

Permutation-based Particle Swarm Algorithm for Tasks Scheduling in Heterogeneous systems with Communication Delays

Xiaohong Kong^{1,2}, Jun Sun¹ and Wenbo Xu¹

¹⁾ *School of Information Technology, Southern Yangtze University
Wuxi, Jiangsu Province 214122, China
xwb@sytu.edu.cn*

²⁾ *Henan Institute of Science and Technology, Xinxiang,
Henan Province 453003, China
nancykong@hist.edu.cn*

Abstract: A distributed heterogeneous system consists of a suite of processors or machines with different processing capacities. It can be performance-to-cost efficient to meet the diverse computation requirements if properly deployed. Task scheduling is a crucial issue to improve the efficiency of this architecture. In this paper, we incorporate a miscellaneous population-based search technique, Particle Swarm Optimization (PSO), with list scheduling and develop an alternative PSO algorithm for multiprocessor tasks scheduling. The framework of the hybrid PSO algorithm for the multiprocessor scheduling is developed according to the permutation-based solution representation to find optimal task-machine pairs. The merit of the hybrid method is to keep improving the potential solution continuously through learning experience from the searching knowledge of one particle itself or the best of all particles in the swarm in the interest of minimizing the scheduling length. We also implement a few assigning rules to select target machine with different processing speeds. The proposed model considers arbitrary networks in more realistic environments and the scheduling method takes communication delays into account. The results show that the proposed algorithm is able to find good quality schedules in reasonable time and has some advantages over other algorithms.

Keywords: Distributed computing, Task scheduling, DAG, Particle Swarm Optimization, Heterogeneous system, Permutation-based representation.

1. Introduction

Thanks to the standardization of communication channels between heterogeneous systems, along with the availability of wide band-width, distributed computing is a promising approach in solving the computing-intensive and data-intensive problems. A parallel application can be decomposed into subtasks which have diverse computation requirements executed on distributed heterogeneous machines concurrently. Heterogeneous Computing System

(HCS) consists of a suite of heterogeneous processors or machines. These machines may be different in processing capacity, architecture, topology network, operation system, and so on. Distributed computing aims to increase the computing power utilizing the potential parallelism inside an application, decrease the completion time and satisfy computing requirements.

One of the critical challenges in this field is the matching and scheduling of subtasks. Matching tackles the problem to select suitable machines for the subtasks through using some mapping strategies in order to achieve high performance. Scheduling, including tasks scheduling and messages scheduling, determines the execution order of subtasks on same machine and computes the starting time and the finish time. It has become apparent that in order to apply this new computing framework to real-world problems and data, researchers have to pay attention to the problems of satisfying diverse computing requirements and communication delay incurred by the data transfer between machines.

The scheduling problem in multiprocessor systems is NP-hard [1]. It becomes more complex in distributed heterogeneous computing environments considering heterogeneous interconnections, interfaces, operating systems, communication protocols, and programming environment. Currently many researchers pay more attention to heuristics due to the complexity [2-19]. In this paper, we devise a hybrid Particle Swarm Optimization (PSO) algorithm for the scheduling problem. In the proposed algorithm, list scheduling is combined with a permutation-based PSO algorithm to find out an optimal or near-optimal scheduling length.

The structure of the paper is as follows. In the next section the problem statement is provided. In section 3 we review the current scheduling algorithms. In section 4 we give a brief survey of Particle Swarm Optimization

algorithm. In section 5 we present the hybrid PSO algorithm for task scheduling with the permutation-based solution representation. In Section 6, the experiment results and discussion is given. Finally, we make a conclusion remarks.

2. Problem statements

In general, the distributed computing model can be represented as a directed acyclic graph (DAG). Defining a tuple $G = (T, E, C, W)$ to describe the graph, there T is a set of task nodes and E is a set of communication edges, corresponding to the data dependence among tasks. Each task $n_i \in T$ is assumed to have an estimated execution time $w(n_i)$. An edge is associated with an estimated communication cost $c(n_i, n_j)$, representing the amount of data units transferred from n_i to n_j . The edge $e_{i,j} = (n_i, n_j) \in E$ represents the partial order between tasks n_i and n_j , defining the node n_i as parent node and the node n_j child node. Partial order dictates that task n_j cannot be executed unless all its parents have been executed. If two tasks are scheduled to the same processor, the communication cost between them is assumed to be negligible. A node without parent node is entry node and a node without child node is exit node. A node can be executed, named ready node or runnable node.

A path from n_i to n_j (more than one) is defined as including all nodes and all edges between n_i and n_j . The length of a path is the sum of all the node and edge weights along the path. The longest path in the DAG is defined as Critical Path (CP). If there is a path from n_i to n_j and n_i is not the parent node of n_j , n_i is defined the predecessor node of n_j and n_j is the successor node of n_i . Note that the term "node" denotes task, and the "cost" is similar to "time". Tasks are basic processing units and can be processed non-preemptively by any processor in distributed systems. Here, execution cost $w(n_i)$ and communication cost $c(n_i, n_j)$ are nominal and the actual time must be computed in real scheduling environments.

