Int j simul model 13 (2014) 3, 323-334
ISSN 1726-4529 Original scientific paper

A PARALLEL OPTIMIZATION ALGORITHM FOR STEEL PLATE PICK-UP OPERATION SCHEDULING PROBLEM

Deng, X.-Y.
College of Business Administration, Huaqiao University, No. 269 Chenghuabei Road, Fengze District, Quanzhou, 362021, China
E-Mail: londonbell.deng@gmail.com

Abstract
The parallel computing is used for optimization modelling on multi-level nested genetic algorithm with superior scale to improve efficiency of the algorithm of steel plate operation scheduling problem. A parallel multi-layer genetic algorithm was designed based on the analysis of the present nested algorithm, and some key problems in parallel algorithm implementation were solved. An experimental example was illustrated. The results showed that the speedup of the proposed parallel algorithm comes up to 3.144 on a computer with quad-core processer and the total execution time is shortened greatly. The parallel nested genetic algorithm can meet the requirement for actual operation period.
(Received, processed and accepted by the Chinese Representative Office.)

Key Words: Operation Scheduling, Parallel Optimization, Parallel Genetic Algorithm, Steel Plate Pick-Up Operation

1. INTRODUCTION

The pick-up operation of steel plates in a shipyard is a process to select the required steel plates from the stockyard based on the production plan of the shipyard. Two major factors are considered in evaluating the optimization algorithms for steel plate pick-up operation scheduling. The first is the effectiveness of the algorithm result, which means that the pick-up operation plan can meet the requirement of minimization of operation cost or time. The second is the efficiency of the algorithm execution. Since the steel plate pick-up operation is a daily routine with heavy workload, the execution of algorithm should be practiced in a very limited time. For the researches on the effectiveness of optimization algorithms, most methods make use of multi-layer nested or hybrid evolutionary algorithms [1-3] to build a combinatorial optimization model for solution to the operation scheduling problem. In contrast, the researches on efficiency of optimization algorithm are relatively few. As the optimization model of steel plate pick-up operation scheduling is a complex and multi-layer combinatorial optimization model which can be regarded as the NP-hard problem, it is usually solved by employing the serial genetic algorithm (GA) [4]. With the increasing scale of the operation scheduling problem, the time of obtaining the effective plan grows very fast, and it is difficult to meet the efficiency requirement of making operation plans.

In recent years, the parallel computing technology has developed rapidly. Parallel computing refers to a form of computation where many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into numerous smaller ones [5]. Currently, parallel computing is widely used, from the compute-intensive problems like the large-scaled traditional scientific calculation to the data-intensive problems such as the large-scaled business search engine [6]. Many scholars have conducted a wide range of research on parallel genetic algorithm (PGA) [7]. Muhlenbein and others have pointed out advantages and disadvantages of PGA compared to GA through analysing deceptive problems and Traveling Salesman with PGA [8]. Spiessens and other scholars have achieved the parallel calculation with large-scale GA [9]. Oleg has claimed advantages of

parallel calculation over serial computation based on the study [10]. Liu Feng and others have obtained a pretty satisfactory result by applying the PGA to solve the complex function equation [11]. Tang Tianbing advocates a parallel GA Framework [12], while Yin Xiang and others propose a multi-task multi-ally problem generating model and parallel algorithm for solution [13].

Therefore, this paper introduces parallel optimization to build a parallel multi-layer optimization model based on PGA for the problem of steel plate pick-up operation scheduling. And this model can provide a new way of improving algorithm execution effectiveness for steel plate pick-up operation plan.

2. PROBLEM STATEMENT AND OPTIMIZATION MODEL

2.1 Problem statement

The problem of steel plate pick-up operation plan is aimed at minimizing the output time taken in choosing the steel plate in the stockyard according to the required type and amount and settling the sequence of the steel plate pick-up [3]. This part explains the related definitions and the optimization model of this problem.

Supposed that the steel plates pile in the stockyard is piled with r steel plates in a row and c in a line, the total number of the steel plate will be $A = r \times c$, and the coordinate b_i is represented by r_i and c_i, which is (r_i, c_i). The set of the pile location is $S = \{S_i \mid i = 1, 2, \ldots, A\}$, and we define the piling state of the plate is set Ω, where $\Omega = \{P(S_i) \mid i = 1, 2, \ldots, A\}$, in which $P(S_i)$ represents the piling state of steel plates in pile S_i, and $P(S_i) = \{p_{ij} \mid j = 1, 2, \ldots, N_i\}$, in which N_i is the number of plates in pile S_i. The state attribute of the steel plate p_{ij} is represented by the triple tuple (S_i, T_{ij}, F_{ij}), and T_{ij} is the number of layers, while F_{ij} is the type of the steel plate. F_{ij} is formed with four parameters, and it can be written as $F_{ij} = (l_{ij}, w_{ij}, h_{ij}, m_{ij})$, in which l_{ij}, w_{ij}, h_{ij} and m_{ij} are the length, width, thickness and material of steel plate p_{ij}. N^F is the number of the steel plate F. The total number of steel plates in Ω is N, and the type of the plate is F.

