The International Journal of Advanced Manufacturing Technology (2018) 94:3399–3412
https://doi.org/10.1007/s00170-017-1447-y

ORIGINAL ARTICLE

Parallel design of intelligent optimization algorithm based on FPGA

Xiaofu Zou[1] · Lina Wang[1] · Yue Tang[1] · Yilong Liu[2] · Shicheng Zhan[3] · Fei Tao[1]

Received: 11 April 2017 / Accepted: 27 November 2017 / Published online: 10 January 2018

Abstract
Intelligent optimization algorithm (IOA) has been widely studied and applied to solve various optimization problems. When scholars improve IOA with mathematical methods, they also want to seek an effective method to implement algorithms with higher real time, especially for a complex problem. Parallel design is an effective method to improve the real time of IOA. Currently, the parallel programming based on open multi-processing (OpenMP) and compute unified device architecture (CUDA) are two popular methods. To find and develop a new IOA parallel method, in this paper, a parallel design and implementation method based on field programmable gate array (FPGA) is explored. In order to validate the proposed method, parallel genetic algorithm (GA) and parallel particle swarm optimization (PSO) algorithm are realized by the proposed method. Furthermore, the performance and advantage of the proposed FPGA-based parallel IOA method are tested by comparing with OpenMP-based parallel programming and CUDA-based parallel programming, the final results show that the proposed method with highest real-time performance in IOA parallel implementation. A case study by using FPGA-based parallel simulate annealing (SA) to address job shop scheduling problem (JSSP) to illustrate the proposed method has high potential in industrial applications.

Keywords Intelligent optimization algorithm (IOA) · Open multi-processing (OpenMP) · Compute unified device architecture (CUDA) · Field programmable gate array (FPGA) · Real-time

1 Introduction

Intelligent optimization algorithm is a kind of intelligent computational method based on biological and natural phenomena, which has been widely applied to engineering optimization, intelligent control, system identification, weather forecast, and scheduling. The study of IOA can be primarily divided into four areas: (1) algorithm improvement, (2) hybrid algorithm design, (3) new algorithm design, and (4) algorithm application. Nowadays, algorithm application has been paid more and more attention by scholars and practitioners [1]. Different from the above studies, this paper focuses on how to improve the real-time ability of the IOA execution from the perspective of parallel.

Currently, studies about parallel implementing for IOA can be divided into the following two classes in mostly: (1) OpenMP parallel programming based on multi-core processors and (2) CUDA parallel programming based on graphics processing unit (GPU) [2–4]. Another parallel method is message passing interface (MPI) based on computer cluster. General computer is served as the hardware carrier for the above methods. In general computer's parallel design, the algorithm is divided into some parts which can be parallel executed in different threads in a single core. The different threads sharing a main thread, it is serial execution in essence. Although a computer can have multi-core processors, the number of cores is limited. Communication between different cores will increase time consumption. Meanwhile, general computer needs translation instructions to form hardware circuit, this increased the time consumption too.

Is there a kind of hardware which can support an algorithm parallel execution in bottom hardware circuit? FPGA is an answer. A blank FPGA resembles "a white paper," which is composed of thousands of logic gates. The user's program

✉ Fei Tao
ftao@buaa.edu.cn

1 School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China

2 The 54th Research Institute of China Electronics Technology Group Corporation, Shijiazhuang 050081, China

3 Beijing Shenzhou Feihang Technology Co., Ltd., Beijing 100089, China

forms the hardware circuit by configuring these logic gates. Different from general processors' serial execution way, once the hardware circuit is formed, the output is obtained after some clock delay. Different user's programs can form different independent hardware circuits, which makes it possible to achieve parallel execution. So FPGA's parallel is born.

In this paper, an FPGA-based IOA parallel design and implementation method is proposed firstly. In order to validate the proposed method, parallel GA and PSO algorithm are realized by the proposed method. Furthermore, the performance and advantage of the proposed FPGA-based parallel method are tested by comparing with OpenMP-based parallel programming and CUDA-based parallel programming; the final results show the proposed method with highest real-time performance in IOA parallel implementation. A case study by using FPGA-based parallel SA to address JSSP illustrates the proposed method has high potential for industry application.

The contributions of this research are listed as follows:

1) Propose a new method which parallel design IOAs based on FPGA.
2) Summarize a general design flow of parallel design IOA based on FPGA.
3) Parallel design GA and PSO by FPGA, OpenMP, and CUDA, the results are compared and analyzed.
4) A case study by using FPGA-based parallel SA to address JSSP to illustrate FPGA-based parallel IOAs has high potential in industrial applications.

The rest of this paper is organized as follows. Section 2 reviews the related work in IOAs and its typical parallel design methods. Section 3 introduces the proposed method and its implementation. Comparisons and analysis are carried out in Section 4. A case study is carried out in Section 5. Section 6 concludes this paper.

2 Literature review

In recent years, IOAs have been extensively studied by scholars. And some IOAs have been successfully applied to practical engineering. GA and PSO is a typical study object of IOAs.

2.1 Related research and application of GA

Siano et al. [5] researched the optimization of smart grid management systems based on GA. Li [6] proposed to use GA to quantitatively analyze wavelength selection of a terahertz spectroscopy. Datta et al. [7] proposed use of multi-objective GA to analyze and design optimization of a robotic gripper is studied. In order to effectively recognize shape contour, a GA-based explicit description is studied in literature [8]. Ye et al. [9] researched the minimum exposure path problem of wireless sensor networks based on hybrid genetic algorithm. GA also can be used to solve carpool services problem in literature [10, 11].

2.2 Related research and application of PSO

Chan et al. [12] studied the use of PSO to enhance speech recognition. Using PSO to enhance the control of complex distributed networks and optimize the management of wind power has been researched in literature [13–15]. As radio frequency identification (RFID) technology is widely used in industrial networks, the optimal planning of RFID network become more and more important. Gong et al. [16] researched the application of PSO in RFID network planning. Yao et al. [17] researched the use of quantum-inspired PSO to decrease the instability of wind power system. Tao et al. [18, 32] researched the resource service composition and parallel method in manufacturing grid system based on PSO. Not only does utilizing PSO to solve various practical optimization problems draw scholars' attention but it is also necessary to research and analyze PSO algorithm itself. Luis et al. [19] researched the stochastic stability in continuous and discrete PSO model.

2.3 Typical IOA parallel design methods

Though some IOAs have been successfully applied to solve practical problems, the efficient execution of IOAs fails in getting enough attention in the past. Only recently, the efficiency execution of IOAs went into researchers' vision, and algorithm parallel design becomes one of research hotspots. General computers were often used as the hardware platform of IOAs previously. However, general computers usually have little CPU cores. Though it can simulate multiple threads by hyper-threading technology to parallel design, this parallel method by time slice circular is not parallel in true. The efficiency of algorithm parallel execution is largely limited. Nowadays, two software-based parallel programming modes combining specific hardware become popular.

1. *OpenMP-based parallel programming mode*. With rapid development of chip technology, it is easy to integrate multiple CPU cores in a single chip. As a result, the real multi-core processors can be achieved, which provide real hardware support to parallel design of IOA. At present, OpenMP programming technique based on shared memory is used in algorithm parallel programming in single-chip multi-core processors in most conditions. Liu et al. [20] researched the time consumption of decoupled software pipelining

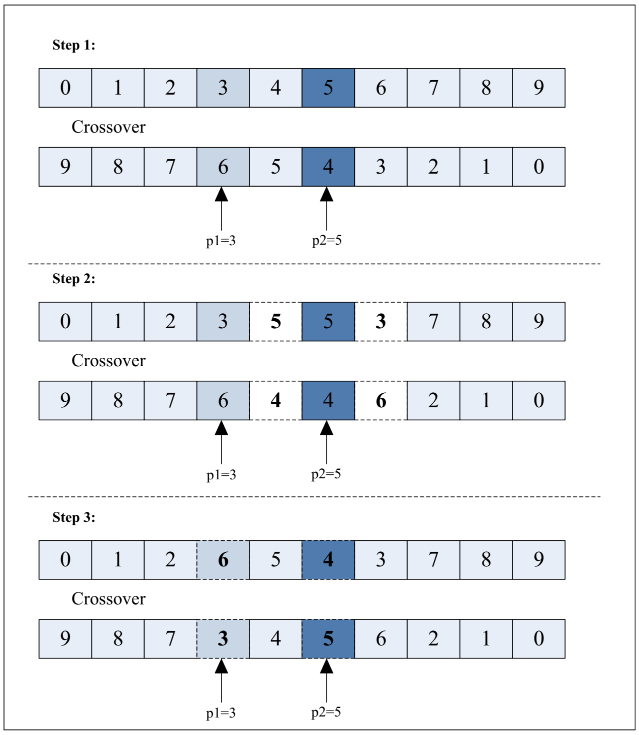


Fig. 1 Crossover operator execution step

(DSWP) based on OpenMP during compiling. Batch queue algorithm is used to improve the parallel execution efficiency of OpenMP pipeline in literature [21]. OpenMP parallelized loops and stream programming problems are studied in literature [22, 23].