The target system is consisted of processors with local memory, connected by an interconnection network with a certain topology. A model for heterogeneous computing systems is presented based on a coefficient-of-variation technique using ETC matrices to express heterogeneity [20]. Task heterogeneity is further studied in [4], characterized by consistent, partially-consistent, and inconsistent. In the heterogeneous architectural model used here, we assume a dedicated communication device is used that interprocessor

communication time can be overlapped with computation time in the schedule process. For simplicity, we also assume a set of processors P with the communication channels enough wide so that interprocessor communications perform without contention. In order to measure the machine heterogeneity, we define the heterogeneity factor matrix H like the method used in literature [3]. So the execution time of task n_i on P_j ($P_j \in P$) is given by $w(n_i) * h(ij)$ and $h(ij)$ mean the heterogeneity factor of task n_i on processor P_j . When two tasks n_i and n_j are allocated to distinct processors, the cost associated with the communication of $c(n_i, n_j)$ data units is $c(n_i, n_j) * L_{mn}$,

where L_{mn} is the average latency time per byte incurred on the links between processors m and n . All communication times and computation times also are known in advance such that scheduling can be calculated at compile time.

Scheduling the graph G to target system is to find out pairs of (task, processor) which optimize the scheduling length or completion time. When the task n_i is mapped to machine P_j , denote the start time and the finish time $ST(n_i, P_j)$ and $FT(n_i, P_j)$ respectively. The scheduling length or completion time is defined as $SL = \max(FT(n_i, P_j), (i, j))$.

3. Related works

Incorporating a variety of available resources, distributed heterogeneous computing is performance-to-cost efficient to improve the computational capacity in parallel system. But it is hard that different architectural attributes, such as system topology, message routing strategy, computation, and processors heterogeneity, are orchestrated to perform an application. Inappropriate scheduling of tasks can fail to exploit the true potential of a distributed system due to significant overhead if communication delays are considered, and can offset the gains from parallelism. It is infeasible to solve the problem using exact algorithm such as enumeration method when problem scale is larger. A number of heuristic scheduling algorithms have been extensively investigated to achieve sub-optimal schedule or near optimal schedule. These algorithms can be classified into list scheduling, scheduling based on duplication and clustering scheduling [2, 3, 5, 13, 14, 15, 17]. They are also be deemed arbitrary network and fixed network. A wealth of information can be found in the scheduling literatures [3, 15, 17].

The basic rationale of list scheduling is to make a scheduling list (a sequence of tasks for scheduling) by assigning them some priorities, and then assign the tasks to processor according to some rule such as the earliest start

time first. The main difference in these algorithms is the method to compute the node priorities. In list scheduling, the most generality is level-based strategy. Various levels about DAG graph were discussed in literatures [2,3,5,11, 13]. The t -level (top level) and b -level (bottom level) are two frequently used *levels* for DAG scheduling. The t -level of node n_i is the length of the longest path from entry node to n_i . As such, the b -level of node n_i is the length of the longest path from n_i to exit node and bounded by the length of the Critical Path. In addition, the two *levels* are often combined to determine the node priority such as b -level- t -level. These *levels* are also dynamically varied when the task is allocated to different machines.

The level-based method is simple, easy to implement and very efficient in homogeneous scheduling text, but they have some changes when transplanted into heterogeneous environments. Most algorithms use the mean values across all the processors and links as the priorities of nodes [2, 5, 11, 13]. The computation cost and communication cost are substituted with the mean computation value across all the machines and the mean communication value over all the possible communication links. Based on the *levels*, Topcuoglu et al. presented two efficient heuristics, the Heterogeneous Earliest-Finish-Time (HEFT) algorithm and the Critical-Path-on-a-Processor (CPOP) algorithm for scheduling directed acyclic weighted task graphs (DAGs) on a bounded number of heterogeneous processors. Sinnen and Sousa compare eight priority schemes for the node order determination in list scheduling and identify the best priority scheme [18]. The DLS (Dynamic Level Scheduling) algorithm dynamically modified priorities to match tasks with processors throughout the scheduling process [13].

The mean value could be sound if the computation time is proportional to processor speed and the speed is the only heterogeneous factor. When the heterogeneity among processing elements (PEs) is conspicuous, nevertheless, this might incur inefficient scheduling decisions. Mean value heuristics do not scale well and cannot handle problems of moderate size.

The key feature of Scheduling based on duplication is reducing the communication delays by redundantly allocating some parent nodes to the same processor as the child node. The disadvantages of the method are increasing the processor workload and decreasing the efficiency. The clustering scheduling first classify the tasks into a few task subsets, then assign the all tasks in a subset onto the same processor and decide the executing order. In all algorithms mentioned above, the task executing order must comply with the partial order.

Over the past years, the advent of metaheuristic methods or the new evolutionary algorithms including Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithm

(GA), has had a significant impact on solving the scheduling problem [4,6,7,8,9,16]. The common features of these algorithms are starting with initial solution and updating through successive iteration process to improve the solution.

GA is based on the mechanisms of evolution and natural genetics and has been successful to solve the multiprocessor scheduling [6, 7, 8]. SA and TS both are local methods searching for better solutions through neighborhood move on current solution. SA, originally simulating the cooling process of solid's evolution to thermal equilibrium [21], is a powerful stochastic search method to combinatorial optimization problems. The algorithm accepts the better new solution unconditionally and the bad new solution probabilistically to escape the local minimum trap. This algorithm has been proven to be a good technique for a lot of applications [16]. This paper employs simulated annealing as a select technique.