$$N = \sum_{i=1}^{A} N_i \tag{1}$$

Supposed that the row distance and column distance adjacent of the steel piles are D_r and D_c, then the distance D between two piles S_i and S_j can be calculated as it is shown in eq. (2).

$$D = D(S_i, S_j) = \sqrt{(r_i - r_j)^2 D_r^{\,2} + (c_i - c_j)^2 D_c^{\,2}} \tag{2}$$

The task operation of steel plate pick-up is specified in the operation table Q, which is defined in eq. (3). $Q_i(F_i, N^{F_i})$ represents that the amount of steel plates of type F_i needed is N^{F_i}, and O is the number of all types needed in the operation table Q. Therefore, the total amount of steel plates needed in Q is:

$$N = \sum_{i=1}^{O} N^{F_i}, \quad Q = \left\{ Q_i\left(F_i, N^{F_i}\right) \middle| i = 1, 2, \ldots, O \right\} \tag{3}$$

2.2 Optimization model

Deciding the operation plan of steel plate output involves two stages. The first stage ($P1$) is choosing plates from Ω, and the next stage ($P2$) settling the operation sequence of outputting steel and the location of the stack. The layer function and constraints of $P1$ depend on the deciding variables and the optimal solution of $P2$. The optimal solution of $P2$, on the other hand, relies on the deciding variables of $P1$. Thus, the module framework of the algorithm is shown in Fig. 1.

In Fig. 1, the deciding variables X and YR of the two stages stands for the steel sheet output set and the stack plan set respectively. Definitions are as follows:

$$X = (x_1, x_2, \ldots, x_i, \ldots, x_N) \tag{4}$$

$$Y = (y_{11}, y_{12}, \ldots, y_{1j}, \ldots, y_{ij}, \ldots, y_{Nj}, \ldots, y_{Nk_N}) \tag{5}$$

in which x_i represents the outputting steel plate chosen from the stockyard and $i = 1, 2, \ldots, N$ represents the outputting sequence of x_i. y_{ij} is the stack decision of moving the steel sheet j to get the required steel plate x_i.

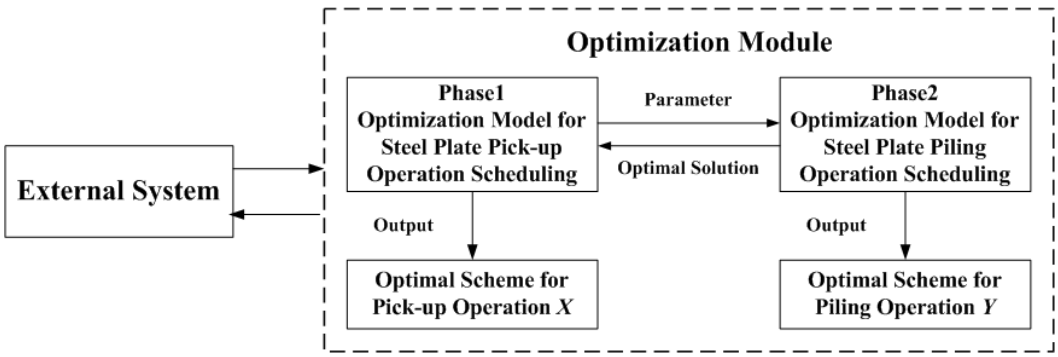

Figure 1: Module framework of the optimization algorithm.

If x_i is not at the top of the steel pile, the stacking operation should be conducted to remove steel plates on x_i. Suppose the set of stacking is R, then

$$R = \left\{ r_{ij} \,\middle|\, S(r_{ij}) = S(x_i), r_{ij} \in \Omega \right\} \tag{6}$$

Assumed that the set of steel plates to pick-up is $X = \{x_i | \, i = 1, 2, \ldots, N\}$, and x_i is the number i steel plate to pick-up. When the set equals to V, the least time spent is $G(X)$; when X, $F(X, Y)$. Y is the stacking plan of the steel plate output set X, and f is the operation time of getting the required steel plate x_i. $g(r_{ij}, y_{ij})$ is the time spent on stacking steel plate r_{ij} from the current pile $S(r_{ij})$ to the target pile y_{ij}. $h(x_i, S_0)$ is the time spent on moving the steel plate x_i from $S(x_i)$ to S_0. It can be categorized as a multi-layer combinatorial optimization problem aiming at minimizing the time spent on output operation. Target functions are as follows:

$$\min_{V \in \Gamma} G(X) = \min_{V \in \Gamma} \min_{X \in \Pi} F(X, Y) \tag{7}$$

$$\begin{aligned} F(X, Y) &= \min_{x_i \in X, y_{ij} \in C_j^{(i)}} \sum_{i=1}^{N} f(x_i, y_{i1}, \ldots, y_{ik_i}) \\ &= \min_{x_i \in X, y_{ij} \in C_j^{(i)}} \sum_{i=1}^{N} \left(\sum_{j=1}^{k_i} g(r_{ij}, y_{ij}) + h(x_i, S_0) \right) \end{aligned} \tag{8}$$

$$g(x_{ij}, y_{ij}) = t_0 + (t_1 + t_2) \times D(S(r_{ij}), y_{ij}) + t_3 \tag{9}$$

$$h(x_i, S_0) = t_0 + t_1 \times D(S(x_i), S_0) + t_2 \times D(S(x_{i+1}), S_0) + t_3 \tag{10}$$

where t_0 is the average time of the crane catching a plate, t_1 is the average time spent on moving a plate in a unit distance. t_2 is the average time spent on the crane's air travelling in a unit distance, and t_3 is the average time taken of the crane unloading a plate.

