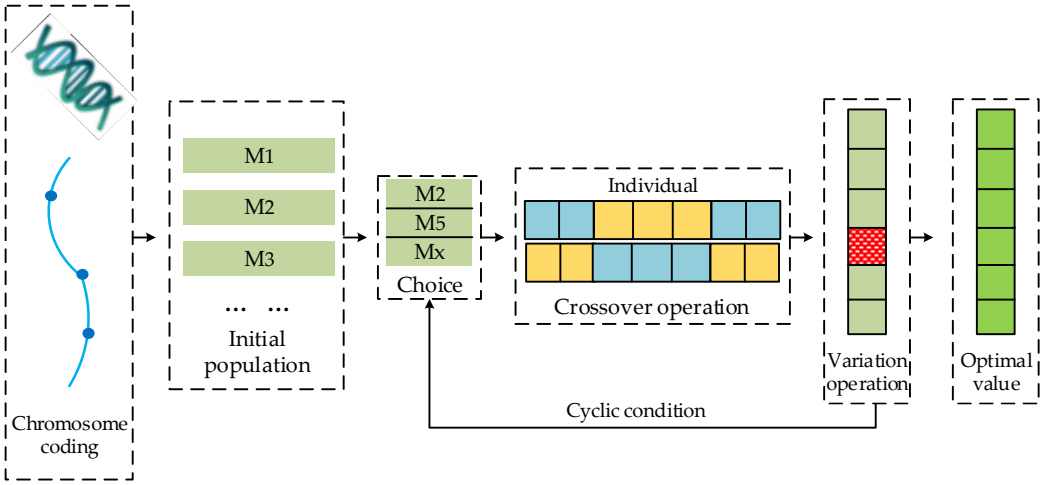

Figure 3. Diagram of classical genetic algorithm.

In order to improve the efficiency of heterogeneous platforms, this paper designs the HSP-QGA with improvements to the genetic algorithm using qubits and quantum rotation gates, and scenario adaptation. The core framework consists mainly of two parts: the task preordering phase and the quantum genetic optimization stage, shown in Figure 4.

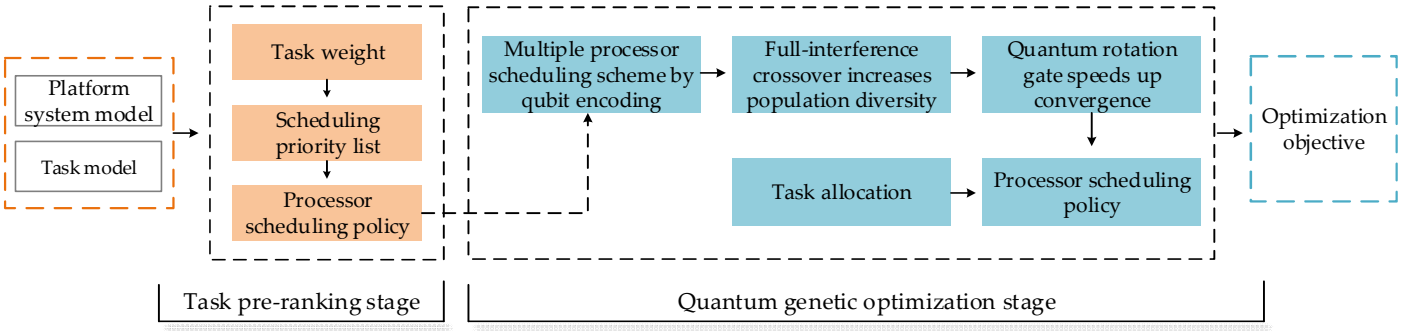

Figure 4. Design framework of HSP-QGA.

In the task pre-ranking stage, the rank of tasks in the table scheduling algorithm is used to recursively calculate the priority value of each task in the heterogeneous system, shown in Formula (1). Then all tasks are sorted in descending order according to the rank

value, and the scheduling strategy of the processor in the system is obtained by the table scheduling algorithm.

$$\text{rank}(v_i) = \overline{w_i} + \max_{v_j \in succ(v_i)} (\overline{c_{i,j}} + \text{rank}(v_j)) \quad (1)$$

where $\overline{w_i}$ represents the average computational cost of task v_i, $succ(v_i)$ represents the set of successor nodes (tasks) of v_i, $\overline{c_{ij}}$ represents the average communication cost between task v_i and v_j. The calculation method is as follows (2).

$$\overline{c_{ij}} = \left(\sum_{m=1}^{n-1} \sum_{s=m+1}^{n} c_{i,j}/q_{k,f} \right) / (p \times (p-1)/2) \quad (2)$$

where $q_{k,f}$ represents the number of communication paths between processors k and f, and p represents the number of processors.

For example, we can sort the tasks shown in Figure 2a to obtain the scheduling list shown in Table 1.

Table 1. Task scheduling priority sample table.

Task	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
Rank	94	64	45	67	70	65	46	35	42	15
Order	1	5	7	3	2	4	6	9	8	10

In the quantum genetic optimization stage, based on the initial scheduling policy of the processor obtained in the task pre-ordering phase, the initial scheduling policy is encoded by using qubits to obtain the initial individuals and population. This avoids the unpredictability of randomly generated initial populations and reduces the search space for understanding. The results are kept from falling into a local optimum by a full-interference crossover. The quantum rotation gate is used to make the population evolve in the direction of greater fitness, while accelerating the convergence of the solution. Finally, an approximate optimal solution is obtained with the objective of minimizing the earliest completion time in the solution space provided by this algorithm.

3.2. Algorithm Design

3.2.1. Chromosome Coding

In the HSP-QGA, the chromosomes of individuals are encoded by qubit. Unlike the binary, gray, and real number methods used in the traditional GA, the qubit is a two-state quantum system. It can be formed using a standard orthogonal basis $\{|0\rangle, |1\rangle\}$ [25] in addition to residing in the superposition state, thus increasing the amount of information carried by the chromosome, shown in Formula (3). In addition, "0" represents the spin-down state, "1" represents the spin-up state, "$|\rangle$" represents a quantum state, α, β represents probability amplitude pair, which exists in the plural, and $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ and $|\beta|^2$ represent the probabilities of "0" and "1" states, respectively.