Defining a neighborhood move as transforming current solution into another solution, a subset of solutions called the neighbor can be generated. TS algorithm starts with a feasible solution and keeps improving it in successive iterations by exploring its neighborhood, finally the best candidate instead of the current solution [22]. A key in tabu search is constructing tabu list which follows track of recently visited solutions to avoid cycling or getting trapped in a local optimum. The tabu algorithm has gradually been utilized in scheduling domain [23].

Tracy et al. made a comparison of eleven solution-solving schemes for the independent tasks scheduling and demonstrated that GA and SA had better performance than TS [4]. In addition, the metaheuristic methods generally outperformed the exact or heuristic methods. But the research consider simplified model, in which the tasks are arbitrary order and have no data dependency.

However, there is little evidence for choosing one among different heuristic rules, and no priority rule dominates all other or performs consistently better than others. Moreover, the general heuristic methods may be trapped within local optima. An alternative method is developing a hybrid algorithm combining several algorithms above.

Swarm intelligence is gradually employed to solve combinatory optimization problem in past several years, including Ant Colony System and Particle Swarm Optimization [24-30]. A hybrid algorithm with a local search is proposed for the job-shop scheduling to find the minimum makespan [24]. Tasgetiren et al. introduce a SPV rule for a few sequencing and scheduling problems based on priority-based solution representation [25, 26], which is similar to random keys in genetic algorithm [27]. Using the PSO algorithm, Wang et al. proposed the concept of Swap Operator to solve Travel Salesman Problem [29]. Zhang et al. develop a solution-solving scheme for the resource-constrained project scheduling problem (RCPSP) in view of minimizing project duration [30].

4. Particle swarm optimization

Particle Swarm Optimization (PSO) [31,32], originally proposed by Kennedy and Eberhart in 1995, is a novel evolutionary algorithm and a swarm intelligence computation technique. Comparing well with other approaches such as genetic algorithms [33], they have been reported to solve a wide range of optimization problem including neural network training and function minimization [12,29,30,34,35]. In a PSO system, a particle or an individual, depicted by its position vector X and its velocity vector V , is a candidate solution to the problem. Considering D dimensions and NP population as an example, the velocity and position for the i th ($i \in [1, NP]$) particle is represented as $V_i = (v_{i1}, \dots, v_{id}, \dots, v_{iD})$ and $X_i = (x_{i1}, \dots, x_{id}, \dots, x_{iD})$ respectively.

In a standard PSO algorithm, a population of initial solutions search through a multidimensional solution space, and each member of the population continually adjust its position and velocity through learning its own experience and the experience of other members of the population. The particles are evolved according to the following formula [31, 32]:

$$v_{id} = v_{id} + c_1 r_1 (P_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \quad (1a)$$

$$x_{id} = x_{id} + v_{id} \quad (1b)$$

Where c_1 and c_2 are acceleration constants; r_1 and r_2 are two random numbers in the range $[0, 1]$. Equation (1a) is used to calculate a particle's new velocity: the first part represents the inertia of pervious velocity; the second part is the "cognition" part, memorizing the distances from its current position to its local best P_i , representing the private experience by itself; the third part is the "social" part, learning the best global experience P_g , representing the cooperation and information-sharing mechanism among the particles [31]. If the sum of accelerations causes the velocity v_{id} on that dimension to exceed v_{max} , then v_{id} is limited to v_{max} , and v_{max} determines the region between the present position and the target position searched.

As one of the latest evolutionary optimization methods, many common points as the former evolutionary are introduced to generate the next generation. PSO has its advantages such as simple concept, immediately accessible for practical applications, simple structure, ease of use, speed to get the solutions, and robustness, parallel direct search method. At the heart of the PSO method is the strategy that the algorithm updates a population of particles with the internal velocity and attempts to benefit from the discoveries of themselves and previous experiences of all other companions.

PSO, initialized by randomly generated population, search for optimum by updating from generation to

generation. In the process of iteration, a key step is evaluating the desired optimization objective function for possible solutions. Compare the evaluated objective value of each particle with its P_{id} . If current value is better than P_{id} , then set the current location as the P_{id} location. Furthermore, if current value is better than P_{gd} , then reset P_{gd} to the current index in particle array.

5. The proposed algorithm

Since 1995, Particle Swarm Optimization has been successfully applied in continuous problems, and scheduling problem is the new field of PSO algorithm in discrete space [12, 30]. If the PSO algorithm is designed to tackle discrete problem, a direct correlation must be found between the particle vector and the solution representation of scheduling problem. In following text, we redefine the position and velocity of particles as well as operations satisfying the precedence constraints between tasks.

A. Solution representation

Lending idea from GA solving combinatorial optimization problem, there are two kinds of solution representation: priority-based and permutation-based. For the permutation form, the position of a task in the vector-represented permutation means the order or sequence the task is scheduled. For the priority form, each location or position in the vector fixedly represents a task, and the corresponding value of each element means the priority a task is scheduled.

(1) Priority-based solution representation

Every dimension d ($d=1: D$) represents the task index number, and the corresponding value of each element in the vector X_i means the priority a node is scheduled to start, i.e. $X_i = (x_{i1}, \dots, x_{id}, \dots, x_{iD})$ that describe the D - dimension position of particle i can be used to represent the priority values of the D tasks.