Accordingly, constraints are listed in the eqs. (11)-(13), in which eq. (11) represents the maximum of permitted safe operation height of the stack when the operation is on. Eq. (12) determines the scale of the stacking operation. Eq. (13) means that the same pile of steel plates should be of similarity in type.

$$\sum_{t=1}^{m_k} h(x_{kt}) + h(r_{ij}) \le h_{\max}, b_k \in C_j^{(i)}, 1 \le k \le m \tag{11}$$

$$D(S(r_{ij}), y_{ij}) \le D_{\max}, r_{ij} \in R \tag{12}$$

$$\begin{aligned} &(|\, l_{\max} - l(r_{ij}) \,| + |\, l(r_{ij}) - l_{\min} \,|) / 2l(r_{ij}) < \alpha_l \\ &(|\, w_{\max} - w(r_{ij}) \,| + |\, w(r_{ij}) - w_{\min} \,|) / 2w(r_{ij}) < \alpha_w \end{aligned} \tag{13}$$

where $h_{\max}$ is the maximum height of the pile, $h(x_{kt})$ is the thickness of x_{kt}, m_k is the number of plates in the current state of stack S_k, and $h(r_{ij})$, $l(r_{ij})$ and $w(r_{ij})$ represent the thickness, length and width of the stacking plates r_{ij}. $l_{\max}$ and $l_{\min}$ are the maximum and minimum length of the steel plate in stack r_{ij}, $w_{\max}$ and $w_{\min}$ are the maximum and minimum width of r_{ij}, and α_l and α_w are the threshold of the similarity level of plates' length and width.

2.3 Analysis of the algorithm complexity

The model optimization model built is the combinatorial optimization model, and it is usually solved by the hybrid optimization algorithm. The outer layer realizes the selection of plates, the middle layer realizes the sorting of α steel plates, and the inner layer realizes the stack deciding function. These three layers are of a sequence relation [3]. To analyse the operation efficiency of the algorithm, we tested the complexity of the current algorithm.

(1) Combinatorial analysis of the steel plates pick-up operation plan

For each Q_i in Q, if $n_i \le m_i$, then plates chosen from U_i have $C_{m_i}^{n_i}$ combinations. Γ is defined as the set of the combinations which meet the need Q of steel plate pick-up X in Ω, and the number of elements in Γ will be $\prod_{i=1}^{s} C_{m_i}^{n_i}$, where Π is defined as the set of feasible output sequence in X, and the number of elements in Π will be determined by the distribution of the output steel plates in the stockyard. If there is at least one output plate in each stack, the number of elements in Π will be $N!$. The problem is to select one combination of plates from Γ, and then settle an output sequence based on $N!$ feasible output sequences from the selected combinations. Thus we can see that the scale of this optimization is pretty large.

(2) The analysis of the times of running triple-layer hybrid structure algorithm

Suppose the number of population is Nx and algebra is Mx, each time conducting the steel sheet output plan means that the outer-layer computing will be conducted for O(Mx) times, the middle-layer computing will be conducted for O($Mx \cdot Nx$) times, and the inner-layer stacking computing will be conducted for O$(Mx \cdot Nx)^2$ times.

We cans see that the calculation is of high complexity, and results in a time consuming computation. To improve the operation efficiency, first we have to analyse the construction of calculation time in each layer. With different population and algebra, the result is in Fig. 2, in which the time of outer-layer algorithm refers to the operation time of the outer-layer calculation except the middle-layer algorithm operation time. Similarly, the middle-layer calculation time will not include the inner-layer calculation time, which is demonstrated in Fig. 2.

(1) Time cost of the reinitialization of the algorithm and the outer-layer calculation only takes 1 % of the total time spent. Thus there is no need to process it.

(2) The inner-layer calculation takes the 74 % of the total time, while the time middle-layer calculation including the inner-layer calculation takes up 98 % of the total time. It is the key to the optimization of algorithm.

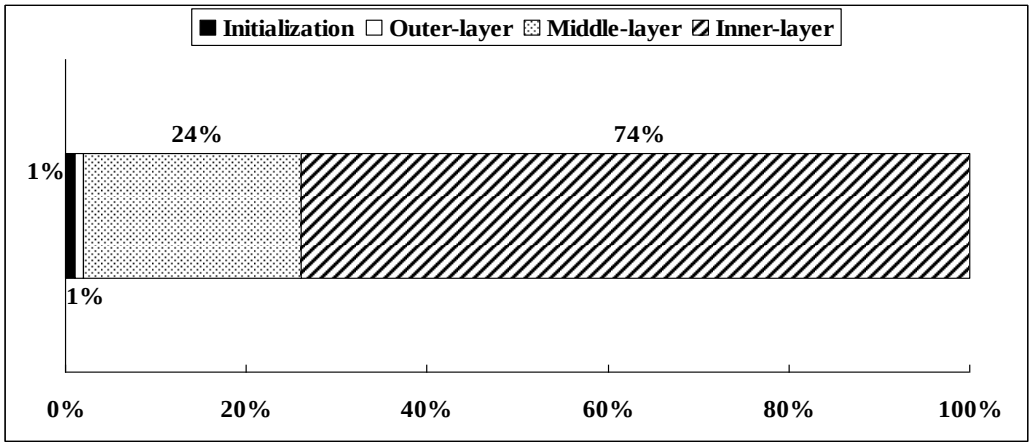

Figure 2: Running time comparison of three layer of optimization algorithm.