2. *CUDA-based parallel programming mode*. GPU is another hardware which can support algorithm parallel design. GPU usually adopts a cooperative work model of CPU-GPU. CPU acts as the master controller which is responsible for algorithm scheduling, and GPU acts as the co-processor which is responsible for algorithm parallel execution. Currently, CUDA parallel programming technique is widely used in GPU. A high-performance acceleration of algorithm parallel design based on CUDA is researched in literature [24]. According to literature [25, 26], CUDA technology in graphics processing hardware can be used to improve the performance of image processing. Zhang et al. [27] researched the neural-machine interface for artificial legs based on GPU.

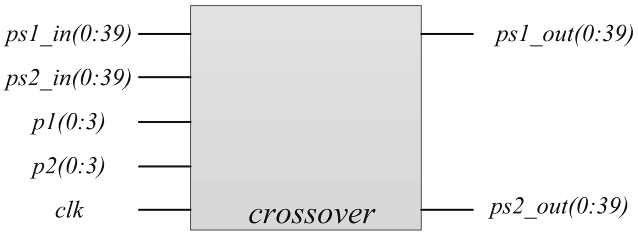


Fig. 2 Structure of crossover operator in FPGA

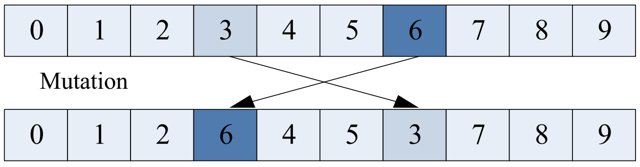


Fig. 3 Mutation operator execution step

2.4 Related research and application of FPGA

The above-mentioned two methods utilize software parallel method to design algorithm parallel, and they all face some restrictions. For example, OpenMP fails to get rid of general computers' serial operation mode. CUDA is complex. In this situation, FPGA is a potential power to change current status. It can support algorithm parallel design from bottom hardware circuit. Not only that, according to other algorithms which is used in smart manufacturing field, such as industrial Internet of things algorithm [28], virtual and entity fusion algorithm [29, 30] and product lifecycle management algorithm with big data [31], can be accelerated by FPGA. Tao et al. [32] proposed a new simulation model under cloud environment. Inspired by above thinking, distributed simulation acceleration method with FPGA has great potential. FPGA-based general purpose neural networks for online applications, PWM generation and current control unit in power conversion system are studied in literature [33–35]. Zou et al. [36] proposed a new method to accelerate collect bottom data in supply chain network with FPGA. However, many scholars only use FPGA to design logic control program, ignoring its potential advantage in IOA parallel design.

3 Parallel design of IOA based on FPGA

Time consumptions of algorithm's execution failed to become the focal points in some practical applications. This section proposes an FPGA-based general method to parallel design IOA. Then considering that GA and PSO are two of the widely used typical IOA, they are used as examples to validate FPGA-based parallel design method in this paper.

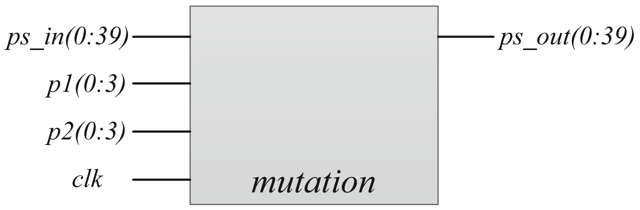


Fig. 4 Structure of mutation operator in FPGA

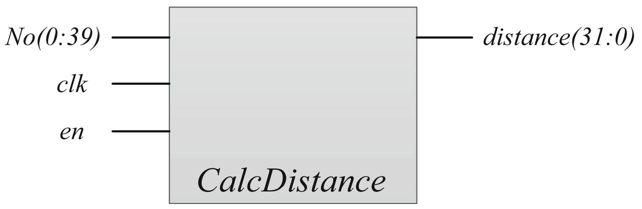


Fig. 5 External interface of FPGA-based TSP

3.1 General method of parallel design IOA based on FPGA

A new FPGA is a blank processor. Except logic resources and other auxiliary computing resources, when users' codes are downloaded into an FPGA chip, hardware circuits are automatically generated. Different algorithm codes generate different hardware circuits automatically. This characteristic supports algorithm parallel design in bottom hardware circuit. The FPGA-based general parallel design method of IOA is as follows:

1) *IOA parallelism analyze*

By analyze principle and mathematical formula, IOA can be divided into several parallel modules, these modules can independent operation. Then it utilizes Matlab to verify the function of each module, and to verify the communication among each module by C++ in multi-threads.

2) *Arithmetic logical unit design in FPGA*

Different from general computer, there is no fixed computing architecture, arithmetic logical unit in FPGA. According to IOA mathematical formula, it needs to design an arithmetic logical unit (ALU) to realize add, subtract, multiply, and divide. In consideration if there is no decimal point computing in FPGA, the ALU needs to have floating-point arithmetic which accord with IEEE754 international standard. If there are many logic resources in FPGA, each parallel module can have one type of ALU; in this case, one module can use the ALU at any moment without waiting for another module. If there are not many logic resources in FPGA, each parallel module can share one type of ALU; this will reduce the parallelism of each module.

3) *Control unit design in FPGA*

According to FPGA characteristic, there is also no controller, it needs to design control unit (CU) to control ALU. The CU is responsible for operation state switching of IOA, such as timing control, communication and interlock of operations, and data read from memory and write to memory, etc. Each parallel module can have its own CU to improve parallel ability of IOA.

4) *Debug and iterative optimization*

Debug is a must for programming, before downloading the codes into FPGA, a simulation can help find problems.

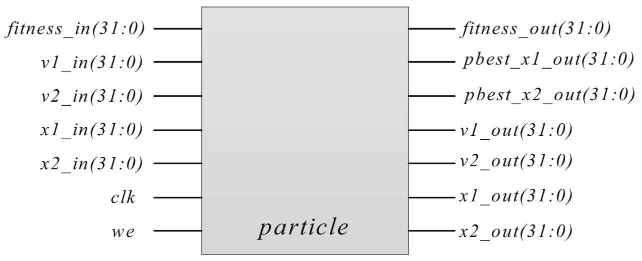


Fig. 6 Single-particle external interfaces in FPGA

3.2 Case of parallel design of IOA based on FPGA

3.2.1 Brief introduction of two typical IOAs

① GA

GA derives from Darwin's biological evolution theory and Mendel's genetic variation theory. Its original model was proposed by Professor John Holland in 1975. Its main idea is to simulate creatures' survival in a natural environment that is the fittest, and the superior survive while the inferior is wiped out. When GA is used, the usual procedure is as follows [37]:

(1) Encode variables to form "chromosome," and then initialize groups randomly.
(2) Evaluate individual fitness among groups.
(3) Perform cross-selection, crossover, and mutation in order to produce next generation.
(4) Judge whether the termination condition is met. If so, then output the results, otherwise, jump to (3).

In GA, cross-selection, crossover, and mutation are independent, so it is feasible to design them concurrently.