It is assumed assigning a higher priority to a task with a smaller element value, i.e. each task was scheduled in ascending order of each element value. Figure 1 illustrates the priority-based solution representation of target individual with five dimensions: the 4th element is the smallest, so the task 4 is scheduled firstly, and so on.

Priorities vector	1.3	4.5	3.7	0.6	2.3
Task index	1	2	3	4	5
Scheduling list	4	1	5	3	2

Figure 1 : Priority-based solution representation

(2) Permutation-based representation

Every dimension d ($d=1:D$) in the vector permutation means the order or sequence the node is scheduled, and the corresponding value of each element means a node index number. So the permutation-based representation actually indicates the task scheduling list. Used to stand for the index number of a task, the parameter of each element in the particle represented permutation should be an integer limited to $[1, D]$. For example, a permutation of 5 tasks for a scheduling can be represented as the particle in Figure 2.

Permutation vector	3	5	1	4	2
Scheduling sequence	1	2	3	4	5
Scheduling list	3	5	1	4	2

Figure 2 : Permutation-based solution representation

Because of the stochastic search of particle swarm, the priority-based method may produce an invalid solution destroying the order constraints. Moreover, Zhang et al. demonstrated the permutation-based solution for the RCPSP (the Resource-Constrained Project Scheduling Problem), had better performance than priority-based representation [30]. The feasible permutation of tasks was also successfully applied in parallel GA by Kwok and Ahmad [8]. We adopt the solution vector with permutation-based definition in this research.

B. Encoding the PSO

Here, we combine the PSO with list scheduling to solving the task matching and scheduling. The main emphasis is placed on how to develop an alternative method for the multiprocessor scheduling by utilizing the features of PSO. The original operations in PSO do not fit well with our problem model, so we tend to modify the solution representation and operations.

Encoding position: In multiprocessor scheduling problem, we define the position vector of a particle represents a feasible solution, denoting a node list satisfying DAG graph topology order based on task permutation. That is to say, the position vector means the permutation of the task index and specifies the order of task to be executed. The vector elements must be encoded as integer limited in $[1, N]$, and N is the task size. Every integer in the scale appear only once in feasible solution.

Defining Swap Operator: Consider the multi-processor scheduling with n tasks, a normal solution S is a list of n nodes. If a new solution S' is obtained when exchanging node n_i and node n_j in solution S , the operation is defined as Swap Operator, denoting the Swap Operator and the process as $SO(n_i, n_j)$ and $S' = S + SO(n_i, n_j)$ [29]. For example,

$$\{n_1, n_2, n_3, n_4, n_5, n_6\} + SO(n_1, n_3) = \{n_3, n_2, n_1, n_4, n_5, n_6\} \quad (2)$$

When Swap Operators $SO_1, SO_2, SO_3, \dots, SO_n$ are imposed to a solution continuously, we define the action as a Set of Swap Operators (SSO) [26]. These process can be depicted as formula (4):

$$SSO = (SO_1, SO_2, SO_3, \dots, SO_n) \\ S' = S + SSO \quad (4)$$

Note that the order of Swap Operations acting is very important.

Defining the minus operation between position vectors: It is assumed that there are two vectors X and Y . When impose SSO to X, Y come into being, i.e. $Y = X + SSO$. So minus operation between two vectors is defined as follows:

$$X = Y + SSO \Leftrightarrow X - Y = SSO, \quad (5)$$

Where SSO represents a Set of Swap Operator, so the $(P_i - X_i(t))$ and $(P_g - X_i(t))$ are also defined as SSO acting on scheduling list.

Encoding velocity: A new position is updated when velocity V is imposed on old position. So the velocity can be defined as SSO acting on a scheduling list according to formula (1a)(5). In specific scheduling environment,

$$V = \{(n_i^k, n_j^k), i, j \in \{1, 2, \dots\}, k \in \{1, 2, \dots\}\} \quad (6)$$

Which represents that node n_i^1 and n_j^1 are swapped firstly, and n_i^2 and n_j^2 are swapped secondly, and so forth.

Defining the merging operation between two SSO s: As a velocity V , that is derived from formula (1a), then the new velocity is consisted of three SSO s: the old velocity, the $(P_i - X_i(t))$ and $(P_g - X_i(t))$. How can several SSO s be incorporated into a new SSO ? Suppose there is two sets of Swap Operators, SSO_1 and SSO_2 , when SSO_1 and SSO_2 act on one solution S in order, namely SSO_1 first, SSO_2 second, get a new solution S' , and there is another SSO' acting on the same solution S , then get the same solution S' . There we define the symbol " \cup " as merging two SSO s into a new SSO' [29]:

$$SSO' = SSO_1 \cup SSO_2 \quad (7)$$

The evolution equation of PSO for scheduling: Using the concept described above, we can get the following evolution equation in task scheduling problem.

$$V_i(t+1) = V_i(t) \cup c_1 r_1 (P_i - X_i(t)) \cup c_2 r_2 (P_g - X(t)) \quad (8a)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (8b)$$

In our algorithm, $c_1 = r_1 = c_2 = r_2 = 1$.

