Asian Journal of Information Technology 15 (7): 1263-1274, 2016
ISSN: 1682-3915
© Medwell Journals, 2016

Parallel Particle Swarm Optimization for Dynamic Task Scheduling Problem in a Multiprocessor Architecture

K. Deeba
Department of Computer Science and Engineering,
Hindusthan College of Engineering and Technology, Tamil Nadu, India

Abstract: This research focuses on Particle Swarm Optimization (PSO) and its variant approaches for dynamic task scheduling problem. Scheduling in a multiprocessor architecture has increased in the past decades due to the changing markets characterized by global competition and rapid development of new processes and technologies. The concepts of PSO and its variants are successfully tested with dynamic tasks with load balancing and without load balancing in a multiprocessor architecture, to reduce the makespan of the entire schedule. The introduction of the bad experience component in the velocity equation called worst particles have proven to be a significant improvement in the results when applied to the problem of multiprocessor task scheduling. Further, the concept of proposed IPSO is hybridized with Ant Colony Optimization (ACO) to achieve better schedule for task scheduling problem in a multiprocessor architecture. To speed up the convergence, parallel IPSO approaches such as Parallel Synchronous Improved Particle Swarm Optimization (PSIPSO) and Parallel Asynchronous Improved Particle Swarm Optimization (PAIPSO) are proposed. Thus, the results reveal that, the proposed parallel approach PAIPSO yields better results for dynamic task scheduling problem.

Key words: PSO, Improved PSO (IPSO), Ant Colony Optimization (ACO), Parallel Synchronous Improved PSO (PSIPSO) and Parallel Asynchronous Improved PSO (PAPSO)

INTRODUCTION

Global optimization problems are important in the fields of science, manufacturing industries, engineering and business. The goal of optimization is to find the best value for each variable in order to achieve satisfactory performance. Optimization is an active and fast growing area of research and has a great impact on the real world. Most of the NP-hard (Non-Deterministic Polynomial-Time Hard) problems cannot be solved using exact methods (Salman *et al*., 2002; Garey and Johnson, 1979). Hence, many heuristic and metaheuristics methodologies are being used to find near optimal solutions for the optimization problems. The objective in such problems is to find the optimal of all possible solutions that minimize or maximize an objective function. A stochastic high quality approximation of a global optimum is more valuable than a deterministic poor quality local minimum provided by a classical method (Mitten, 1970; Adam *et al*., 1974).

Some complex multidimensional problems cannot be solved using classical optimization techniques. This insight has led to an increased interest in a special class of searching algorithms, namely heuristic algorithms (Lee *et al*., 1988; Selvakumar and Murthy, 1994). These algorithms find approximate solutions and suggest some approximations to the solution of optimization problems with low time complexity. There can be a single or multiple objective functions that evaluate the quality of the generated solution. In recent decades, evolutionary and stochastic algorithms are hybridized with other evolutionary, intelligent and metaheuristic algorithms to solve extremely challenging problems such as single and multi-objective functions (Lee and Lee, 1998).

Scheduling, in general, is concerned with allocation of limited resources to certain tasks to optimize few performance criterion, like the completion time, waiting time or cost of production. Job scheduling problem is a popular problem in scheduling area of this kind. The importance of scheduling has increased in recent years due to the extravagant development of new process and technologies. Scheduling, in multiprocessor architecture can be defined as assigning the tasks of precedence constrained task graph onto a set of processors and determine the sequence of execution of the tasks at each processor (Wu and Gajski, 1990). A major factor in the efficient utilization of multiprocessor systems is the proper assignment and scheduling of computational tasks among the processors. This multiprocessor scheduling problem is known to be Non-deterministic Polynomial (NP) complete except in few cases have been proposed earlier to solve this kind of problem (Deeba and Thanushkodi, 2009).