② PSO algorithm

PSO is an intelligent algorithm based on foraging behavior of birds, and algorithm's convergence process is like birds looking for food. Each particle is like a bird. Algorithm getting its optimal or ideal solution is like birds finding their food, which means a bird (a particle) is located in food place

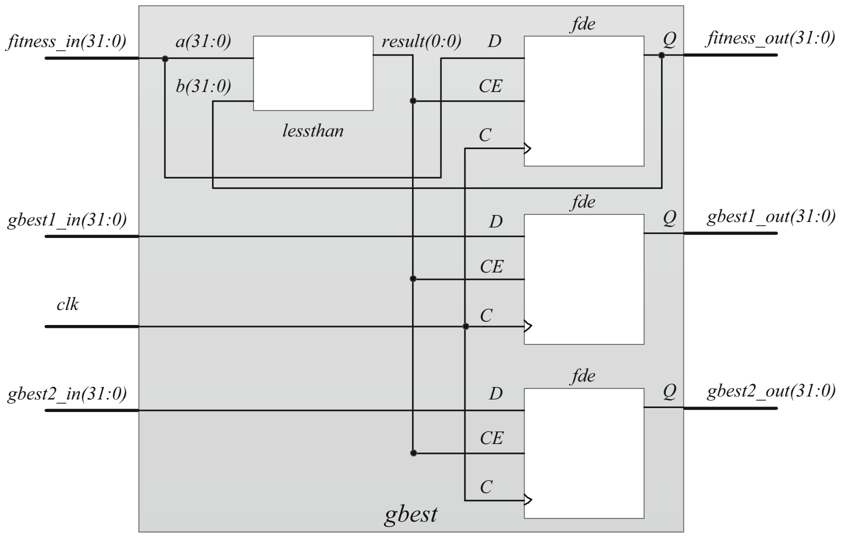


Fig. 7 Global optimal solution module external interfaces in FPGA

(optimal solution). PSO algorithm is described as the following mathematical formula:

$$\mathrm{v}_{ij}(t+1) = \mathrm{v}_{ij}(t) + c_1 r_1 \left[p_{ij} - \mathrm{x}_{ij}(t)\right] + c_2 r_2 \left[p_{gj} - \mathrm{x}_{ij}(t)\right]$$

$$\mathrm{x}_{ij}(t+1) = \mathrm{x}_{ij}(t) + \mathrm{v}_{ij}(t+1), j = 1, \ldots, d$$

where d is the dimension of an algorithm search space, $x_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$ is the current time position of particle i, and $v_i = [v_{i1}, v_{i2}, \ldots, v_{id}]$ is the current time velocity of particle i. $p_{\mathrm{i}} = [p_{\mathrm{i1}}, p_{\mathrm{i2}}, \ldots, p_{\mathrm{id}}]$ is the best location of each particle at current time, while p_g is the best position of swarm. c_1 and c_2 are accelerated constants. $r_1$and r_2 are uniformly distributed random numbers within interval (0, 1).

When basic PSO algorithm is used, the usual procedure is as follows [38]:

(1) Initialize related swarm parameters: swarm size, accelerated constant, randomly generated position, and velocity of each particle.
(2) Evaluate each particle, put the best trajectory of the particle into *pbest*, and then put the best of *pbest* into *gbest*.
(3) Update each particle's parameters according to the formula of speed and position.
(4) Re-evaluate and modify each particle and then update *pbest* and *gbest*.
(5) Judge whether the termination condition is met. If so, then output the results, otherwise, jump to (3).

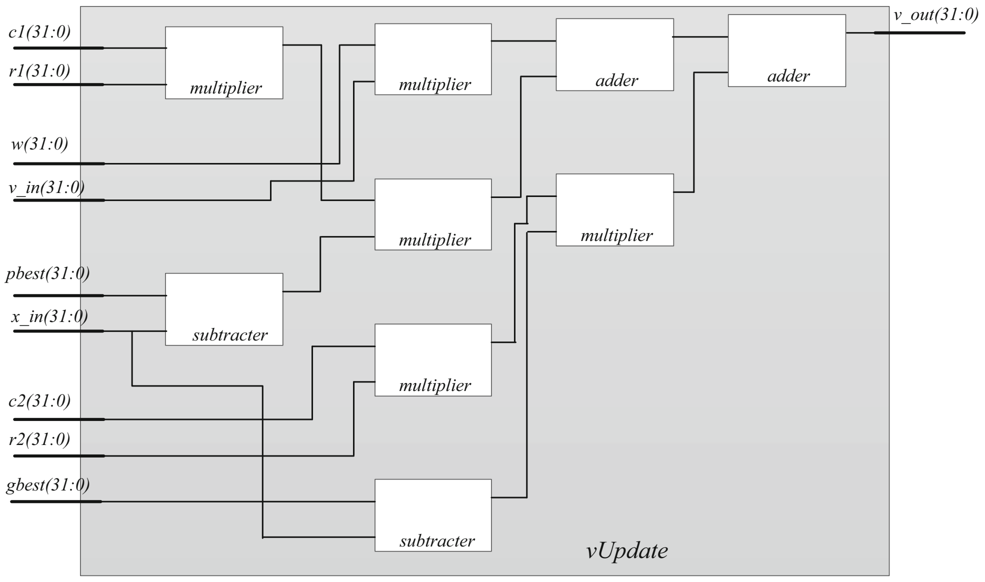


Fig. 8 Velocity updates module external interfaces in FPGA

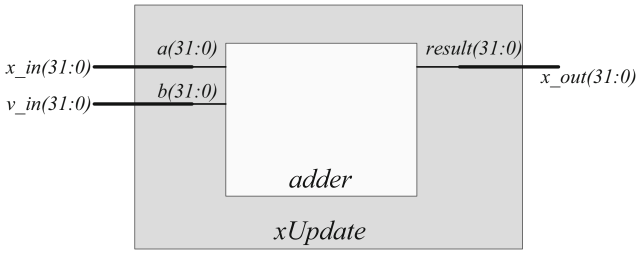


Fig. 9 Position updates module external interfaces in FPGA

3.2.2 Two study cases of GA and PSO

① The traveling salesman problem (often called TSP) TSP is a classic algorithmic problem, which can be described as that the distances between *n* cities are given, and it is required to determine the shortest route between these cities. It is focused on optimization. In this context, a better solution often means a solution that is cheaper.

In this paper, TSP is taken as a study case for GA to solve the optimization problem to keep the travel cost as well as travel distance as low as possible.

② Brief introduction of Sphere function

In this paper, parallel design for PSO algorithm is used to execute Sphere function. The mathematical formula of Sphere is

$$f(X) = \sum_{i=1}^{n} x_i^2, |x_i| \le 100$$

The optimal status of the function is $min[f(X^*)] = 0$, and the optimal value is $X^* = [0, \ldots, 0]$.

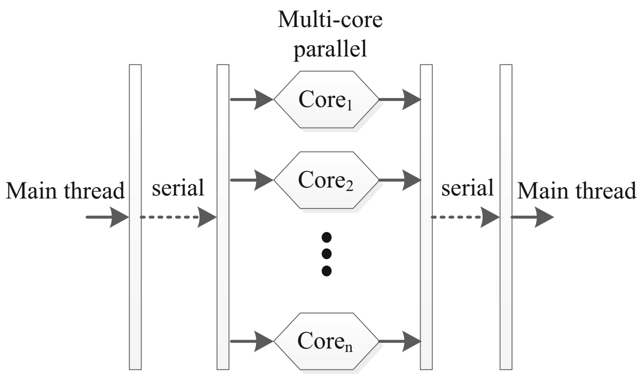


Fig. 10 Parallel execution theory of OpenMP

3.2.3 GA parallel design based on FPGA

In this paper, using GA for TSP is taken as an example of FPGA-based GA parallel design. The design process can be divided into five steps: ① put TSP benchmark parameters into FPGA, ② design a scheme to calculate path length and its hardware circuit, ③ design GA encoding scheme based on the scheme in ②, ④ design the mutation and crossover operator of GA, and ⑤ debug the program and verify result.

Only high and low voltage levels are used in FPGA digital circuit, and binary coding is used to design GA. According to the process of solving TSP, each candidate solution represents a traversal sequence, and the traversal sequence is an integer data, which can be described by unsigned binary.

In the study case, ten cities are assumed, and the city number can be denoted by *0,1,…,9*. These numbers in binary are in the range of $2^4 = 16$, so each traversal sequence can be represent by 4-bit binary code. The total code length is $4 \times 10 = 40$ bits.

As shown in Fig. 1, assign *0123456789* (hexadecimal) and *9876543210* to two sequences, and set *p_1* = *3* and *p_2* = *5* in step 1. In step 2, after pre-modification of the data, the two sequences are changed to *0123553789* and *9876446210*. In step 3, after exchanging the corresponding position data, the two sequences are changed to *0126543789* and *9873456210*. FPGA-based crossover operator implementation structure is shown in Fig. 2. There are three input signals and one output signal—sequence value input signal *ps_in(ps1_in,ps2_in)*, parameter *p1* and *p2*, time signal *clk*, and sequence value output signal *ps_out(ps1_out,ps2_out)*.