C. The initial population

Like other evolutionary algorithms, the PSO algorithm starts from an initial population. On the basis of above, the initial vectors, i.e. initial scheduling list, must satisfy the precedence constraints. The first position individual is obtained from the topological sorting algorithm. The other individuals are derived from the first through neighborhood exchange so that the topological orders generated can cover the whole space of the feasible solutions for the problem.

The initial velocity population is randomly generated by neighborhood exchange, and there has control over the exchange time. Because the exchange is random, a strategy is designed to ensure the feasible solution.

D. The mechanism to produce valid solution

The new individual is produced on the basis of implementing the SSO on old individual. Denoting the old individual and the final Set of Swap Operation as I_{old} and SSO :

$$I_{old} = \{n_1, n_2, \dots, n_i, \dots, n_m, \dots, n_j, \dots, n_D\} \quad (9)$$

$$SSO = \{(n_i^k, n_j^k), i, j \in \{1, 2, \dots, N\}, k \in \{1, 2, \dots, \}\} \quad (10)$$

When SSO acts on I_{old} , n_i^k, n_j^k in the old individual are exchanged in the k th swap action. To ensure the feasible solution, the task n_m between n_i^k, n_j^k is not the successor node of n_i^k , neither the predecessor node of n_j^k . Otherwise, the Swap Operation n_i^k, n_j^k is discarded.

E. Simulated annealing selection strategy

Once a new solution is generated, a decision must be made whether or not to accept the newly derived vector as next generation. The general strategy is the greedy technique, in which the solution having better performance is accepted. The algorithm using greedy criterion can converge fairly fast, but it runs the risk of becoming trapped by a local minimum. In typical PSO, all the new derived solutions are transferred into next generation without selection. This method can preserve the diversity and lead to slow convergence as well. To improve the global convergence, the mechanism of simulated annealing is introduced into PSO algorithm. The hybrid algorithm effectively integrates both the ability to jump out of the local minima in simulated annealing and the capacity of searching the global optimum in PSO algorithm.

In original simulated annealing, a system is attributed with a temperature T and state energy E . A random disturbance is implemented on old state and the ΔE in energy change is measured to determining if the disturbance

favorable or not [21]. The new state is unconditionally accepted if it lowers the current energy of the system. If the energy of the system is increased by the disturbance, the new state is accepted with some random probability. At every temperature the process is iterated sufficient times so that the system achieve the lowest energy equilibrium. Drop the system temperature and start next loop until a stop criterion is met, usually attaining a good objective value or a predefined maximum number of generations [36].

When the SA is employed in scheduling problem, by analogy with the system 'energy' in the original application, the objective function is involved. In this paper, the objective function considers the scheduling length. The algorithm not only accepts changes that decrease objective function, but also some changes that increase it. The latter are accepted with a probability [36]:

$$P = e^{-\Delta f/T} \quad (11)$$

Where Δf is the increase value of objective function.

F. Processor Selection

The salient characteristic in the proposed algorithm is to construct the scheduling list meanwhile avoid the mean level. Particle Swarm Optimization, in our proposed algorithm, is used to obtain candidate scheduling lists complying with the precedence constraints among task nodes. These scheduling lists are evaluated and updated according to the operation defined above until the terminate condition is met. Because of the global search capacity of PSO, the sub-optimal or near-optimal solution can be obtained in rational time.

An important factor in static list scheduling is the task allocation strategy that determines the "best" processor on which a ready task node is to be scheduled. The popular assigning rule is Earliest Start Time (EST) in homogeneous multiprocessors scheduling, emphasizing the fittest processor the processor enabling the earliest start time for the given task n_i , while in heterogeneous environment the best machine is the processor enabling the earliest finish time. The Heterogeneous Earliest Finish Time (HEFT) is considered one of the best algorithms for scheduling tasks onto heterogeneous processors [11]. In addition, the Heterogeneous Critical Parents with Fast Duplicator (HCPFD) proposed a fast machine assignment mechanism based on task duplication. The critical parent node is the parent node sending the latest data which decides the starting time of the given node. HCPFD algorithm selects a machine that minimizes the finish time of the selected task, and then duplicates the CP node to the same machine if the duplication condition is satisfied [2]. The experiments testified the method could improve the scheduling length to some degree. Except these strategies, because each task has different execution time, we also try the strategy to select target machine that has the Most Fast Execution Time (MFET) to the selected task.

G. PSO Algorithm for Task Scheduling Problem

Having determined some important factors of hybrid PSO algorithm for tasks scheduling problem, the proposed algorithm is described as Algorithm 1:

Algorithm1. Particle Swarm Optimization algorithm for Task Scheduling

Step1 Generate an initial population of scheduling lists by topology sort and neighborhood search, and set the initial parameter T ;
 Step2 Evaluate the scheduling length (Schedule each scheduling list), find out p_g ;
 Step3 Update the particle position (scheduling list) according to the evolution equation (8a, 8b) to get a new feasible solution;
 Step4 Schedule each scheduling list, if scheduling length of current p_i is better than p_i of the particle, update the p_i as do p_g ;
 Step5 Using SA technique to select the next generation;
 Step6 Termination criterion is met? No, step3; otherwise step 7;
 Step 7 Output the SL.