3. DESIGN AND IMPLEMENTATION OP PARALLEL OPTIMIZATION ALGORITHM

3.1 Selection of parallel computing model

Parallel computing model, or parallel model, refers to an abstract computing model from the analysis and the design of parallel algorithm, and it can be realized on all kinds of parallel computers. The parallel model of PGA can be categorized into 3 types: client-server model, coarse-grained model and fine-grained model. Functions are parallelizing the fitness of heredity individual, population analysis and all operations of the heredity respectively [6]. The parallel computing model with highest efficiency should be chosen in practice.

The client-server model of PGA parallel model is direct parallelization plan, and it refers to the parallel model in which the server conducts the overall operation with sequential operation, with only one group, initialization of population, selection, crossover and mutation, while the server and all clients parallelize the solution of individual fitness functions. The parallelizing process is shown in Fig. 3. The client-server model is easy to be achieved, and it is a very effective parallelization method when the calculation time of the algorithm is concentrated on the evaluation of fitness.

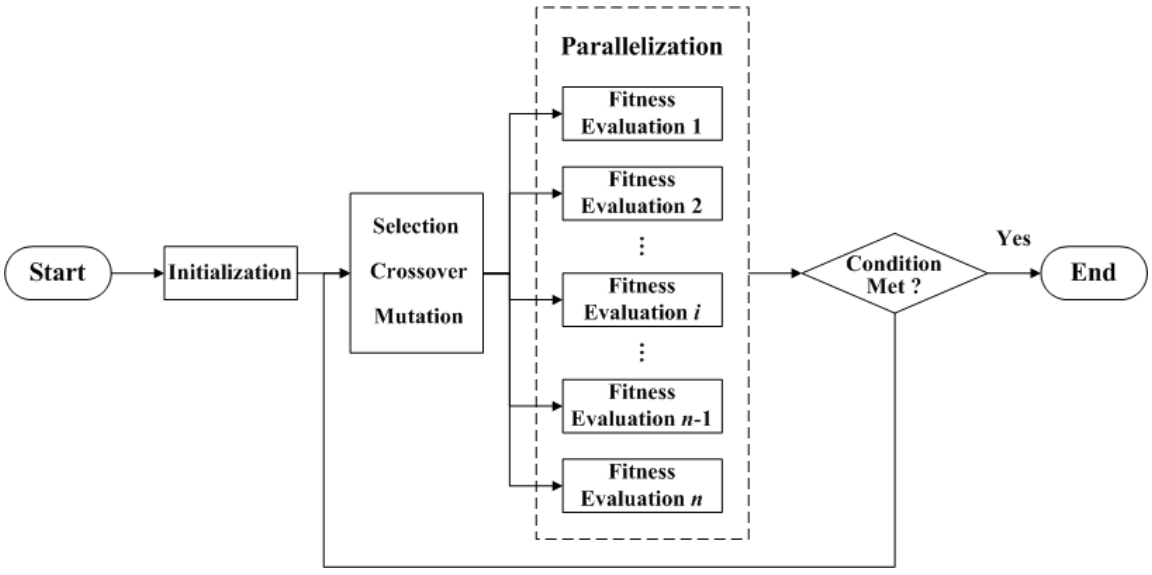

Figure 3: Client-server parallel model of PGA.

From Fig. 2 we can see that the middle-layer algorithm takes 98 % time of the total calculation time, and it is because the middle-layer algorithm is achieving the function of fitness evaluation of outer-layer algorithm. Thus, this paper applies the client-server parallel model to conduct the parallel optimization of algorithm.

After choosing the client-server parallel model, the nest step is to settle that the parallel optimization model should be built according to which layer. To evaluate the advantages of different plans, we introduce the evaluation indicators of the parallel optimization effect, which is the *SpeedUp* as it is shown in eq. (14):

$$SpeedUp = T_1 / T_n \tag{14}$$

where T_1 is the optimal sequential algorithm calculation time of the uniprocessor, T_n is the optimal parallel algorithm calculation time of n processors. The larger the speedup ratio is, the better the algorithm functions. Amdahl proposed a formula f with proportion relation of speedup, number of CPU of processors and calculation parts which cannot be analyzed by parallel algorithm, which is also known as Amdahl's law [13]. Suppose S is one part of the sequential operation of some calculation, P is part of the parallel operation, and $S+P=1$, then the maximum speedup value of a parallel computer with n processors can be calculated in eq. (15).

$$S(n) = \frac{S+P}{S+P/n} \tag{15}$$

It is clear that in eq. (15) the larger the S is, the smaller the $S(n)$ is. From Fig. 2, the S value is 2 % when the parallel optimization is conducted on middle-layer algorithm and inner-layer together. S value is 26 % if the parallel optimization is only conducted on inner-layer algorithm. Fig. 4 is according to the relation between speedup and the number of processors achieved in eq. (15) though conducting parallel optimization of inner-layer algorithm only and of middle-layer algorithm and inner-layer together. In Fig. 4, the effect of conducting parallel optimization on middle-layer algorithm and inner-layer together is more significant than the effect of conducting parallel optimization of inner-layer algorithm only. Therefore, the method of conducting parallel optimization on middle-layer algorithm and inner-layer together is chosen.