Two locations of the two sequences are randomly chosen by mutation operator, and then exchange corresponding point. The schematic and FPGA implementation structure are as shown in Figs. 3 and 4.

External interface of FPGA-based TSP is shown in Fig. 5. Its input signals are time signal clk, enable signal *en* and cities' traversal sequence signal, while the output signal is path length of traversal sequence.

3.2.4 PSO parallel design based on FPGA

PSO algorithm is mainly composed of five modules which are particle swarm module, global optimal solution module, velocity and position updating module, random data generation module, and objective function module. Random data generation module is mainly used to generate random numbers *r1* and *r2* in velocity updated formula. There are two ROMs which are 128 in depth and 32 bits in width. Therefore, there are 128 single-precision random data in each ROM.

Particle swarm module is constituted by many single particles, and each single particle's structure design and external interface is shown in Fig. 6. It is the result of FPGA synthesis. As can be seen in Fig. 6, besides time signal clk and write

Fig. 11 OpenMP parallel design codes of GA and PSO in C++. **a** C++ codes for GA. **b** C++ codes for PSO

```
          //OpenMP key source code of GA
          //
#pragma omp parallel for//for loop parallel scheduling, population parallel
evolution
          for (int index=0;index<popsize;index+=2)               {
               sel(seln,index,p);//Choose two individuals
               cross(cityNum,seln,s,scross,index,pc);//Crossover operation
               mutation(cityNum,smutation,scross,index,pm);//Variation operation
               mutation(cityNum,smutation,scross,index+1,pm);
          }
```

(a) C++ codes for GA

```
          //OpenMP key source code of PSO
          //gbest update
#pragma omp parallel for //for loop parallel scheduling
          for(int i=0;i<N;i++)
                if (sphere(gbest,d) > sphere(&pbest[i*d],d))
#pragma omp critical (gbest){
                      for (int j=0;j<d;j++)
                            gbest[j]=pbest[i*d+j];
                }
```

(b) C++ codes for PSO

enable signal *we*, there are another three input signals and four output signals in each single particle. They are speed input signal *v_in(v1_in,v2_in)*, position input signal *x_in(x1_in,x2_in)*, fitness value input signal *fitness_in*, speed output signal *v_out(v1_out,v2_out)*, position output signal *x_out(x1_out,x2_out)*, particle previous optimal solution output signal *pbest_out(pbest_x1_out, pbest_x2_out)*, and current fitness output signal *fitness_out*.

Global optimal solution module contains global optimal value and fitness value of PSO, Its structure design and external interface is shown in Fig. 7 which is the result of FPGA synthesis. From Fig. 7, we can see that besides time signal clk, there are another two input signals and two output signals. They are fitness value input signal fitness_in, current global optimal solution input signal *gbest_in(gbest1_in,gbest2_in)*, fitness value output signal fitness_out, and current global optimal solution output signal *gbest_out(gbest1_out,gbest2_out)*.

Velocity update module mainly updates the velocity of PSO, and it is in the form of combinational circuit to achieve parallel execution velocity updating formula. Its structure design and external interface is shown in Fig. 8 which is the result of FPGA synthesis.

As shown in Fig. 8, velocity update module has nine input ports and one output port. The nine input ports are acceleration constant *c1* and *c2*, inertia weight *w*, particle previous optimal solution *pbest*, current global optimal solution *gbest*, random

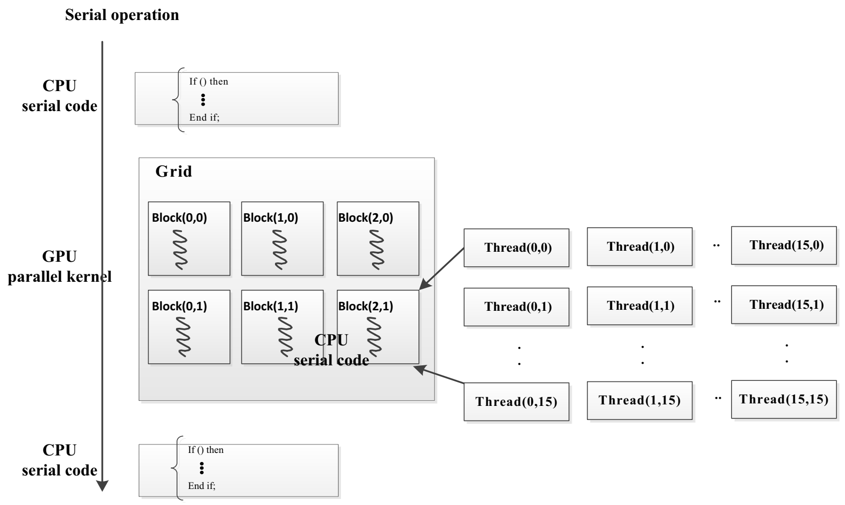
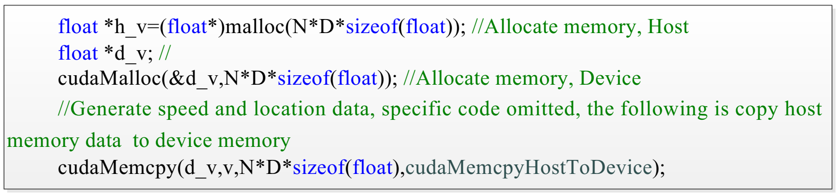


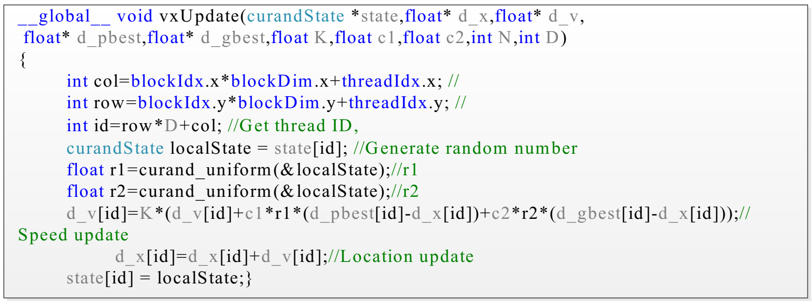
Fig. 12 Schematic diagram of CUDA parallel execution

Fig. 13 CUDA parallel programming code of PSO



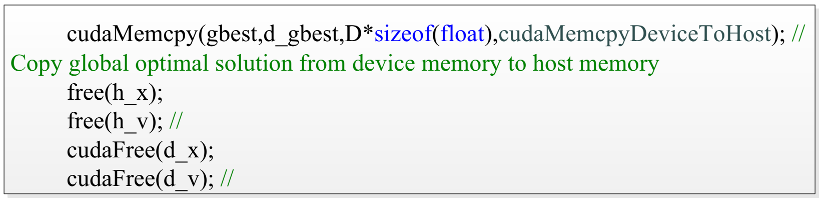
```
     float *h_v=(float*)malloc(N*D*sizeof(float)); //Allocate memory, Host
     float *d_v; //
     cudaMalloc(&d_v,N*D*sizeof(float)); //Allocate memory, Device
     //Generate speed and location data, specific code omitted, the following is copy host
memory data  to device memory
     cudaMemcpy(d_v,v,N*D*sizeof(float),cudaMemcpyHostToDevice);
```

(a)



```
__global__ void vxUpdate(curandState *state,float* d_x,float* d_v,
 float* d_pbest,float* d_gbest,float K,float c1,float c2,int N,int D)
{
     int col=blockIdx.x*blockDim.x+threadIdx.x; //
     int row=blockIdx.y*blockDim.y+threadIdx.y; //
     int id=row*D+col; //Get thread ID,
     curandState localState = state[id]; //Generate random number
     float r1=curand_uniform(&localState);//r1
     float r2=curand_uniform(&localState);//r2
     d_v[id]=K*(d_v[id]+c1*r1*(d_pbest[id]-d_x[id])+c2*r2*(d_gbest[id]-d_x[id]));//
Speed update
          d_x[id]=d_x[id]+d_v[id];//Location update
     state[id] = localState;}
```

(b)



```
     cudaMemcpy(gbest,d_gbest,D*sizeof(float),cudaMemcpyDeviceToHost); //
Copy global optimal solution from device memory to host memory
     free(h_x);
     free(h_v); //
     cudaFree(d_x);
     cudaFree(d_v); //
```

(c)

numbers r1 and r2, current velocity input v_in, and current position input x_in, while the output port is updated velocity output v_out.