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (3)$$

In this paper, the existing large and complex quantum coding approach is mapped in a reasonably simplified way [26]. Denote each chromosome as a processor serial number and each individual as a processor scheduling strategy. Each individual is composed of n chromosomes, and the number of chromosomes equals the total number n of tasks in the heterogeneous system. Each chromosome can be represented by m quantum genes, and each quantum gene is represented by $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ where $m = \left\lceil \log_2^p \right\rceil$, "$p$" means the number of processors in the system, and "$\lceil\ \rceil$" means rounding up. As shown in (4), S represents

a chromosome code consisting of m quantum genes. As shown in (5), $\boldsymbol{I}$ is described as a quantum-coded individual in which each chromosome is represented, as shown in (4).

$$S = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1m} \\ \beta_{11} & \beta_{12} & \cdots & \beta_{1m} \end{pmatrix} \tag{4}$$

$$I = (S_1\ S_2\ S_3 \cdots \cdots S_n) \tag{5}$$

3.2.2. Chromosome Decoding

As $|\alpha|^2$ and $|\beta|^2$ tend to 0 or 1, each quantum gene bit will converge to a single state. Thus each quantum gene can eventually tensor to 0 or 1, which can be abstracted into the "0" and "1" codes of the binary string in the mathematical model. In the decoding process, using this mapping relationship, each quantum gene is decoded as 0 or 1, and each chromosome is decoded into a binary string of length m. By converting the binary string to a real number so that each chromosome represents a real number, i.e., a processor serial number. Finally, a processor scheduling strategy of n length is obtained by fully decoding quantum individuals. According to this decoding method, all individuals in the population are decoded.

To better describe the problem of chromosome coding and scheduling strategies, we have mapped quantum genetic algorithms and task scheduling, as shown in Table 2. Let us assume that there are 4 processors, numbered from 0 to 3. The number of tasks is five, and the number of chromosomes is also five according to the chromosome coding process. The number of quantum genes per chromosome is $m = \left\lceil \log_2^4 \right\rceil = 2$. $\begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \beta_{11} & \beta_{21} \end{pmatrix}$ denotes a chromosome encoded by 1 quantum bit. Based on the criterion of $|\alpha|^2 + |\beta|^2 = 1$, the initial value of $(\alpha_i, \beta_i)^T$ can be set to $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)^T$. The bit coding is converted to binary coding according to some rules (e.g., generate a random number in the interval [0, 1] and assign α to 1 if the number is greater than α^2). In a binary string, every two neighboring numbers are converted into real numbers. In the end, two processor scheduling policies can be generated. For example, scheduling policy 1 is to assign tasks 1–5 to processors with serial numbers of 1, 3, 1, 0, 2, respectively. With quantum bit coding, we can also see that two scheduling strategies are generated. Compared to traditional genetic algorithms, this approach increases the diversity of the population.

Table 2. Mapping table between Quantum codec and task scheduling.

Method of Operation	**Method of Encoding**
Qubit encoding	$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{21} & \alpha_{22} & \alpha_{31} & \alpha_{32} & \alpha_{41} & \alpha_{42} & \alpha_{51} & \alpha_{52} \\ \beta_{11} & \beta_{12} & \beta_{21} & \beta_{22} & \beta_{31} & \beta_{32} & \beta_{41} & \beta_{42} & \beta_{51} & \beta_{52} \end{pmatrix}$
Binary encoding	$\begin{pmatrix} 0\,1\,1\,1\,0\,1\,0\,0\,1\,0 \\ 1\,0\,0\,0\,1\,0\,1\,1\,0\,1 \end{pmatrix}$
Real to binary conversion	$\begin{pmatrix} 1 & 3 & 1 & 0 & 2 \\ 2 & 0 & 2 & 3 & 1 \end{pmatrix}$
Scheduling strategy 1	1 3 1 0 2
Scheduling strategy 2	2 0 2 3 1

3.2.3. Full-Interference Crossover

The crossover operator in the traditional genetic algorithm is limited to two individuals. When two individuals are the same, the crossover operation no longer works, and the population diversity decreases. In this paper, a new method: full-interference crossover is designed using quantum coherence. By crossover of each chromosome (processor serial number) in the population, chromosome variation is increased, and results are prevented from falling into the local optimum. A simplified example is as follows:

There is a population of five individuals, each with five chromosomes, where I (1)~I (5) represents the first individual (a processor scheduling strategy), I (1) represents the first chromosome (processor serial number) in the individual, and other individual manifestations are sub-analogous. The full-interference crossover operation of the population is shown in Table 3. The first chromosome of each individual remains unchanged. Starting from the second chromosome, the corresponding chromosomes of all individuals turn downwards to achieve full-interference crossover operation.

Table 3. Example of full-interference crossover operation.

Example	Individual	Chromosome				
		First	**Second**	**Third**	**Fourth**	**Fifth**
Before full interference crossover	1	I (1)	I (2)	I (3)	I (4)	I (5)
	2	II (1)	II (2)	II (3)	II (4)	II (5)
	3	III (1)	III (2)	III (3)	III (4)	III (5)
	4	IV (1)	III (2)	IV (3)	IV (4)	IV (5)
	5	V (1)	V (2)	V (3)	V (4)	V (5)
Afore full interference crossover	1	I (1)	V (2)	IV (3)	III (4)	II (5)
	2	II (1)	I (2)	V (3)	IV (4)	III (5)
	3	III (1)	II (2)	I (3)	V (4)	IV (5)
	4	IV (1)	III (2)	II (3)	I (4)	V (5)
	5	V (1)	IV (2)	III (3)	II (4)	I (5)

To illustrate and compare the role of the full interference crossover operation in the overall algorithm, we visualize the output of the scheduling length for each step in the algorithm. A scatter plot of the solutions after the full-interference crossover operation in each generation is shown in Figure 5. The full-interference crossover operation increases the diversity of the population and provides a sufficiently large solution space at the beginning of the iteration. A scatter plot of the nodes of the full flow of the algorithm is shown in Figure 6. Based on the solution space provided by the full- interference crossover operation, the algorithm accelerates the mutation (evolution) of the population toward the optimal solution using a quantum rotation gate. The algorithm converges after the 25th iteration. The full-interference crossover and variance converge to stable values and no longer play a role in the algorithm.