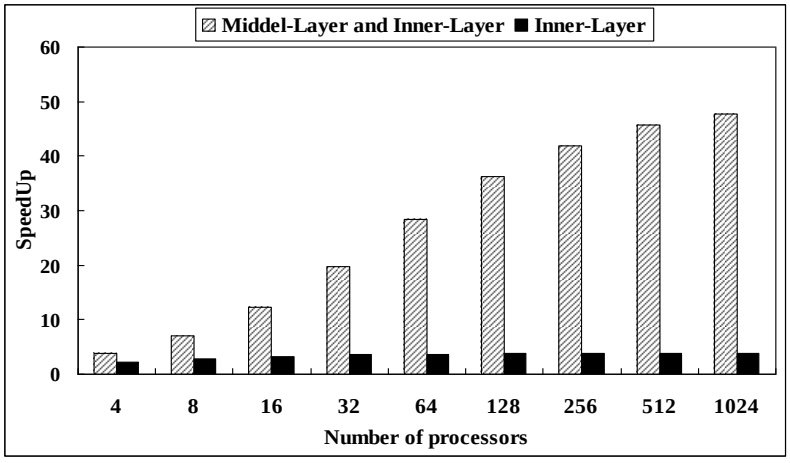

Figure 4: The ratio of speedup and the number of processors.

Conducting parallel optimization of middle-layer algorithm is the parallelization of the individual fitness of the outer-layer PGA. The flowchart of algorithm after parallel optimization is shown in Fig. 5. The left column in Fig. 5 represents the algorithm parallel calculation of the middle-layer sorting, and the right column is the flowchart of the single

middle-layer algorithm. Two problems should be solved as choosing the client-server parallel model: the load unbalancing of client and server processors, access latency of sharing data. Solutions will be provided in the next part.

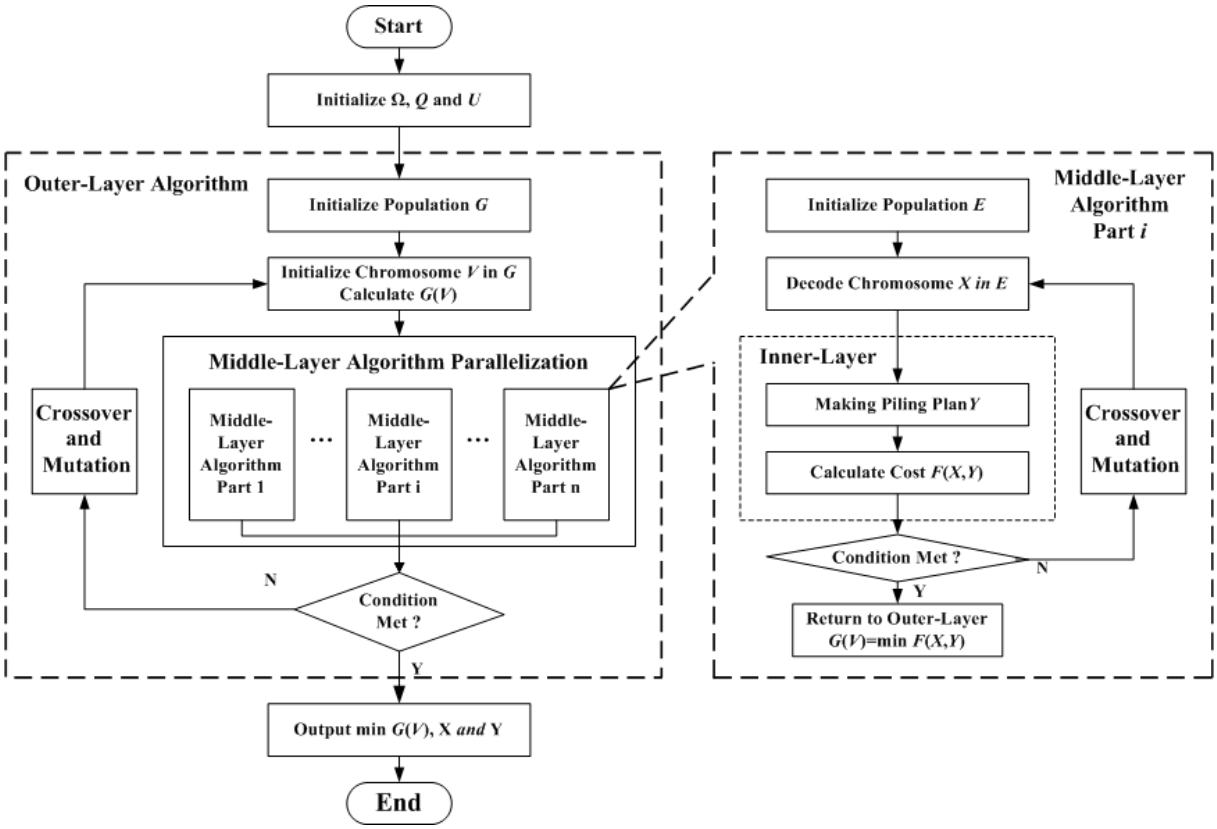

Figure 5: Procedure of parallel optimization algorithm.