Eberhart and Kenedy (1995) and Eberhart and Shi (1998) proposed a new designed crossover and mutation operators based on the characteristic of the job shop problem itself are specifically designed. Based on these, an improved genetic algorithm is proposed by Allahverdi *et al.* (2008) and Lee and Lee (1996) proposed a new course grain genetic algorithm in which the initial population is divided into multi sub population toward for reduce solution search speed and to prevent early convergence by migration between subpopulations.

Activity network of heterogeneous systems represented by Directed Acyclic Graphs (DAG) (Ahmad and Kwok, 1999; Casavant and Kuhl, 1988) proposed an effective memetic algorithm to solve Hybrid Flow Shop Scheduling with Multiprocessor Task (HFSMT) problem (Alba And Troya, 2001). Higginson *et al.* (2005) analyzed the importance of simulated parallel annealing within a neighbourhood for optimization of bio-mechanical systems. Schutte *et al.* (2004) discussed the paralleization of an increasingly popular global search method, the Particle Swarm Optimization (PSO) algorithm in detail to obtain enhanced throughput and global search capacity. In the next sections discusses about the job scheduling in multiprocessor architecture, Optimization algorithms such as GA, PSO, Proposed Improved PSO (IPSO), Hybrid Algorithms IPSO with ACO and Parallel IPSO algorithms for Dynamic job scheduling in multiprocessor architecture.

MATERIALS AND METHODS

Job scheduling in multiprocessor architecture: Job scheduling, considered in this study, is an optimization problem in operating system in which the ideal jobs are assigned to resources at particular times which minimizes the total length of the schedule. Also, multiprocessing is the use of two or more central processing units within a single computer system. This also refers to the ability of the system to support more than one processor or the ability to allocate tasks between them. In multiprocessor scheduling, each request is a job or process. A job scheduling policy uses the information associated with requests to decide which request should be serviced next. All requests waiting to be serviced are kept in a list of pending requests. Whenever, scheduling is to be performed, the scheduler examines the pending requests and selects one for servicing. This request is handled over to server. A request leaves the server when it completes or when it is preempted by the scheduler, in which case it is put back into the list of pending requests. In either situation, scheduler performs

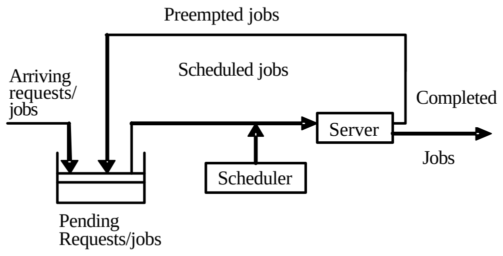

Fig. 1: A schematic of job scheduling

scheduling to select the next request to be serviced. The scheduler records the information concerning each job in its data structure and maintains it all through the life of the request in the system. The schematic of job scheduling in a multiprocessor architecture is shown in Fig. 1.

Dynamic task scheduling: In this research, the dynamic task scheduling is examined with the following scenario. The processors in the system are heterogeneous and they are capacitated with different units of memory and processing resources. Hence a tasks execution cost varies it it is executed on different processors. And assumed all communication links are identical. If communication need exists between two tasks executed on different processors then there exists a communication cost. For the execution task need some resources consume from its execution processor. Load balancing of dynamic tasks is particularly useful in a system, where processor utilization is the major issue instead of minimizing execution time of the applications. The tasks are scheduled as and when they arrive in the queue.

The objective is to minimize the average total execution cost encountered by the task assignment, of all tasks allocated to the processors. i.e., minimisation of makespan of the entire schedule. A particle is evaluated by calculating its fitness function. Fitness function indicates the goodness of the schedule. Dynamic task scheduling can be done under two cases namely:

- Considering only the arrival time of the tasks (without load balancing)
- Considering both the time of arrival of each task and efficient processor utilization. (with load balancing)

Fitness function to minimise the schedule length varies for the above said two cases, hence dealt individually in the following subsections.