Position update module mainly updates the position of PSO, using combinational circuit to achieve parallel execution position updating formula. Its structure design and external interface is shown in Fig. 9 which is the result of FPGA synthesis. There are two input signals and one output signal, which are current position input signal x_in, current velocity input signal v_in, and updated position output signal x_out.

In this paper, PSO parallel design is verified by Sphere function, which contains ten multipliers and adders.

4 Comparisons and analysis

In order to verify the advantages of FPGA-based parallel design of GA and PSO, OpenMP-based and CUDA-based parallel design of GA and PSO are implemented. Then analysis of the result of above methods is compared.

4.1 Parallel design of IOA based on OpenMP

At present, parallel design in multi-core processors is primarily based on the shared-memory OpenMP technology. OpenMP mainly contains three types of programming

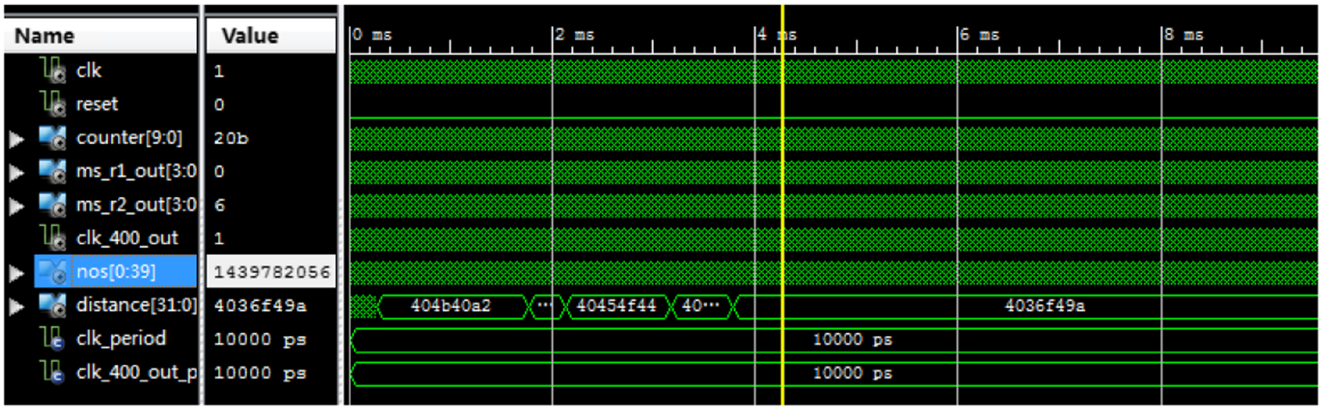


Fig. 14 GA-based TSP simulation results in FPGA. FPGA uses VHDL language, and its operating frequency is 100 MHz. FPGA belongs to Xilinx Co. Ltd., and its model is XC5VLX50T. At the same time, the integrated development software is ISE, version is 14.1

Table 1 Time-consuming comparison of PSO and GA (unit: ms)

Items	PSO	GA
OpenMP	62.3	88.7
CUDA	315.8	365.6
FPGA	20	24.5

OpenMP runs on an Intel i3 dual-core hyper-threading processor. *CUDA* runs in NVIDIA's GeForce 310M GPU. *FPGA* uses VHDL language, and its operating frequency is 100 MHz. *FPGA* belongs to Xilinx Co. Ltd., and its model is XC5VLX50T. At the same time, the integrated development software is ISE, version is 14.1

Table 2 Accuracy comparison of PSO and GA

Items	PSO	GA
Ideal convergence value	0	2.6907
FPGA convergence value	0.0012	2.8587

elements: compiled guidance, API function set, and environment variables. By calling API functions, the thread task can be assigned in multiple cores to improve efficiency of the algorithm.

The mechanism of OpenMP parallel execution is shown in Fig. 10.

In the process of programming, the commands of OpenMP compiler guidance begin with *#pragma omp*, followed by specific instructions. In the case of GA and PSO, parts that can be parallel designed are updating processes. These processes are often implemented by *for cycle*. Adding *#pragma omp parallel for* instruction before for cycle, the parallel design can be achieved.

OpenMP parallel design codes for GA and PSO in C++ is as Fig. 11 shows.

4.2 Parallel design of IOA based on CUDA

In a CUDA platform, CPU acts as a "host" instructing the process to GPU. And processing data is done in serial in CPU. GPU, functioning as a co-processor, mainly executes parallel computing in each core. Schematic diagram of CUDA parallel execution based on CPU+GPU architecture is shown in Fig. 12.

Kernel function is defined by *_global_*, and called by CPU. Each thread grid contains some "blocks," and each block contains some threads. In the hardware architecture, the number of "blocks" contained in GPU is the number of "blocks" to execute kernel function in parallel.

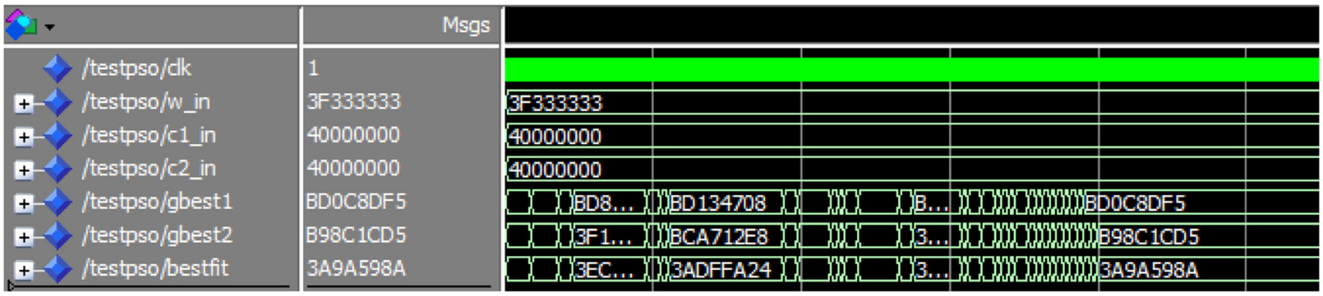
Before CUDA for an IOA is designed, the parallelism of the IOA should be analyzed at first. When adopting GA to solve TSP, each individual's selection, crossover, and mutation is independent of each other, and no data exchange exits. That is very suitable for GPU parallel implementation. Each individual's process can be assigned to a distinct core. When adopting PSO to solve the Sphere function optimization issue, each particle's velocity, position, and optimal solution updating is independent of each other, and also no data exchange exists. That is also very suitable for GPU parallel implementation. Each particle's updating process can be assigned to a distinct core. However, the global optimal solution is achieved by sharing each particle's optimal solution, so it is more suitable to be implemented in CPU.

Taking CUDA parallel programming for PSO as an example, the key codes are as follows.

Figure 13a shows the process of generating memories for CPU and GPU, which are used to store variables; Fig. 13b shows the kernel function to realize the updating of the speed, location, and individual optimal solution; and Fig. 13c shows the process of coping kernel function results from GPU memory to CPU memory, and then releasing previous memory spaces.

4.3 GA simulation result analyses

A set of the same algorithm parameters is used for different methods to ensure comparability. Set crossover probability to $pc = 0.8$, mutation probability to $pm = 0.15$, population size to 100, and iteration steps to 1000. Under random selection, the average running time of OpenMP is 88.7 ms. When setting



```
                    Msgs
/testpso/clk        1
/testpso/w_in       3F333333     3F333333
/testpso/c1_in      40000000     40000000
/testpso/c2_in      40000000     40000000
/testpso/gbest1     BD0C8DF5     BD8...  BD134708  B...  BD0C8DF5
/testpso/gbest2     B98C1CD5     3F1...  BCA712E8  3...  B98C1CD5
/testpso/bestfit    3A9A598A     3EC...  3ADFFA24  3...  3A9A598A
```

Fig. 15 Single particle of PSO simulation results in FPGA. FPGA uses VHDL language, and its operating frequency is 100 MHz. FPGA belongs to Xilinx Co. Ltd., and its model is XC5VLX50T. At the same time, the integrated development software is ISE, version is 14.1