3.2 Design of the load balancing

The problem of load balancing refers to the calculation work load balancing of processors of a parallel computer. We choose to conduct the parallel calculation of the middle-layer algorithm. The work load of middle-layer algorithm calculation is large, and the cost of thread scheduling is much smaller than the cost of middle-layer algorithm, thus the speedup room is larger. Factors influencing the load balancing include the evenness of dividing the thread, certainness of the operation time of thread and the number of cores in CPU [14]. To achieve a better load balancing, this paper divides the parallel parts and tries to ensure each processor conducts equal amount of calculation. The division is as follows:

(1) Suppose each time the middle-layer algorithm allocated to one CPU, the times is P, after the CPU completed the work, the rest times of middle-layer algorithm is Q;

(2) If $P<Q$, P times of middle-layer algorithm will be allocated to CPU. Otherwise, Q times are allocated to this CPU, and the parallelization part ends. Conducting the test to the dividing plan, suppose $P = 3$, $n = 4$, and the number of outer-layer population is $Nx = 100$, the time of each CPU processing the middle-layer algorithm is shown in Fig. 6.

We can know from Fig. 6 that the frequency of each CPU conducting the middle-layer algorithm is basically the same, from the original 100 times to 27. The value of speedup reaches 3.70, which means a better load balancing can be achieved.

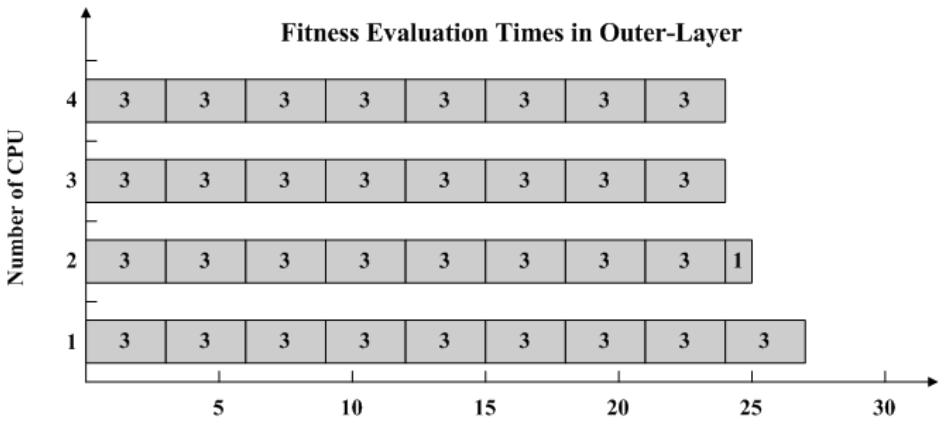

Figure 6: CPU loading graph.

3.3 Design of shared data access

Besides, there is another important problem in parallel algorithm design, which is the competition of multithread with data sharing. Most of the Parallel programs involve reading and typing data at the same time. These data are shared data. In the process of conducting parallelization of middle-layer algorithm, the shared data include: steel plate in stock information Ω, set of optional steel plates, set of stack location B and stack location and the steel plate information at the location $P(B)$. If multiple threads conduct reading and typing function to the shared data, competition will occur, which will further cause the low efficiency of the operation and even data errors. To improve algorithm operation efficiency and at the same time ensure the effectiveness of the result, we should avoid shared data errors in reading and typing. Generally, two methods are used to avoid the errors occurred in typing operation of shared data in parallel algorithm: to lock the typing function or to avoid typing same data.

The method applied in this paper is to avoid typing same data. When the time spent on locking is less than the time spent on avoiding typing, we will choose the first method, and minimize the locking data.

A similar problem with the shared data competition is the false sharing. It is caused by the two different memories of the same Cache at the same CPU sharing the data in one memory. To resolve the problem of false sharing, first we should check which areas are in the same Cache row in the given memory. There are many methods to solve the false sharing problem, but the one adopted in this paper is locking when the problem of false sharing might occur in the reading and typing operation. This method is simple, memory saving and the added time of the access latency is little.

3.4 Implementation of Algorithm shared data access

This parallel optimization algorithm is achieved by using Thread Building Blocks (TBB). TBB is a database which can support the expandable parallel programming. It has not only a good function of parallel optimization and expandability, but also with high feasibility, and well suits the parallel program developing of multi CPU [14]. Algorithm module achieved in the process of programming parallel hybrid algorithm with TBB is shown in Fig. 7, and the key steps are as follows:

(1) Under the parallel programming condition, distributor of TBB expandable memory can be used to manage the memory to solve the reading and typing problems;

(2) Task scheduler of TBB is used to schedule the parallelization of calculation task;

(3) With a certain parallel grain, parallelize the parallel object;

(4) Realize the hybrid multi-layer parallel optimization.