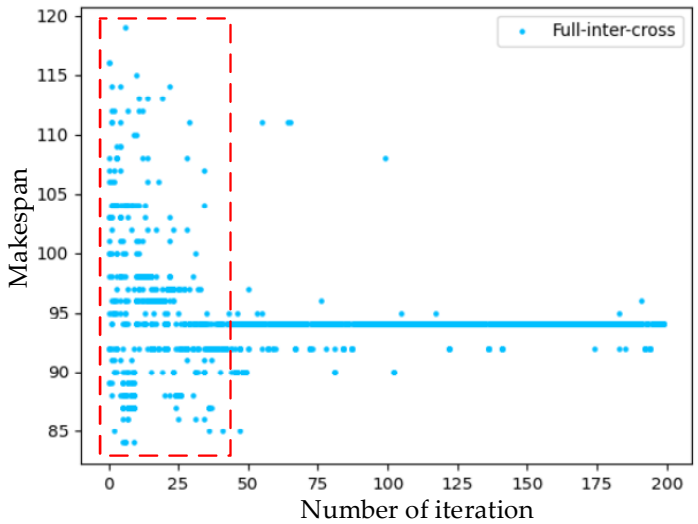

Figure 5. Scatter plot of the solution after full interference crossover operation.

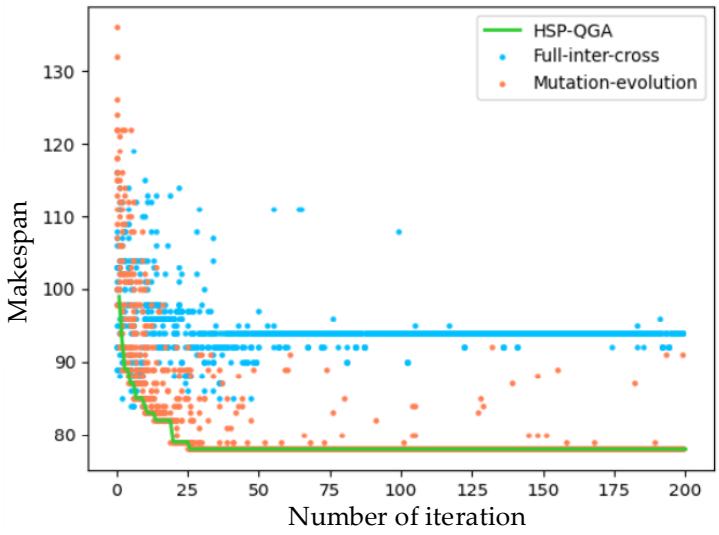

Figure 6. Scatter plot of the nodes of the full flow of the algorithm.

3.2.4. Variation Operations

The coding and decoding process shows that each chromosome after decoding (processor serial number) corresponds to each chromosome before decoding (qubit matrix). A mutation operation on the decoded chromosome is equivalent to a mutation operation on the qubit matrix before decoding. Because of this correspondence, the quantum rotation gate [27] is introduced to mutate the gene bits (qubits) of the chromosome before decoding. In contrast to the randomness generated by the variation operation of the traditional genetic algorithm, the main function of the quantum rotation gate is to make the population mutate to the current optimal individual obtained. In this way, the optimal solution can be obtained and the speed of convergence increased. The rotation direction is shown in Figure 7.

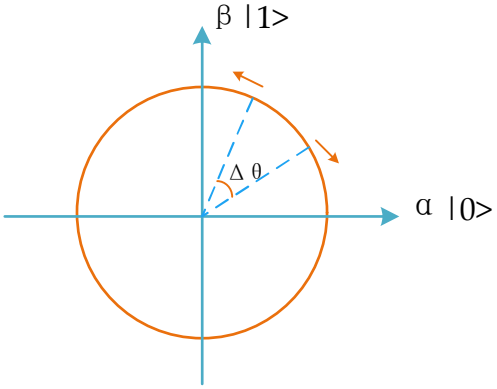

Figure 7. Rotation direction of rotation gate.

where $\Delta\theta$ represents the directional rotation angle, α and β represent the direction of 0 and 1 states in the qubit, corresponding to 0 and 1 in the binary string, respectively. The calculation formula of the quantum rotation gate $U(\theta_i)$ is shown in (6). Where θ_i represents the vector rotating to α or β, and the calculation method is shown in (7). The updated formula of the qubit $\begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix}$ is shown in (8).

$$U(\theta_i) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \tag{6}$$

$$\theta_i = s(\alpha_i, \beta_i) \times \Delta\theta_i \tag{7}$$

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \mathrm{U}(\theta_i)\begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix}\begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \tag{8}$$

In Formula (7), where $s(\alpha_i, \beta_i)$ is the direction of rotation and $\Delta\theta_i$ is the angle of rotation, their values are determined according to the calculation rules in Table 4 [28].

Table 4. Calculation rule table of rotation gate.

$x_{(s,i)}$	$x_{best(s,i)}$	$f(x) > f(x_{best})$	$\Delta\theta_i$	$s(\alpha_i,\beta_i)$			
				$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	NO	0	0	0	0	0
0	0	YES	0	0	0	0	0
0	1	NO	0.04π	+1	−1	0	±1
0	1	YES	0.04π	−1	+1	±1	0
1	0	NO	0.04π	−1	+1	±1	0
1	0	YES	0.04π	+1	−1	0	±1
1	1	NO	0	0	0	0	0
1	1	YES	0	0	0	0	0

In Table 4, $x_{(s,i)}$ is the i-th bit of the binary string encoded by the s-th chromosome (qubit-coded) of the current individual, $x_{best(s,i)}$ is the i-th bit of the binary string corresponding to the s-th chromosome of the best individual. $f(x)$ is the fitness function that represents the reciprocal of the system scheduling length corresponding to the processor scheduling strategy, as shown in (9).

$$f(x) = 1/\mathrm{EFT}(v_{exit}, p_k) \tag{9}$$

where the independent variable x is the matrix of dimension $t \times u$, t represents the number of chromosomes, u represents the coding length of each chromosome; v_{exit} is the last task in the scheduling system; p_k is the processor assigned by v_{exit}, k is the real number converted from binary to decimal in decoding, representing the processor number, as shown in (10).

$$k = \sum_{i=0}^{m-1} b_i \times 2^i \tag{10}$$

Table 4 shows that the fitness value $f(x)$ of the current individual is compared with the fitness value $f(x_{best})$ of the best individual in the population, if $f(x) > f(x_{best})$, then (α_i, β_i) evolves in a direction that favors the appearance of $x_{(s,i)}$; on the contrary, (α_i, β_i) evolves in the direction that favors the appearance of $x_{best(s,i)}$.

3.3. HSP-QGA Algorithm

The flow chart of the heterogeneous signal processing platform task scheduling algorithm based on the quantum genetic algorithm is shown in Figure 8.