6. Experiment results

A. Application graphs

Firstly, a small size graph set from literatures is selected to test four assigning strategies and the performance of the hybrid PSO algorithm. At the same time, a graph generator is implemented to generate two suites of graphs with various characteristics such as CCR , heterogeneity factor, connectivity, number of tasks [4]. The communication-to-computation ratio (CCR) of a parallel program is defined as the ratio between its average communication and computation cost. The first suite consisted of simple jointree and fork-tree. The second is randomly generated graphs. In these graphs, different tasks size, CCR and processors size are discussed.

B. Performance evaluation

Afterwards, the average performances of different algorithms to these graphs from are compared. The main goal of the scheduling is achieve the minimal schedule length for parallel application. Account for graphs with different properties, we normalize the schedule length to the lower bound, which is so called Normalized Schedule Length (NSL) [15]:

$$NSL = \frac{SL}{\sum_{i \in CP} \min_{p_j \in P} \overline{w(i, j)}} \quad (12)$$

The denominator is the computation sum of node in critical path, where $\overline{w(i, j)}$ is the mean execution time.

C. The results

The number of the particles is set 20 and the number of the generation is 30. Four processor assigning rules are performed to the small size graphs from literature and the best solutions are listed in Table 1.

From Table 1, HCPFD technique is slight preference to HEFT and MFET is the worst without considering the interprocessor communication. EST strategy is better in homogeneous scheduling but isn't suitable in heterogeneous without incorporating the processing speed. So we combine PSO with HCPFD as a hybrid heterogeneous PSO scheduling algorithm.

Table 1. Schedule length generated by different assign rules for a small size set graphs in literature

Source of task graph	No of processors	No of task	PSO + HEFT	PSO + EST	PSO + HCPFD	PSO + MFET	Known solution
[10]	3	7	2069	2701	2018	2745	2018
[2]	4	10	118	120	94	116	94
[3]	3	9	143	146	140	260	138
[4]	3	5	32	55	32	33	-
[5]	3	8	14	14	14	32	14
[11]	3	10	76	94	73	104	80

At the same time, we compare the performance of the proposed hybrid PSO algorithm, HEFT and HCPFD under different number of tasks and processors for randomly generated DAG graphs. Graph parameters are set as follows:

CCR {0;1; 1;0; 10}
 Tasks size n {20; 30; 40; 60; 80; 100}
 Machines size m {2; 4; 6; 8; 10; 12}

There include 5 graphs for every parameter combination (CCR ; n , m) and compute the mean result after 10 runs.

Firstly, the results of the mean scheduling length for two special type graph including fork-tree and join-tree are listed in Table 2, and we assume the processor speed is inconsistent [4]. The fork and join are basic building blocks of many general task graphs. A conclusion can be derived from Table 2 that the proposed algorithm always has better performance than other two algorithms, and the highest improving ratio in scheduling length can arrive to 18.3%. The improving ratio is defined as a ratio of the best result of the proposed algorithm to the result of other algorithms.

Table 2 : Schedule lengths of random join-tree and fork-tree

Graph structure	No of tasks	No of processors	HEFT	HCPFD	PSO + HCPFD	Improving ratio(%)
Join-tree	50	4	550	526	462	12.6
	50	8	282	252	232	17.7
	80	4	827	801	786	4.9
	80	8	558	502	416	18.3
Fork-tree	50	4	562	547	465	13.7
	50	8	307	310	285	7.2
	80	4	807	830	720	10.8
	80	8	431	431	405	6.1

For random architecture system, the effects of different tasks size and machines size are investigated and the results are depicted in Figure 3 and Figure 4. The hybrid PSO algorithm has better performance whether different tasks size or different machines size. The reason behind the results can be analyzed. The three algorithms are based on list scheduling, but the method producing scheduling list and processor assigning rule are different. The HEFT use the mean *b-level* of task graph while HCPFD use the Average Latest Starting Time (ALST) as the node priorities and the priorities preserve unchanged during scheduling [2,11]. The HEFT assign the target machine based on insertion mechanism and the others duplicate the critical parent. The hybrid PSO scheduling algorithm first construct a better scheduling list through the guided particles' search technique and the cooperation among particles, then duplicate the critical parent to improve the performance further.

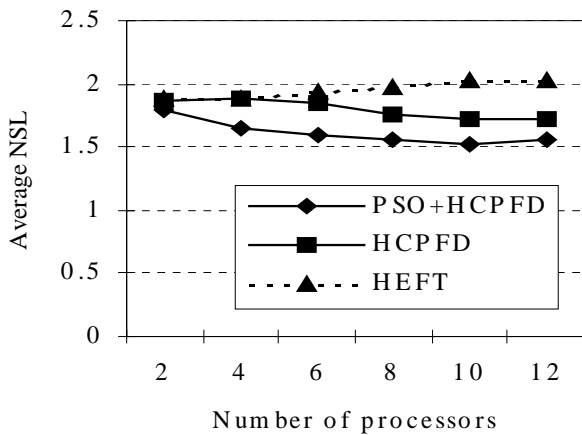


Figure 3 : Average NSL of random generated graphs with different machines size (n=100)