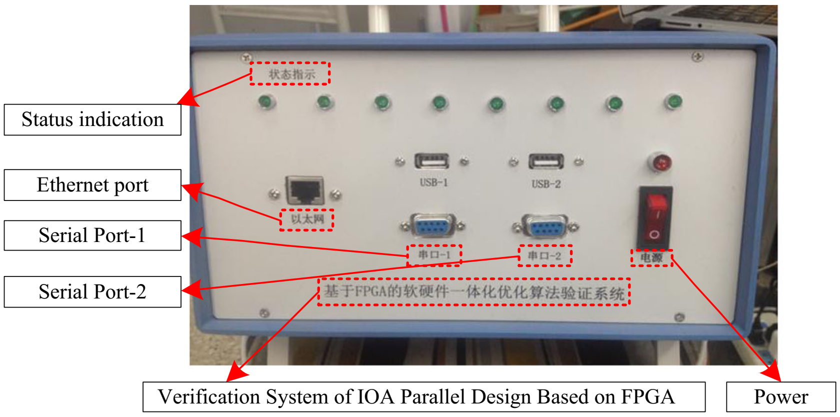
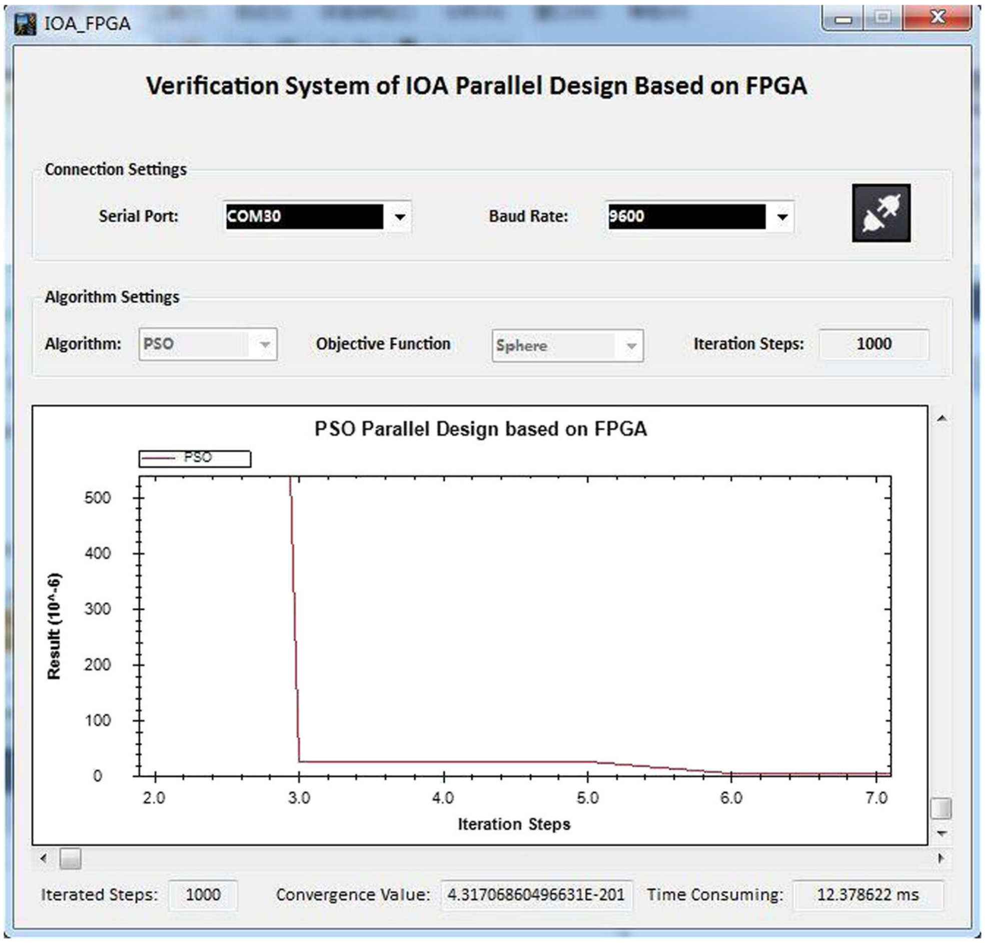


Fig. 16 The device to verify IOA based on FPGA

256 threads in each block, the average running time of GPU-based CUDA is 365.6 ms.

Next, FPGA-based time consuming of TSP is analyzed. Figure 14 is the simulation result of ten cities' TSP based on GA. From Fig. 14, we can see that the final convergence result is *4036f49a* (hexadecimal) = 2.8587 which is much close to ideal convergence result 2.6907.

Figure 14 is the result of the individual after multiple iterations under random selection. Without adding crossover operator, individual needs 22 clocks to complete one iteration. Each individual needs 2.5 clocks on average. At this situation, individual needs 22 + 2.5 = 24.5 clocks to complete one iteration. Under 100-MHz clock frequency, the time consuming of 100 individuals with 1000 iteration is 1000*×100*×24.5*×1000/100 MHz = 24.5 ms, which is shorter than the other two methods; it can be seen from Table 1. It can prove that FPGA-based method can significantly improve real time of GA.


IOA_FPGA

Verification System of IOA Parallel Design Based on FPGA

Connection Settings

Serial Port: COM30 Baud Rate: 9600

Algorithm Settings

Algorithm: PSO Objective Function Sphere Iteration Steps: 1000

Iterated Steps: 1000 Convergence Value: 4.31706860496631E-201 Time Consuming: 12.378622 ms

Fig. 17 The outcome of PSO based on FPGA. FPGA uses VHDL language, and its operating frequency is 50 MHz. FPGA belongs to Xilinx Co. Ltd., and its model is XC6SLX16. At the same time, the integrated development software is ISE, version is 14.4

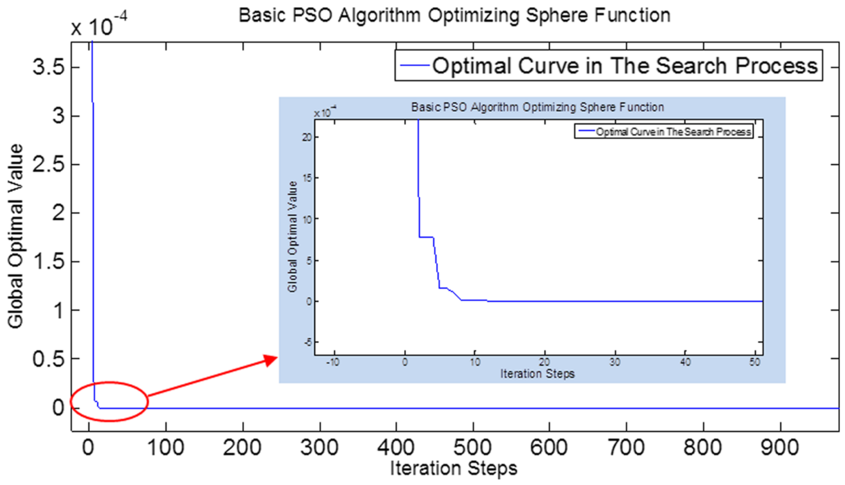


Fig. 18 The simulation result of PSO based on Matlab. Matlab runs in a computer, and its configuration is Acer, Intel® Core™ i5-3317U @ 1.70 GHz, 4G memory, and Windows 7 of 64-bits. The version of Matlab is 7.0

4.4 PSO simulation result analyses

Similarly, set same algorithm parameters for different methods to ensure comparability. The size of particle swarm is 100, iteration steps is 1000, function dimension is 2, acceleration constant is 2, inertia weight is 1, maximum initial position is 1, and maximum initial velocity is 3. After testing, the results are shown below.

The average running time of OpenMP based on multi-core processor is 62.3 ms. When setting 256 threads in each block, the average running time of GPU-based CUDA is 315.8 ms. GPU has many cores, so it should cost much less running time than OpenMP in theory. However, because it needs to call *curand_uniform* function which will cost longer time in order to generate uniformly distributed random numbers in the process of program execution, so its running time becomes longer than that of OpenMP.

In the FPGA-based PSO, each individual needs 20 clocks to complete one update, and one clock is 10 ns. Limited to internal resources of FPGA, the program just takes up one arithmetic unit. Therefore, if there are 100 individuals and 1000 steps iteration, the total time consuming is 1000*×100*×20 = 2,000,000 clocks. Under the condition of 100-MHz clock frequency, it will need only 2000000*×1000/100 MHz = 20 ms to complete one total calculation. The time consuming is less than the other three methods. Therefore, it can be considered that FPGA-based method could significantly improve the real time of PSO. With rapid increasing of hardware resources in FPGA, more arithmetic units can be called at the same time, so that the efficiency of FPGA execution could be greatly improved.