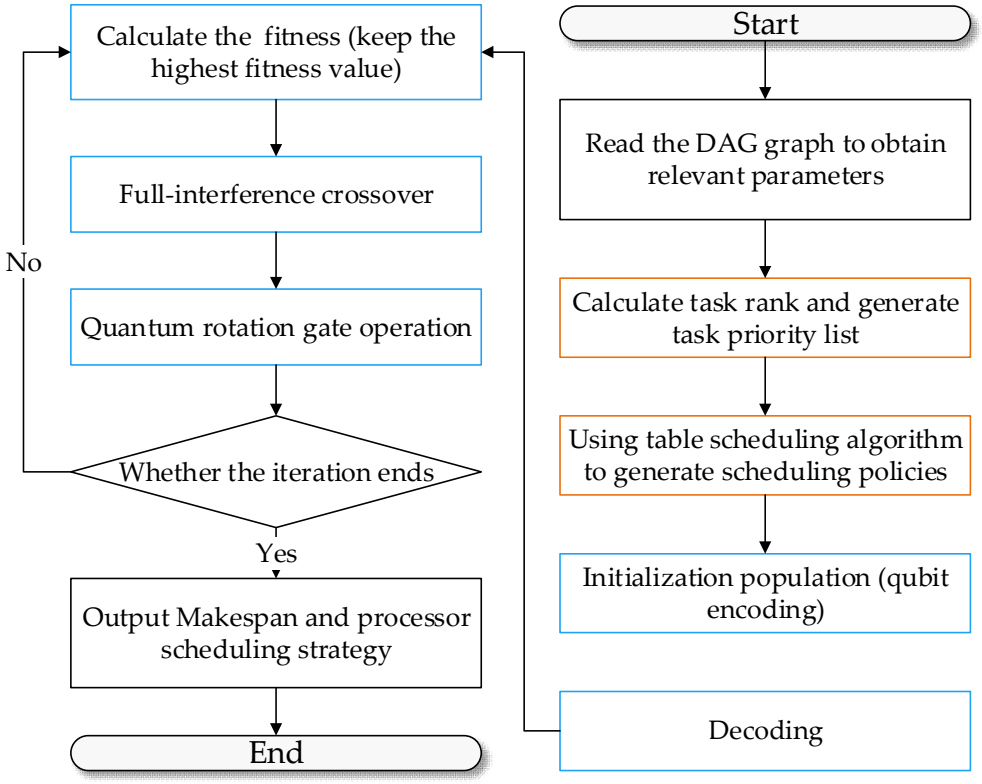

Figure 8. Flow chart of the HSP-QGA.

The main operation flow of the HSP-QGA (Algorithm 1).

Algorithm 1: Main operation process of the HSP-QGA.

Input: DAG graph, population size, iteration times, and binary length
Output: Makespan, speedup, efficiency, and scheduling policy
1: Calculate the rank of each task
2: Generate processor scheduling strategy
3: Population initialization
4: While i < Iteration times:
5: decoding the chromosomes of each individual in the population
6: decode qubits into binary strings according to the scheduling strategy
7: calculate the corresponding real number according to the binary string
8: form a decoded population
9: calculate the fitness of each individual
10: if fitness(current individual) > fitness(best):
11: fitness(best) = fitness(current individual)
12: full-interference crossover operation
13: quantum rotation gate operation
14: $i \pm 1$
15: Record the best fitness in each cycle
16: end.

4. Simulation Experiment and Results Analysis

4.1. Experimental Parameter Setting

The DAG is randomly generated by setting parameters such as task number, communication calculation ratio (CCR) [3], parallel factor, and heterogeneous factor, which are used for experimentation. The HSP-QGA algorithm is both an improved genetic algorithm and a heuristic algorithm. Since ant colony algorithms are more often used in heuristic algorithms, we chose the genetic algorithm PA-CGA [29], the ant colony algorithm ACO [30], and the improved ant colony algorithm ACOQ [31] for comparison experiments

with the HSP-QGA. To limit the dimension of a task graph in randomly generated DAGs, set the specific parameters as follows: the parallel factor $\alpha = \{2, 4, 6, 7\}$, representing the number of tasks in each layer of DAG; the heterogeneous factor $\beta = \{0.1, 0.35, 0.65, 0.8, 1\}$, representing the difference in the ability of heterogeneous processors to handle tasks. As the various DAGs were generated randomly and independently, and the meta-heuristic algorithm is stochastic in nature, the method of averaging among 100 Monte Carlo runs is used in this paper. The primary parameter settings of each algorithm are shown in Table 5.

Table 5. Basic parameter setting table of each algorithm.

Algorithm	Parameter	Numerical Value
HSP-QGA	population size	40
	iterations	500
PA-CGA	population size	40
	iterations	500
	crossover probability	0.35
	mutation probability	0.1
ACO	population size	40
	iterations	500
	α	0.1
	β	1
	ρ	0.3
	Q	1
Q-learning	learning rate α	0.1
	discount factor γ	0.8
	exploration factor ε	0.05

4.2. Performance Evaluation Indicators

According to the actual working environment and task execution, the task scheduling needs to meet the following conditions:

1. A task is assigned to work on one processor;
2. The execution cost of all tasks on each processor cannot be greater than the maximum processing capacity of the processor itself;
3. Tasks on the processor are not allowed to terminate until execution is complete.

We use scheduling length (makespan), speedup, and efficiency as evaluation indicators to measure the quality of task scheduling in heterogeneous signal processing platforms.

The scheduling length (makespan) reflects the completion time of all tasks on the platform. The smaller the scheduling length, the shorter the scheduling time for the system to complete all tasks, as shown in Formula (11).

$$\text{makespan} = \text{EFT}(v_{exit}) \tag{11}$$

where v_{exit} represents the exit node, EFT represents the completion time of the task on the target processor, and it is the sum of the EST and execution time of task v_i, as shown in Formula (12).

$$\text{EFT}(v_i, p_k) = \text{EST}(v_i, p_k) + w(v_i)/r(p_k) \tag{12}$$

where $w(v_i)$ represents the computational cost of the task, $r(p_k)$ represents the processing power of the processor p_k, $w(v_i)/r(p_k)$ represents the execution time of task v_i on the processor p_k, and EST represents the earliest start time of the task on the target processor, as shown in Formula (13).

$$\text{EST}(v_i, p_k) = \max\{\text{Available}(v_i, p_k), \max_{v_j \in pred(v_i)}(\text{EFT}(v_j, p_f) + c_{j,i})\} \tag{13}$$

where Available(v_i, p_k) represents the earliest time block that the processor allows task v_i to execute, $pred(v_i)$ represents the set of all the predecessor tasks of v_i, and $c_{j,i}$ represents the communication time from task v_j (scheduled on the processor p_f) to task v_i (scheduled on the processor p_k).