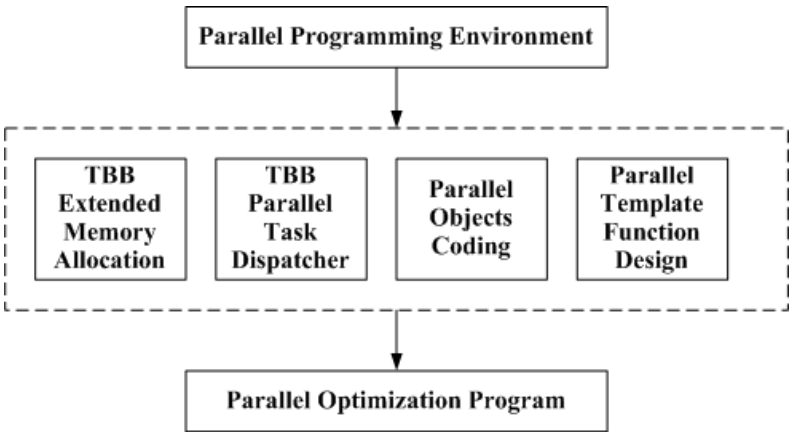

Figure 7: Parallel optimization algorithm module.

The key of this paper is the parallel iteration parallel optimization of the outer-layer algorithm individual fitness. The function of all individual fitness evaluation in the population is Sga_Selecting::evaluate(). Parallel_for module is used to analyse the outer-layer PGA population individual fitness. Parallel_for is a cycling iterative method of TBB, and is the simplest form of expandable parallel, in which all the iterative can be operated at the same time without influencing each other.

To solve the problem in this paper, we use Parallel_for module to realize the parallelization:

(1) Build a parallel object ParallelDo, and specify its functions to conduct the population individual fitness evaluation of outer-layer PGA;

(2) Analyse the variable involved in the function and steps of individual fitness evaluation in the sequential programming;

(3) Design the membership variable of object ParallelDo, such as the information of steel plates in stock allPlaP and the set of stack location in stockyard stackNoP;

(4) Design the method function of object ParallelDo, such as building a function, transfer ParallelDo() of related variables;

(5) Create the constructor of ParallelDo and reinstall the membership variables;

(6) Program the operator() method of ParallelDo: first to settle the iteration and set the method as constant. In the function system, realize the solution seeking of the objective function value of individuals in the population;

(7) Settle the parameters of the parallel module function Parallel_for() and use parallel_for() method in the fitness evaluation function Sga_Selecting::evaluate() of the outer-layer PGA population.

4. EXPERIMENTS

4.1 Experiment design

To test the operation efficiency of the hybrid multi-layer parallel PGA, we compare the algorithm proposed in this paper with sequential algorithm in literature [3]. Data including the stockyard and operation parameters are presented in Table I, and the operation task table is shown in Table II. Numbers of first 10 types of steel plates in 26 types are listed in Table I.

The experiment is carried out on a computer with Intel E3-1230 CPU (3.2 GHz) and 8 GB physical memory, and the program is coded with Visual C++.

Table I: Parameters of steel plates operation in stockyard.

Parameters	**Values (Unit)**	**Parameters**	**Values (Unit)**
r	10 (row)	t_2	1 (second/meter)
c	6 (column)	t_3	35 (s)
M	624 (piece)	m_1	28 (piece)
F	76 (type)	m_2	40 (piece)
d_r	20 (meter)	m_3	30 (piece)
d_c	5 (meter)	m_4	26 (piece)
α_l	0.2	m_5	30 (piece)
α_w	0.15	m_6	29 (piece)
$h_{\max}$	2 (meter)	m_7	22 (piece)
$d_{\max}$	50 (meter)	m_8	26 (piece)
t_0	30 (second)	m_9	13 (piece)
t_1	1.2 (second/meter)	m_{10}	30 (piece)

Table II: Data of operation task.

Specifications of Steel Plate	**Steel Plate Batchlist Q_i and Q_i's requirement n_i**									
	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
F_1	6	7	6	7	6	8	9	10	10	10
F_2	4	5	5	6	4	6	8	9	10	8
F_3	0	5	5	5	5	5	7	9	10	10
F_4	0	3	4	4	7	4	9	10	10	8
F_5	0	0	8	3	6	5	6	7	7	7
F_6	0	0	2	6	5	4	8	7	7	7
F_7	0	0	0	5	4	8	4	9	10	10
F_8	0	0	0	4	5	7	10	6	6	6
F_9	0	0	0	0	5	6	2	5	5	10
F_{10}	0	0	0	0	3	5	3	4	4	8
F_{11}	0	0	0	0	0	2	4	9	6	6
F_{12}	0	0	0	0	0	0	0	5	5	10
N	10	20	30	40	50	60	70	80	90	100

Suppose the crossover rate and the mutation rate of PGA are 0.02 and 0.6 respectively [3], with the outer-layer GA demography number as $Nx = 64$ and number of heredity generation is $Mx = 200$, then in the parallel algorithm, each time the middle-layer algorithm to one CPU, the operation frequency is $P = 3$ with the core number of the CPU as $n = 4$.