As shown in Fig. 15, the inertia weight is set to *3F333333* (hexadecimal) = 0.7, and the acceleration constant is *40,000,000* (hexadecimal) = 2. The final convergence result of PSO algorithm is *3A9A598A* (hexadecimal) = 0.0012, which is much close to ideal convergence result 0. It means that PSO algorithm parallel design based on FPGA has high real time.

At last, the time consuming of all the above-mentioned simulation results are shown in Tables 1 and 2.

The above-mentioned results are based on simulation. The following are the experimented results based on the real hardware and software. Figure 16 shows the development device.

As can be seen in Fig. 16, there are eight led lights on the top of the device to indicate the current state. There is a key on the right to control the power with a red light to indicate on/off state. There are also two USB ports, two serial ports, and one Ethernet port to communicate with computers. The final outcome is shown in Fig. 17.

Under the same parameter setting, Matlab-based PSO simulation result is shown in Fig. 18.

The comparison results of FPGA-based parallel PSO design and Matlab-based serial PSO design is shown in Table 3, both of the iterative steps are 1000. Obviously, FPGA has great advantages over Matlab in real time.

5 JSSP solving based on SA and FPGA

JSSP is a typical combinatorial optimization problem, and can be described as follows [39–41]: there exist *n* jobs needed to be operated on *m* machines. The jobs' processes have been

Table 3 Comparison of outcomes of PSO based on FPGA and Matlab

PSO	Convergence value	Time consuming	Frequency of clock
FPGA	4.31706860496631e-201	12.378633 ms	50 MHz
Matlab	5.4991e-210	227 ms	1.70 GHz

Table 4 Mathematical model parameter of JSSP

Parameter	Description
n	Number of jobs
m	Number of machines
i	Index of jobs ($i = 1,2,...,n$)
j	Index of machines ($j = 1,2,...,m-1$)
$O_{i,j}$	Job i in the j times process
$t_{i,j}$	Start time of $O_{i,j}$
$p_{i,j}$	Execution time of $O_{i,j}$
S	Scheduling sequence
C_{max}	Processing time (makespan)

determined at first, and each machine at most processes one job at any time which cannot be terminated during working.

The mathematical model of JSSP can be described as follows: There are n jobs $\{J_i\}_{1 \le i \le n}$, each job has certain technical constraints, these constraints form a matrix $Bound_{n \times m} = \{M_{ij}\}_{1 \le i \le n,\ 1 \le j \le m}$. Each job J_i is processing in a machine M_j is called operation O_{ij}. O_{ij} needs an uninterrupted time p_{ij}, in this time, J_i is alone in possession of M_j. These uninterrupted time form a matrix $T_{n \times m} = \{P_{ij}\}_{1 \le i \le n,\ 1 \le j \le m}$. A job shop scheduling process can be expressed as $S_{n \times m} = \{O_{ij}\}_{1 \le i \le n,\ 1 \le j \le m}$. The goal of JSSP is to find a scheduling sequence corresponding to shortest processing time (makespan)C_{max}. Parameter implication of JSSP is shown in Table 4.

Objective

$$minimize \quad C_{max}(S)$$

Constraint

(1) Job's procedure is determined, and job's processing cannot be terminated if start.
(2) Each machine can process only one job at a time.
(3) $t_{i,j+1} \ge$ the time of job i in the j times complete process
(4) $t_{i,j+1} \ge$ the last time of job i in the $j + 1$ times complete process
(5) $t_{i,1} \ge \max(0,$ the last time of job i in the first times complete process)

SA is simulating solid cooling process. Solution of objective function of SA can easy to jump out of local optimal solution, and eventually tend to the global optimal solution

Table 5 Comparison of outcomes of SA based on FPGA and Matlab

Items	Convergence value	Time consuming in one anneal step	Frequency of clock
FPGA	60	450 ms	100 MHz
Matlab	55	2070 ms	2.53 GHz

Matlab runs in a computer, and its configuration is Lenovo G460, Intel® Core ™ i3 CPU M380 @ 2.53 GHz, 2G memory, and Windows 7 of 32-bits. The version of *Matlab* is 7.9.0. *FPGA* uses VHDL language, and its operating frequency is 100 MHz. *FPGA* belongs to Xilinx Co. Ltd., and its model is XC5VLX50T. At the same time, the integrated development software is ISE, version is 14.1

[42, 43]. Therefore, using SA to solve JSSP is an effective method.

FPGA's advantage in IOA parallel design has been proved in above. This section is a case study, in which FPGA-based SA parallel design is used to solve JSSP.

First of all, the parameters of SA algorithm should be assigned. t_f referring to the terminate temperature is set 0.01. t_0 representing the initial temperature is set 500. *alpha*, a coefficient relevant to the cooling process is set 0.8.

As can be seen from Fig. 19, the cooling process repeats 6 cycles. The process times of these cooling processes respectively are *41*, *3c*, *3d*, *3f*, *3d*, and *3c* in hexadecimal, which equal to *65*, *60*, *61*, *63*, *61*, and *60* in decimal. The processing time corresponding to the initial solution of the algorithm is *8c*, which represents *140* in decimal. It can be concluded that after the execution of the algorithm, the satisfying schedule sequence can be found. The process time is reduced. However, the algorithm fails to reach the minimal process time which is *55*. Therefore, the convergence property needs to be improved. However, FPGA has significant advantage in real time in IOA parallel design (Table 5).

6 Conclusion and future works

In this paper, GA and PSO parallel design methods based on OpenMP and CUDA are proposed, and then FPGA-based parallel design of the above two algorithms is implemented. The results are compared and analyzed, and show that parallel design based on FPGA has significant advantage in real time. According to different practical solutions, the effective implementation steps are IOA parallelism analyze, ALU and CU

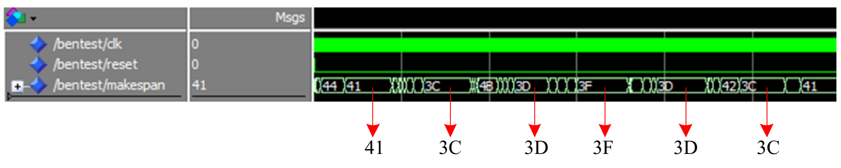


Fig. 19 The result of JSSP based on SA and FPGA

design in FPGA, and debug and iterative optimization, which are presented in general method of parallel design IOA based on FPGA in section 3. More specifically, a user-friendly interface that enables users to provide their own functions and integrate the FPGA-based algorithm seamlessly into their own solutions will be more effective methods. This work is authors' future research.

With the rapid development of FPGA technology, many FPGA chips have developed into SoC (System on Chip). In the future, it is expected to build a heterogeneous multi-core framework, which includes FPGA, DSP, ARM, etc., to accelerate IOA.

Funding information This work is financially supported in part by Beijing Natural Science Foundation No. 4152032, National Natural Science Foundation of China under Grant No. 51475032, Beijing Nova Program under Grant No. Z161100004916063.