The speedup represents the ratio of the earliest completion time and the scheduling length of the task execution on a single processor, which can well reflect the algorithm performance. When the speedup is greater than 1, heterogeneous computing is better than homogeneous computing, and the larger the speedup, the better. As shown in Formula (14), where w_i represents the computation cost of the i-th task.

$$\text{speedup} = \frac{\sum_{v_i \in v} \overline{w_i}}{\text{makespan}} \tag{14}$$

The scheduling efficiency is the ratio of speedup to the total number of processors, representing the performance and fitness of the processing platform, as shown in Formula (15), where p represents the total number of processors in the processing platform.

$$\text{efficiency} = \frac{\text{speedup}}{p} \tag{15}$$

4.3. Experimental Analysis and Summary

4.3.1. Comparison Experiment of Algorithm Convergence Speed and Scheduling Length under the Same Number of Tasks

The experiment is carried out under the same number of tasks, the number of tasks is 20, other relevant parameters are uniformly set to the CCR of 0.5, the number of processors is 4, and the number of DAGs is 1. As shown in Figure 9, by comparing the four algorithms, the HSP-QGA algorithm begins to converge at the 27th iteration, and the convergence speed is the fastest; the scheduling length is 76 when convergence and the scheduling length is the smallest. Compared to the PA-CGA algorithm, HSP-QGA encodes the initial chromosome according to the task order, which reduces the search range of initial solution space. At the same time, it uses full-interference crossover to jump out of the local optimum quickly and uses the quantum rotation gate to accelerate the overall convergence speed of the algorithm. Experimental results show that HSP-QGA has the best results in terms of both scheduling length and convergence speed for the same number of tasks.

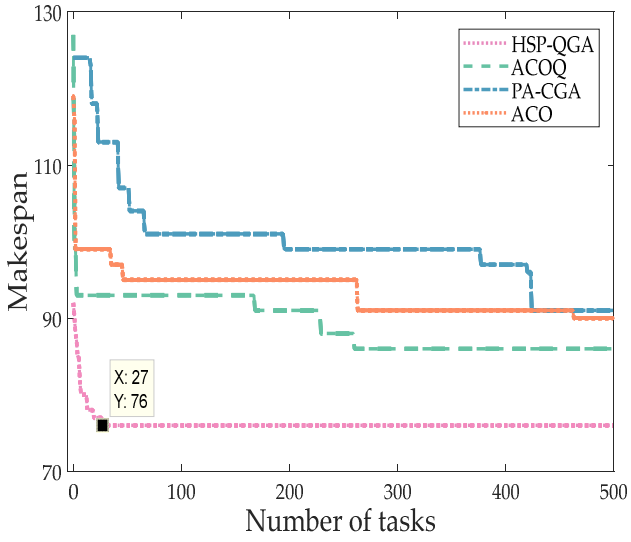

Figure 9. Comparison of convergence speed and scheduling length.

4.3.2. Comparison Experiment of Scheduling Length of Algorithms under Different Task Numbers

The experiment mainly verifies whether the scheduling result of HSP-QGA is still the optimal value under different task numbers (number of tasks: 20, 40, 60, and 80). In order to eliminate the influence of CCR values on the experimental results, smaller and larger CCR values were set up to carry out the experiments. Other parameters are uniformly set: the number of DAGs is 1, and the number of processors is 4.

Figure 10a shows the scheduling length of each algorithm when CCR = 0.1 and the number of tasks is 20, 40, 60, and 80, respectively. The results showed that the HSP-QGA reduced scheduling length by an average of 12.38% (standard deviation reduced by 10.6%) compared to the PA-CGA algorithm, 14.86% (standard deviation reduced by 18.9%) compared to the ACOQ algorithm and 20.69% (standard deviation reduced by 23.1%) compared the ACO algorithm. Figure 10b shows the scheduling length of each algorithm under different task numbers when CCR = 4. The HSP-QGA reduced scheduling length by an average of 20.47% (standard deviation reduced by 14.7%) compared to the PA-CGA algorithm, 17.61% (standard deviation reduced by 16.7%) compared to the ACOQ algorithm, and 24.80% (standard deviation reduced by 14.2%) compared to the ACO algorithm. The experimental results show that the HSP-QGA has a smaller scheduling length, whether in the case of CCR less than 1 or CCR greater than 1.

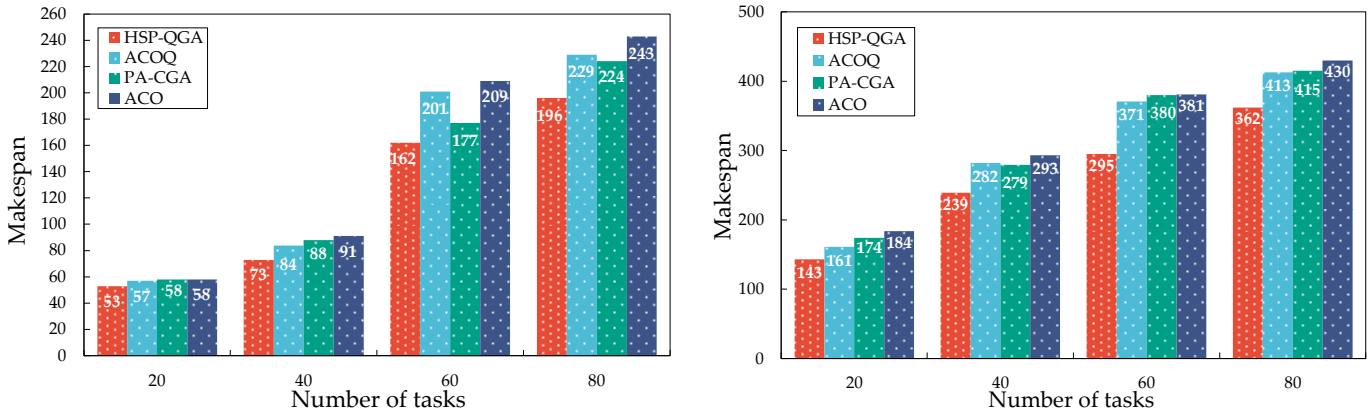

Figure 10. Comparison of scheduling lengths for different numbers of tasks: (**a**) CCR = 0.1; (**b**) CCR = 4.