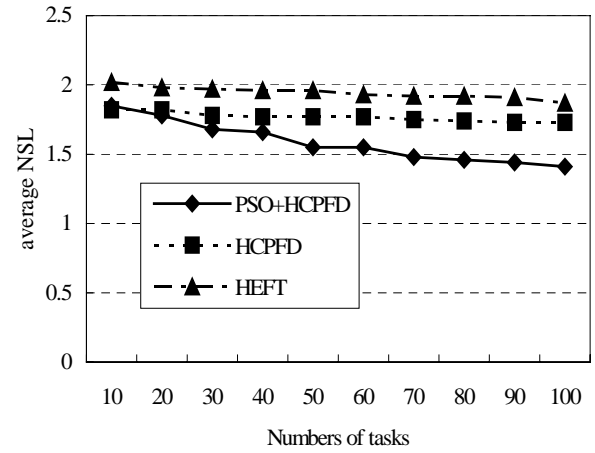


Figure 4 : Average NSL of random generated graphs with different nodes size (p=4)

Finally, SA technique is utilized to improve the solution quality. When the new solution is better than the parent, it is into the next generation to substitute for the parent. When the new solution is inferior to the parent, the solution is accepted according to the probability of SA strategy. We implement the SA in hybrid PSO algorithm and compare the results. The result is depicted in Figure 5. The improving ratio is defined as the decreasing ratio of scheduling length with SA select strategy to without strategy under the identical scheduling problem.

From the Figure 5, it is evident that a hybrid PSO strategy incorporating temperature-based acceptance criterion of SA is preferable to monotonically selecting manipulation. The using of probability to discard inferior solutions in earlier stage decreases the algorithm time while acceptance to bad solution in later period enhance the solution diversity, and escapes the local optima. With the increasing of tasks size, the improving ratio decreases because the large-scale problem requires more time to search solution.

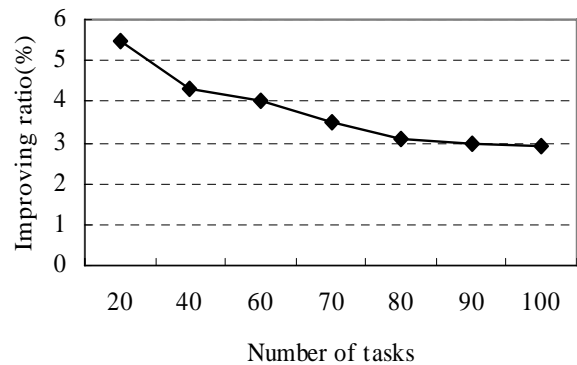


Figure 5 : Average improving ratio of algorithm with and without SA selection strategy to different tasks

7. Conclusion

This research presents an algorithm that schedules parallel application onto heterogeneous multiprocessor system. Particle swarm optimization (PSO) is based to develop a solution-solving scheme in heterogeneous multiprocessors text. In practice, we re-encode the particle variables to the scheduling environment in order to utilizing the search capability of PSO. The major motivation of using PSO is theoretically find out an optimal scheduling list rapidly. Further, a task duplication technique is adopted in the task allocation procedure to reduce communication cost. Finally, SA algorithm is utilized as select strategy to improve the convergence speed and algorithm performance. As can be see from the previous results of the performance-testing experiment, in most cases the hybrid PSO method can find better schedule.

8. References

- [1] M. R. Garey, D. S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness." W.H. Freeman and Co., 1979.
- [2] T. Hagra, Jan Janecek. "A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems." 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 1, 2004.
- [3] Y. K. Kwok, I. Ahmad. "Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors." *Cluster Computing*, (2000), 3(2): 113-124.
- [4] T. D. Braun, H. J. Siegel, N. Beck. "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems." *Journal of Parallel and Distributed Computing*, 2001, 61(6):810-837.
- [5] C. W. Chiang, C. N. Lee. "A Study in Allocating Task Graphs onto A Heterogeneous Cluster-computing System." *Journal of in Da-Ye University*, 2002, 11(1):103-110.
- [6] L. Wang, H. J. Siegel, V. P. Roychowdhury, et al. "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach." *Journal of Parallel and Distributed Computing*, 1997, 47(1):8-22
- [7] H. Singh, A. Youssef. "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *Proceedings of the Heterogeneous Computing Workshop*, 1996, pp.86-97.
- [8] Y. K. Kwok, I. Ahmad. "Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm," *Journal of Parallel and Distributed Computing*. November 1997, 47(1):58-77.
- [9] P. Shroff, D. W. Watson, N. S. Flann, R. F. Freund. "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments." *Proceedings of the Heterogeneous Computing Workshop*, 1996, pp.98-104.
- [10] M. K. Dhodhi, I. Ahmad, A. K. Al-Yatama, Ishfaq Ahmad. "An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing Systems." *Journal of Parallel and Distributed Computing*, 2002, 62(9): 1338-1361.
- [11] H. Topcuoglu, S. Hariri, M. Y. Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing." *IEEE Transaction on Parallel Distributed System*, 2002, 13 (3):260-274.
- [12] J. Sun, W. B. Xu, B. Feng. "Scheduling Tasks onto Multiprocessors Using Particle Swarm Optimization." *DCABES 2005 Proceedings*, 2005, pp.109-113.
- [13] G. Sih, E. Lee. "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures." *IEEE Transaction on Parallel Distributed System*, 1993, 4:75-87.
- [14] I. Ahmad, Y. Kwork. "A comparison of task-duplication-based algorithms for scheduling parallel programs to message-passing systems." *Proc. 11th Int. Symp. Of High-Performance Computing Systems*, 1997, pp. 39-50.
- [15] Y. K. Kwok, I. Ahmad. "Benchmarking and Comparison of the Task Graph Scheduling Algorithms." *Journal of Parallel and Distributed Computing*, vol. 59, no. 3, pp. 381-422, December 1999.
- [16] W. B. Xu, J. Sun. "Efficient Scheduling of Task Graphs to Multiprocessors Using a Simulated Annealing Algorithm." *DCABES 2004 Proceedings*, vol 1:435-439.
- [17] A .Gerasohlis, T. Yang. "A comparison of clustering heuristics for scheduling DAGs on multiprocessors." *Parallel and Distributed Computing*, 1992, 16(4): 276-291.
- [18] O. Sinnen, L A. Sousa. "List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures." *Parallel Computing*, 2004, 30(1):81-100.
- [19] H. Topcuoglu, S. Hariri, M.Y. Wu. "Task scheduling algorithms for heterogeneous processors." *Heterogeneous Computing Workshop*, 1999. (HCW '99) *Proceedings*. Eighth. pp: 3-14.
- [20] S. Ali, H. J. Siegel. M. Maheswaran, D. Hensgen. "Representing task and machine heterogeneities for heterogeneous computing systems." *Tamkang*