References

1. Liu YL, Zhang Y, Zhao DM, Tao F, Zhang L (2012) Concept and framework of reconfigurable intelligent optimization algorithm. Adv Mater Res 479-481(5):1875–1879. https://doi.org/10.4028/www.scientific.net/AMR.479-481.1875
2. Wang DZ, Wu CH, Ip A, Wang DW, Yan Y (2008) Parallel multi-population particle swarm optimization algorithm for the uncapacitated facility location problem using openMP. IEEE Congress on Evolutionary Computation 1214–1218
3. Arenas MG, Mora AM, Romero G, Castillo PA (2011) GPU computation in bioinspired algorithms: a review. Adv Comput Intell: 433–440
4. Contreras I, Jiang YY, Hidalgo JI, Núñez-Letamendia L (2012) Using a GPU-CPU architecture to speed up a GA-based real-time system for trading the stock market. Soft Comput 16(2):203–215
5. Siano P, Cecati C, Yu H, Kolbusz J (2012) Real time operation of smart grids via FCN networks and optimal power flow. IEEE Trans Ind Inf 8(4):944–952. https://doi.org/10.1109/TII.2012.2205391
6. Li Z (2016) Wavelength selection for quantitative analysis in terahertz spectroscopy using a genetic algorithm. IEEE Trans Terahertz Sci Technol 6(5):658–663
7. Datta R, Pradhan S, Bhattacharya B (2016) Analysis and design optimization of a robotic gripper using multiobjective genetic algorithm. IEEE Trans Syst Man Cybern Syst 46(1):16–26. https://doi.org/10.1109/TSMC.2015.2437847
8. Wei H, Tang XS (2015) A genetic-algorithm-based explicit description of object contour and its ability to facilitate recognition. IEEE Trans Cybern 45(11):2558–2570. https://doi.org/10.1109/TCYB.2014.2376939
9. Ye M, Wang YP, Dai C, Wang XL (2016) A hybrid genetic algorithm for the minimum exposure path problem of wireless sensor networks based on a numerical functional extreme model. IEEE Trans Veh Technol 65(10):8644–8657. https://doi.org/10.1109/TVT.2015.2508504
10. Jiau MK, Huang SC (2015) Services-oriented computing using the compact genetic algorithm for solving the carpool services problem. IEEE Trans Intell Transp Syst 16(5):2711–2722. https://doi.org/10.1109/TITS.2015.2421557
11. Huang SC, Jiau MK, Lin CH (2015) Optimization of the carpool service problem via a fuzzy-controlled genetic algorithm. IEEE Trans Fuzzy Syst 23(5):1698–1712. https://doi.org/10.1109/TFUZZ.2014.2374194
12. Chan KY, Yiu CKF, Dillon TS, Nordholm S, Ling SH (2012) Enhancement of speech recognitions for control automation using an intelligent particle swarm optimization. IEEE Trans Ind Inf 8(4):869–879. https://doi.org/10.1109/TII.2012.2187910
13. Tang Y, Gao HJ, Kurths J, Fang J (2012) Evolutionary pinning control and its application in UAV coordination. IEEE Trans Ind Inf 8(4):828–838. https://doi.org/10.1109/TII.2012.2187911
14. Chan KY, Dillon TS, Kwong CK (2011) Modeling of a liquid epoxy molding process using a particle swarm optimization-based fuzzy regression approach. IEEE Trans Ind Inf 7(1):148–158. https://doi.org/10.1109/TII.2010.2100130
15. Zhao JH, Wen F, Dong ZY, Xue YS, Wong KP (2012) Optimal dispatch of electric vehicles and wind power using enhanced particle swarm optimization. IEEE Trans Ind Inf 8(4):889–899. https://doi.org/10.1109/TII.2012.2205398
16. Gong YJ, Shen M, Zhang J, Kaynak O, Chen WN, Zhan ZH (2012) Optimizing RFID network planning by using a particle swarm optimization algorithm with redundant reader elimination. IEEE Trans Ind Inf 8(4):900–912. https://doi.org/10.1109/TII.2012.2205390
17. Yao F, Dong ZY, Meng K, Xu Z, Iu HHC, Wong KP (2012) Quantum-inspired particle swarm optimization for power system operations considering wind power uncertainty and carbon tax in Australia. IEEE Trans Ind Inf 8(4):880–888. https://doi.org/10.1109/TII.2012.2210431
18. Tao F, Zhao DM, Hu YF, Zhou ZD (2008) Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. IEEE Trans Ind Inf 4(4):315–327. https://doi.org/10.1109/TII.2008.2009533
19. Luis J, Martínez F, Gonzalo EG (2011) Stochastic stability analysis of the linear continuous and discrete PSO models. IEEE Trans Ind Inf 15(3):405–423
20. Liu XX, Zhao RC, Han L (2013) A compile-time cost model for automatic OpenMP decoupled software pipelining parallelization. The 14th ACIS international conference on software engineering, artificial intelligence, networking and parallel. Distrib Comput:253–260
21. Preud'homme T, Sopena J, Thomas G, Folliot B (2012) An improvement of OpenMP pipeline parallelism with the BatchQueue algorithm. IEEE 18th International Conference on Parallel and Distributed Systems 348–355
22. Zheng Z, Chen XH, Wang ZY, Shen L, Li JW (2011) Performance model for OpenMP parallelized loops. International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE) 383–387
23. Tang T, Lin YS, Ren XG (2010) Mapping OpenMP concepts to the stream programming model. The 5th International Conference on Computer Science & Education 1900–1905
24. Karunadasa NP, Ranasinghe DN (2009) Accelerating high performance applications with CUDA and MPI. The 4th International Conference on Industrial and Information Systems, pp 331–336
25. Laan WJV, Jalba AC, Roerdink JBTM (2011) Accelerating wavelet lifting on graphics hardware using CUDA. IEEE Trans Parallel Distrib Syst 22(1):132–146. https://doi.org/10.1109/TPDS.2010.143
26. Thurley MJ, Danell V (2012) Fast morphological image processing open-source extensions for GPU processing with CUDA. IEEE J Sel Top Sign Process 6(7):849–855. https://doi.org/10.1109/JSTSP.2012.2204857
27. Zhang XR, Liu YH, Zhang F, Ren J, Sun YL, Yang Q, Huang H (2012) On design and implementation of neural-machine interface for artificial legs. IEEE Trans Ind Inf 8(2):418–429. https://doi.org/10.1109/TII.2011.2166770
28. Tao F, Cheng JF, Qi QL (2017) IIHub: an industrial Internet-of-things hub towards smart manufacturing based on cyber-physical system. IEEE Trans Ind Inf:1. https://doi.org/10.1109/TII.2017.2759178

29. Tao F, Cheng JF, Qi QL, Zhang M, Zhang H, Sui FY (2017) Digital twin-driven product design, manufacturing and service with big data. Int J Adv Manuf Technol. https://doi.org/10.1007/s00170-017-0233-1
30. Tao F, Zhang M (2017) Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing. IEEE Access 5:20418–20427. https://doi.org/10.1109/ACCESS.2017.2756069
31. Li JR, Tao F, Cheng Y, Zhao LJ (2015) Big data in product lifecycle management. Int J Adv Manuf Technol 81(1–4):667–684. https://doi.org/10.1007/s00170-015-7151-x
32. Tao F, Cheng JF, Cheng Y, SX G, Zheng TY, Yang H (2017) SDMSim: a manufacturing service supply-demand matching simulator under cloud environment. Robot Comput Integr Manuf 45: 34–46. https://doi.org/10.1016/j.rcim.2016.07.001
33. Gomperts A, Ukil A, Zurfluh F (2011) Development and implementation of parameterized FPGA-based general purpose neural networks for online applications. IEEE Trans Ind Inf 7(1):78–89. https://doi.org/10.1109/TII.2010.2085006
34. Agarwal A, Agarwal V (2012) FPGA realization of trapezoidal PWM for generalized frequency converter. IEEE Trans Ind Inf 8(3):501–510. https://doi.org/10.1109/TII.2012.2193406
35. Sanchez PM, Machado O, Bueno Peña EJ, Rodríguez FJ, Meca FJ (2013) FPGA-based implementation of a predictive current controller for power converters. IEEE Trans Ind Inf 9(3):1312–1321. https://doi.org/10.1109/TII.2012.2232300
36. Zou XF, Tao F, Jiang PL, SX G, Qiao K, Zuo Y, LD X (2016) A new approach for data processing in supply chain network based on FPGA. Int J Adv Manuf Technol 84(1–4):249–260. https://doi.org/10.1007/s00170-015-7803-x
37. Liu YL (2013) Implementation and application of parallel optimization algorithm based on FPGA. Beihang university
38. Tao F, LaiLi YJ, Xu LD, Zhang L (2013) FC-PACO-RM: a parallel method for service composition optimal-selection in cloud manufacturing system. IEEE Trans Ind Inf 9(4):2023–2033. https://doi.org/10.1109/TII.2012.2232936
39. Ge HW, Sun L, Liang YC, Qian F (2008) An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. IEEE Trans Syst Man Cybern A Syst Hum 38(2):358–368
40. Gao H, Kwong S, Fan BJ, Wang R (2014) A hybrid particle-swarm Tabu search algorithm for solving job shop scheduling problems. IEEE Trans Ind Inf 38(2):2044–2054
41. Ma PC, Tao F, Liu YL, Zhang L, Lu HX, Ding Z (2014) A hybrid particle swarm optimization and simulated annealing algorithm for job-shop scheduling. 2014 I.E. International Conference on Automation Science and Engineering (CASE), pp 125–130
42. Gomez D, Prieto F, Guzman M (2015) Nearest neighbors by adaptive simulated annealing. IEEE Lat Am Trans 13(7):2398–2404. https://doi.org/10.1109/TLA.2015.7273804
43. Trovão JPF, Santos VDN, Pereirinha PG, Jorge HM, Antunes CH (2013) A simulated annealing approach for optimal power source management in a small EV. IEEE Trans Sustain Energy 4(4):867–876. https://doi.org/10.1109/TSTE.2013.2253139