4.3.3. Experiment on the Effect of Different Tasks on Speedup

The analysis of experiment (2) shows that the scheduling length increases as the number of tasks increases. It is known that this phenomenon is related to the task scheduling attribute. But it is unknown whether it is associated with the decrease in algorithm performance caused by the increasing number of tasks. In order to observe the effect of task number changes on the algorithm performance, this experiment compares and analyzes the speedup of the algorithm by setting different task numbers (number of tasks: 20, 40, 60, 80). Other parameters are uniformly set: the number of DAGs is 1, and the number of processors is 4.

The experimental results are shown in Figure 11. When CCR is small (CCR = 0.1) or large (CCR = 4), the speedup of each algorithm does not decrease with the number of tasks, indicating that the increase in the number of tasks does not lead to the decrease in the algorithm performance and that the increase in scheduling length with the number of tasks in the experiment (2) is caused by the task scheduling attribute. In addition, the speedup tends to increase as the number of tasks increases. This proves the suitability of the HSP-QGA algorithm for computation-intensive tasks and its ability to search the solution space compared to other algorithms. It is observed that the speedup fluctuation of the same algorithm is slight for different numbers of tasks, and the trend of the line graph is similar

across the algorithms. Considering that parallel factors (number of tasks per layer in the DAG) and heterogeneous factors (difference in processing ability between heterogeneous processors) are selected at random in experimental parameter settings, indicating that speedup fluctuations in the number of tasks are due to randomness generated by the DAG graphs. In addition, the speedup of HSP-QGA is more significant than other algorithms for each number of tasks, which proves that HSP-QGA has better performance.

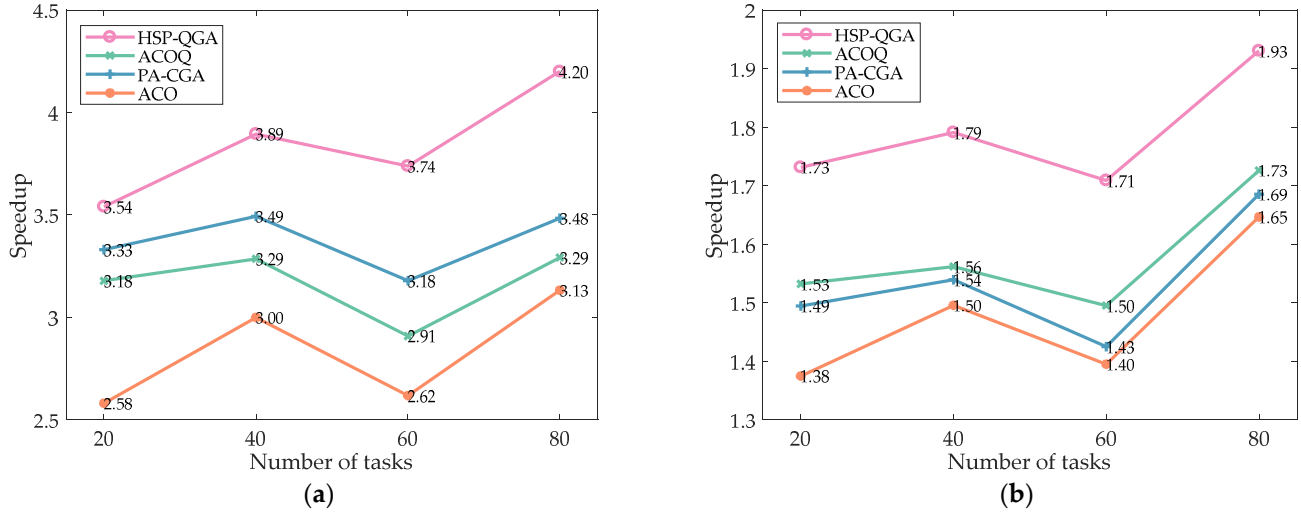

Figure 11. Effect of the number of tasks on speedup: (**a**) CCR = 0.1; (**b**) CCR = 4.

4.3.4. Experiment on the Effect of Different CCRs on Speedup

Experiments (1) to (3) show that the HSP-QGA algorithm has obvious advantages in scheduling length, convergence speed, and algorithm performance in both large and small tasks. However, in Experiment (3), speedup was significantly different when CCR = 0.1 and CCR = 4 were performed with the same number of tasks in the same algorithm.

In this experiment, we set the same number of tasks, to observe the influence of different CCR values on the speedup, and compare and analyze the scheduling performance and applicability of the algorithm. The relevant parameters are uniformly set: the number of processors is 4, the number of tasks is 20, and the number of DAGs is 2. Experimental results are shown in Figure 12. When CCR = 0.1, computing costs are much higher than communication costs, the HSP-QGA has an acceleration ratio of 3.27, 24.3% higher than the PA-CGA, 29.8% higher than the ACOQ, and 40.9% higher than the ACO. When CCR = 6, communication costs are higher than computational costs, and the HSP-QGA has an acceleration ratio of 1.09, which is 13.5%, 11.2%, and 15.9% higher than the other three algorithms, respectively. When CCR = 8, the communication cost is far higher than the computation cost, and the acceleration ratio for all algorithms is less than 1. The value of CCR is usually used to determine the type of task scheduling. Tasks with CCR > 1 are expressed as communication-intensive, and vice versa for computational-intensive. Experimental results show that the HSP-QGA speedup is larger and performs better when CCR = 0.1; when CCR = 6, the performance of the algorithm is equivalent to that of a single machine, but with a slight advantage; when CCR = 8, the algorithm performance is already inferior to that of a single machine. Although the speedup of HSP-QGA is higher than other algorithms under different CCR values, the experimental results show that the algorithm is more suitable for computation-intensive task scheduling.