- Journal. of Science and Engineering. 2000, 3(3): 195–207
- [21] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller. "Equations of state calculations by fast computing machines." *J. Chem. Phys.*, 1953, 21:1087–1091.
- [22] F. Glover. "New approaches for heuristic search, a bilateral linkage with artificial intelligence." *European Journal of Operational Research*, 1989, 39(1):119-130.
- [23] S. C. Porto, J. P. Kitajima, C. C. Ribeiro. "performance evaluation of a parallel tabu search task scheduling algorithm." *Parallel Computing*, 2000, 26:73--90.
- [24] W. J. Xia, Z. H. Wu, G. K. Yang. "A new hybrid optimization algorithm for the job-shop scheduling problem." *Proceedings of the American Control Conference*, v 6, *Proceedings of the 2004 American Control Conference (AAC)*, 2004, pp. 5552-5557.
- [25] M. F. Tasgetiren, Y. C. Liang, M. Sevkli, G. Gencyilmaz. "Particle Swarm Optimization Algorithm for Makespan and Maximum Lateness Minimization in Permutation." *4th International Symposium on Intelligent Manufacturing Systems, IMS2004*, 2004, pp.431-441.
- [26] M. F. Tasgetiren, M. Sevkli, Y. C. Liang, G. Gencyilmaz. "Particle Swarm Optimization Algorithm for Permutation Flowshop Sequencing Problem." *4th International Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS2004*, Vol. LNCS , No. 3172, 2004, pp. 382-390.
- [27] J. Bean. "Genetic algorithms and random keys for sequencing and optimization." *ORSA Journal on Computing*, 1994, 6(2): 154-160.
- [28] D. Merkle, M. Middendorf, H. Schmeck. "Ant colony optimization for resource-constrained project scheduling." *IEEE Transaction on Evolutionary Computation*, 2002, 6(4):333 339.
- [29] K. P. Wang, L. Huang, C.G. Zhou, W. Pang. "Particle swarm optimization for traveling salesman problem." *Proceedings of the Second International Conference on Machine Learning and Cybernetics*. November 2003,5:1583-1585.
- [30] H. Zhang, X. D. Li, H. Li. "Particle Swarm Optimization-Based Schemes for Resource-Constrained Project Scheduling." *Automation in Construction*, 2005, 14: 393-404.
- [31] J. Kennedy, R. C. Eberhart. "Particle Swarm Optimization." *Proc. IEEE Int. Conf. Neural Networks*. Piscataway, NJ (1995) pp.1942-1948.
- [32] Y. Shi , R. Eberhart. "Empirical study of particle swarm optimization." *Proceedings of IEEE Congress on Evolutionary Computation*, 1999, pp.1945-1950.
- [33] R. C. Eberhart, Y. Shi. "Comparison between Genetic Algorithm and Particle Swarm Optimization." *Evolutionary Programming VII (1998)*, *Lecture Notes in Computer Science 1447*, pp 611-616.
- [34] F. van den Bergh, A. P. Engelbrecht. "Cooperative Learning in Neural Networks using Particle Swarm Optimizers." *South African Computer Journal*, 2000, (26):84 90.
- [35] S. C. Esquivel, C. A. Coello. "On the use of particle swarm optimization with multimodal functions." *Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003)*, Canbella, Australia. pp. 1130-1136. \
- [36] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. "Optimization by Simulated Annealing." *Science*, 1983, 220: 671-680.

Author Biographies

Xiaohong Kong received the B.Sc. degree in Electrical Engineering from Zhengzhou Textile Institute, Zhengzhou, China. She is currently a candidate for the Ph.D. degree in the School of Informatics, Southern Yangtze University, Jiangsu, China.

Jun Sun received the M.Sc. degree in computation from the University of Southern Yangtze, where he is an associate professor. He is currently a candidate for the Ph.D. degree in the School of Informatics, University of Southern Yangtze.

Wenbo Xu is a tutor in the School of Informatics, University of Southern Yangtze, where he is a professor majoring in High Performance Computing.