4.2 Experimental result

Solutions are made with parallel algorithm and sequential algorithm according to the different 10 tasks respectively. Calculation time of the algorithm is compared and tested, and the average operation time of the 10 operation results is used as the operation time of the algorithm. Speedup is calculated according to eq. (12), and the results are listed in Table III.

From Table III, we can know that the PGA speedup can reach 2.933 to 3.144 in a quad-core CPU condition. With the scale of the problem becomes larger, the effect of speedup will be better, which means the requirement of timeliness of the actual calculation can be met.

Table III: Execution time comparison of GA and PGA.

Steel Plate Batchlist	Problem Scale *N*	GA (second)	PGA (second)	*SpeedUp*
Q_1	10	56.28	19.18	2.933
Q_2	20	92.43	30.92	2.989
Q_3	30	222.71	72.64	3.066
Q_4	40	304.81	101.47	3.004
Q_5	50	418.26	139.02	3.008
Q_6	60	476.53	152.33	3.127
Q_7	70	504.03	161.39	3.125
Q_8	80	544.25	173.49	3.141
Q_9	90	647.33	207.01	3.127
Q_{10}	100	734.25	233.53	3.144

5. CONCLUSION

This paper analyses the hybrid parallel multi-layer optimization model of the steel sheet output operation plan, and proposes a method of applying the client-server model to conduct the parallel optimization of the hybrid multi-layer calculation of PGA. Problems of realizing the parallel algorithm are solved, including load balancing and shared data competition. The experiment has shown that the highest speedup of this parallel optimization algorithm on a computer with quad-core CPU can reach 3.144, which can significantly save the execution time of hybrid optimization algorithm and meet the timeliness requirement of the actual practice. The parallel optimization algorithm proposed in this paper provides a new way to raise the computing efficiency of the large scale steel sheet output plan algorithm.

6. ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation (No.71240002).

REFERENCES

[1] Hansen, J.; Kristensen, T. F. (2003). *Crane scheduling for a plate storage in a shipyard: Modelling the problem* (Technical Report), Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen

[2] Jin, C.; Liao, Y.; Huang, Y. (2010). Optimal layout model for steel plate yard with shipbuilding production sequence constraints, *Industrial Engineering and Management*, Vol. 15, No. 3, 49-54

[3] Jin, C.; Wang, G.; Gao, P. (2009). Modeling and optimization on operation scheduling for a steel plate yard in a shipyard, *Industrial Engineering and Management*, Vol. 14, No. 6, 12-17

[4] Ben-Ayed, O.; Blair, C. E. (1990). Computational difficulties of bilevel linear programming, *Operations Research*, Vol. 38, No. 3, 556-560, doi:10.1287/opre.38.3.556

[5] Asanovic, K.; Bodik, R.; Catanzaro, B. C.; Gebis, J. J.; Husbands, P.; Keutzer, K.; Patterson, D. A.; Plishker, W. L.; Shalf, J.; Williams, S. W.; Yelick, K. A. (2006). *The landscape of parallel computing research: A view from Berkeley* (Technical Report), University of California, Berkeley

[6] Mishra, B. S. P.; Dehuri, S. (2011). Parallel computing environments: A review, *IETE Technical Review*, Vol. 28, No. 3, 240-247
[7] Lim, D.; Ong, Y.-S.; Jin, Y.; Sendhoff, B.; Lee, B.-S. (2007). Efficient hierarchical parallel genetic algorithms using Grid computing, *Future Generation Computer Systems*, Vol. 23, No. 4, 658-670, doi:10.1016/j.future.2006.10.008
[8] Muhlenbein, H. (1991). Evolution in Time and Space – The Parallel Genetic algorithm, Rawlins, G. J. (Ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, 316-337
[9] Spiessens, P.; Manderick, B. (1991). A massively parallel genetic algorithm: Implementation and first analysis, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, 279-286
[10] Shylo, O. V.; T. Middelkoop, T; Pardalos, P. M. (2011). Restart strategies in optimization: parallel and serial cases, *Parallel Computing*, Vol. 37, No. 1, 60-68, doi:10.1016/j.parco.2010.08.004
[11] Tang, T.; Wei, L.; Xie, X.; Yan, Y. (2011). Study on distributed parallel computing on hybrid genetic algorithm, *Computer Engineering and Applications*, Vol. 47, No. 9, 207-209
[12] Yin, X.; Jiang, J.; Xia, N.; Chang, C. (2008). Multi-task multi-coalition generation problem: Model and algorithm, *System Engineering – Theory & Practice*, Vol. 28, No. 4, 90-95, doi:10.1016/S1874-8651(09)60021-1
[13] Yavits, L.; Morad, A.; Ginosar, R. (2014). The effect of communication and synchronization on Amdahl's law in multicore systems, *Parallel Computing*, Vol. 40, No. 1, 1-16, doi:10.1016/j.parco.2013.11.001
[14] Pedemonte, M.; Nesmachnow, S.; Cancela, H. (2011). A survey on parallel ant colony optimization, *Applied Soft Computing*, Vol. 11, No. 8, 5181-5197, doi:10.1016/j.asoc.2011.05.042