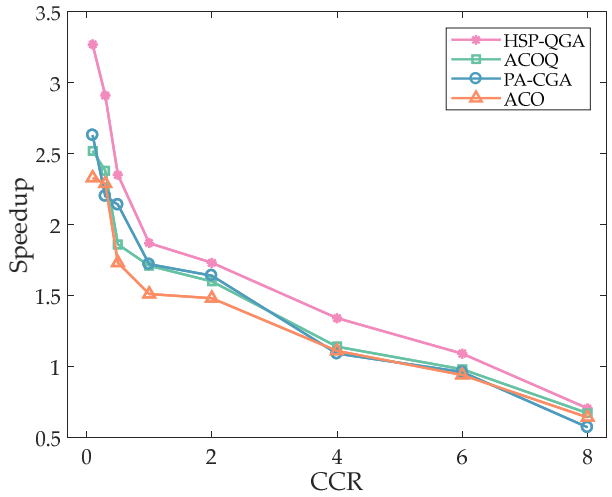

Figure 12. Effect of different CCRs on speedup.

4.3.5. Comparison Experiment of Applicability of HSP-QGA in Heterogeneous and Homogeneous Systems

The above experiments have verified the advantages of HSP-QGA in task scheduling. To verify the applicability of HSP-QGA to heterogeneous platforms, this experiment is carried out in heterogeneous and homogeneous systems. Set three data groups with 10, 20, and 30 tasks to test the HSP-QGA algorithm and compare their efficiency. The other relevant parameters except for the number of tasks in the two systems are set as follows: the number of processors is 4, CCR is 0.3, and the number of DAGs is 2.

The experimental results show scheduling efficiency in 10, 20, and 30 tasks in heterogeneous and homogeneous systems, as shown in Figure 13. When the number of tasks is 10, the scheduling efficiency of the heterogeneous system is 41.7% higher than that of the homogeneous system; when the number of tasks is 20, the scheduling efficiency of the heterogeneous system is 24.8% higher than the homogeneous system; when the number of tasks is 30, the scheduling efficiency of the heterogeneous system is 43.3% higher than that of the homogeneous system. Considering the negligible error caused by the difference in processor release performance and the randomness of generated DAG, the results show that HSP-QGA is more appropriate for heterogeneous scheduling platforms.

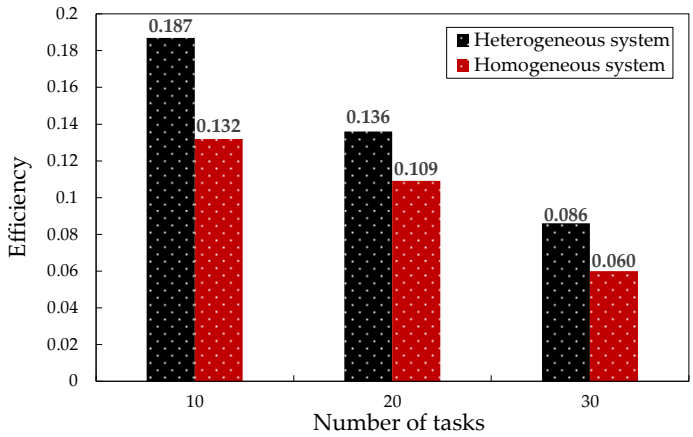

Figure 13. Scheduling efficiency of different processing systems.

5. Conclusions

This paper provided a new approach to task scheduling for heterogeneous signal processing platforms. The main technology of this paper was the adaptation of quantum genetic algorithms to task scheduling on heterogeneous processing platforms by improving traditional genetic algorithms through quantum computing. These included qubit

coding to address the lack of population diversity, full-interference crossover operations to address the problem of easily falling into local optima, and quantum rotation gates to address the problem of slow convergence. From the analysis, it could be concluded that the proposed algorithm had an average reduction in the scheduling length of 14.5%, an average increase in the convergence speed of 1319.7%, and an average increase in the speedup of 24.2% compared to other algorithms. The algorithm improved the scheduling efficiency in heterogeneous systems by an average of 36.6% over homogeneous systems. Overall, the algorithm offered higher scheduling performance and scenario adaptation for computation-intensive tasks and heterogeneous systems. However, task scheduling is a NP-hard problem, and the hardware structure and signal resources of heterogeneous signal processing platforms are complex. The algorithm in this paper still has some shortcomings and needs to be improved in practical applications:

(1) When modeling the hardware structure, this paper set the communication transmission between heterogeneous processors as an ideal state, without considering the problem of communication competition. However, in the actual platform application, the transmission bandwidth between processors or boards is different, and the communication competition during data transmission may cause delays. The next step is to improve and perfect the DAG model and hardware architecture model.
(2) The algorithm proposed in this paper used a large number of binary matrices in encoding and decoding, which increased the time complexity of the algorithm and the running time of the system. To solve this problem, we can map the processor scheduling policy to the vector set. In decoding, the quantum individual is mapped to a vector, thus a vector set is obtained, which corresponds to a processor scheduling strategy. Or we can use algorithms that specifically calculate matrix operations, such as the CORDIC (coordinated rotation digital computer) algorithm instead of quantum rotation gate operation.
(3) The algorithm proposed in this paper was mainly aimed at computation-intensive task scheduling, but there are also communication-intensive tasks in signal processing. We can reduce the communication overhead by task clustering or task duplication.

Author Contributions: Conceptualization, Y.L.; data curation, X.S.; formal analysis, Z.X. and J.M.; supervision, K.Z.; project administration, Z.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this paper can be obtained by contacting the corresponding author.

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

References

1. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 2nd ed.; W.H. Freeman & Co., Ltd.: New York, NY, USA, 1979.
2. Li, Y.; Ma, J.; Xie, Z.; Shen, X. Research on Task Scheduling of Heterogeneous Platform Signal Processing. *Electron. Sci. Technol.* **2022**, *37*, 24–34. [CrossRef]
3. Haluk, T.; Salim, H.; Min-You, W. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
4. Sih, G.C.; Lee, E.A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.* **1993**, *4*, 175–187. [CrossRef]
5. Fuxi, Z.; Yanxiang, Z. *Theory and Design of Scheduling Algorithm in Parallel Distributed Computing*, 1st ed.; Wuhan University Press: Wuhan, China, 2003.
6. Paprocka, I.; Krenczyk, D.; Burduk, A. The Method of Production Scheduling with Uncertainties Using the Ants Colony Optimisation. *Appl. Sci.* **2020**, *11*, 171. [CrossRef]

7. Sanabria, P.; Tapia, T.F.; Toro Icarte, R.; Neyem, A. Solving Task Scheduling Problems in Dew Computing via Deep Reinforcement Learning. *Appl. Sci.* **2022**, *12*, 7137. [CrossRef]
8. Fang, G. Research on Parallelization of Machine Learning Algorithms for On-Chip Heterogeneous Multi-core Systems. Ph.D. Thesis, Beijing University of Technology, Beijing, China, 2017.
9. Aziz, R.M.; Mahto, R.; Goel, K.; Das, A.; Kumar, P.; Saxena, A. Modified Genetic Algorithm with Deep Learning for Fraud Transactions of Ethereum Smart Contract. *Appl. Sci.* **2023**, *13*, 697. [CrossRef]
10. Raji, M.; Nikseresht, M. UMOTS: An uncertainty-aware multi-objective genetic algorithm-based static task scheduling for heterogeneous embedded systems. *J. Supercomput.* **2021**, *78*, 279–314. [CrossRef]
11. Fekih, A.; Hadda, H.; Kacem, I.; Hadj-Alouane, A.B. A Random-key based genetic algorithm for the flexible job-shop scheduling minimizing total completion time. In Proceedings of the 2021 14th International Conference on Developments in eSystems Engineering (DeSE), Sharjah, United Arab Emirates, 7–10 December 2021; pp. 443–447.
12. Basahel, S.B.; Yamin, M. A Novel Genetic Algorithm for Efficient Task Scheduling in Cloud Environment. In Proceedings of the 2022 9th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 23–25 March 2022; pp. 30–34.
13. Hui, C.; Zhang, J.; Chao, Z. Chaos updating rotated gates quantum-inspired genetic algorithm. In Proceedings of the 2004 International Conference on Communications, Circuits and Systems, Chengdu, China, 27–29 June 2004; pp. 1108–1112.
14. Gandhi, T.; Nitin; Alam, T. Quantum genetic algorithm with rotation angle refinement for dependent task scheduling on distributed systems. In Proceedings of the 2017 Tenth International Conference on Contemporary Computing (IC3), Noida, India, 10–12 August 2017; pp. 1–5.
15. Alam, T.; Raza, Z. Quantum genetic algorithm based scheduler for batch of precedence constrained jobs on heterogeneous computing systems. *J. Syst. Softw.* **2018**, *135*, 126–142. [CrossRef]
16. Konar, D.; Bhattacharyya, S.; Sharma, K.; Sharma, S.; Pradhan, S.R. An improved Hybrid Quantum-Inspired Genetic Algorithm (HQIGA) for scheduling of real-time task in multiprocessor system. *Appl. Soft Comput.* **2017**, *53*, 296–307. [CrossRef]
17. Guo, J.; Sun, L.-J.; Wang, R.-C.; Yu, Z.-G. An Improved Quantum Genetic Algorithm. In Proceedings of the 2009 Third International Conference on Genetic and Evolutionary Computing, Guilin, China, 14–17 October 2009; pp. 14–18.
18. Teng, H.; Zhao, B.; Yang, B. An Improved Mutative Scale Chaos Optimization Quantum Genetic Algorithm. In Proceedings of the 2008 Fourth International Conference on Natural Computation, Jinan, China, 18–20 October 2008; pp. 301–305.
19. Chang, Y.-F.; Xie, H.; Huang, W.-C.; Zeng, L. An Improved Multi-objective Quantum Genetic Algorithm Based on Cellular Algorithm. In Proceedings of the 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 23–25 November 2018; pp. 342–345.
20. Zhu, X.-Q.; Gui, Y.; Gao, X.-H. A novel multi-subpopulation quantum genetic algorithm. In Proceedings of the 2008 International Conference on Machine Learning and Cybernetics, Kunming, China, 12–15 July 2008; pp. 3530–3534.
21. Ni, H.-M.; Wang, W.-G. A Niche Quantum Genetic Algorithm Used In Multi-peak Funktion Optimization. In Proceedings of the 2010 Sixth International Conference on Natural Computation, Yantai, China, 10–12 August 2010; pp. 2239–2242.
22. Zhao, Z.; Zhen, S.; Shang, J.; Kong, X. Research on Cognitive Radio Decision Engine Based On Quantum Genetic Algorithm. *Acta Phys. Sin.* **2007**, *56*, 6760–6766. [CrossRef]
23. Pitchai, A.; Reddy, A.V.; Savarimuthu, N. Quantum walk based Genetic Algorithm for 0-1 Quadratic Knapsack Problem. In Proceedings of the 2015 International Conference on Computing and Network Communications (CoCoNet), Trivandrum, India, 16–19 December 2015; pp. 283–287.
24. Chatterjee, N.; Paul, S.; Mukherjee, P.; Chattopadhyay, S. Deadline and energy aware dynamic task mapping and scheduling for Network-on-Chip based multi-core platform. *J. Syst. Archit.* **2017**, *74*, 61–77. [CrossRef]
25. Saad, H.M.H.; Chakrabortty, R.K.; Elsayed, S.; Ryan, M.J. Quantum-Inspired Genetic Algorithm for Resource-Constrained Project-Scheduling. *IEEE Access* **2021**, *9*, 38488–38502. [CrossRef]
26. Wang, Y.; Li, Y. A Novel Quantum Genetic Algorithm for TSP. *Chin. J. Comput.* **2007**, *30*, 748–755.
27. Yang, S.; Jiao, L.; Liu, F. Quantum Evolutionary Algorithm. *Chin. J. Eng. Math.* **2006**, *23*, 235–246.
28. Li, L.; Cui, G.; Lv, X.; Sun, X.; Wang, H. An Improved Quantum Rotation Gate in Genetic Algorithm for Job Shop Scheduling Problem. In Proceedings of the 2018 International Conference on Information Systems and Computer Aided Education (ICISCAE), Changchun, China, 6–8 July 2018; pp. 322–325.
29. Dorronsoro, B.; Pinel, F. Combining Machine Learning and Genetic Algorithms to Solve the Independent Tasks Scheduling Problem. In Proceedings of the 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), Exeter, UK, 21–23 June 2017; pp. 1–8.
30. Xiang, B.; Zhang, B.; Zhang, L. Greedy-Ant: Ant Colony System-Inspired Workflow Scheduling for Heterogeneous Computing. *IEEE Access* **2017**, *5*, 11404–11412. [CrossRef]
31. Li, N.; Gao, B.; Xie, Z.; Zhang, F.; Wan, J. Q-learning Based Intelligent Ant Colony Scheduling Algorithm in Heterogeneous System. In Proceedings of the 2021 IEEE 4th International Conference on Electronics Technology (ICET), Chengdu, China, 7–10 May 2021; pp. 1020–1025.