Dynamic task scheduling without load balancing: Dynamic task scheduling is adopted to schedule the tasks in such a way that how they are arriving. Dynamic load balancing does not require the prior knowledge about the tasks as like static, i.e., it need not be aware of the run-time behaviour of the applications before execution. In dynamic scheduling, tasks are redistributed among processors during the execution time. This redistribution is done by transferring tasks from the heavily loaded processors to the lightly loaded processors (called load balancing), to improve the processor utilization and performance. To minimise the makespan of the schedule, the equation is formulated and represented as Eq. 1 and 2:

$$\text{TFT}\left(P_j\right) = \text{arrival}_i + \sum_{i=1}^{n} \text{fin}_i \quad (1)$$

$$\text{fin}_i = \begin{Bmatrix} \text{exe}_i & \text{if arrival}_i \text{ £ fin}_{i-1} \\ \text{arrival}_i + \text{exe}_i & \text{otherwise} \end{Bmatrix} \quad (2)$$

Fitness function is formulated to minimise the average of total finishing time Eq. 3:

$$F(X) = \min\left\{\frac{\sum_{j=1}^{m} \text{TFT}\left(P_j\right)}{m}\right\} \quad (3)$$

Where:

'n'	=	The number of tasks
'm'	=	The number of processors
TFT(P_j)	=	Total Finishing Time of Processor (j)
arrival$_i$	=	Arrival time of task (i)
fin$_i$	=	Finishing time of task (i)
exe$_i$	=	Execution time of task (i)

Dynamic task scheduling with load balancing: The drawback in the previous study, i.e., in dynamic scheduling without load balancing, some of the processors may be lightly loaded and some are heavily loaded. To overcome the problem and also to improve the processor utilization and performance, load balancing is considered, which gives a significant improvement in the processors utilization. To achieve load balancing in a dynamic task scheduling, tasks has to be allocated in such a way that they are arriving, i.e., makespan of the schedule has to be minimized and processor utilization has to be increased. To achieve better load balancing, first the tasks has to be assigned to the processors in such a way to reduce the makespan and to increase the utilization. Hence, equations are formulated and are represented as follows Eq. 4-6:

$$\text{TAQ}\left(P_j\right) = \text{arrival}\left(t_1\right) + \left(\frac{1}{\text{maxspan}}\right) \times \text{Avg Utilization} \quad (4)$$

$$\text{Utilization}\left(P_j\right) = \frac{\text{CT}\left(P_j\right)}{\text{maxspan}} \quad (5)$$

$$\text{Avg Utilization} = \frac{\sum_{j=1}^{m} \text{Utilization}\left(\left(P_j\right)\right)}{m} \quad (6)$$

Where:

TAQ(P_j)	=	The used to evaluate the Quality of task assignment in a processor
arrival(t_1)	=	The arrival time of task 1
max span	=	The total finishing time of the schedule
CT(P_j)	=	The completion time of processor j
Avg Utilization n	=	The sum of all processors utilization divided by the total number of processor
m	=	The number the processor

Effective utilization of processors supports the concept of load balancing. If all the processors are used to their maximum, then the loads, which are the measures of idleness of processors are effectively reduced. Hence, fitness function calculates the average of the total execution time of the set of tasks allocated to the processors as Eq. 7:

$$F(X) = \max\left\{\frac{\sum_{j=1}^{m} \text{TAQ}\left(P_j\right)}{m}\right\} \quad (7)$$

Optimization techniques: Several heuristic traditional Algorithms were used for solving the job scheduling in a multiprocessor architecture, which includes Genetic Algorithm (GA), Particle Swarm Optimization (PSO) algorithm, Simulated Annealing, Artificial Immune System (AIS) and Ant Colony Optimization. In this study a new Parallel Improved PSO (PIPSO) is suggested for the job scheduling problem in dynamic environment.

Particle swarm optimization for scheduling: The Particle Swarm Optimization (PSO) technique appeared as a promising algorithm for handling the optimization problems. The PSO is a population-based stochastic optimization technique, inspired by social behavior of bird flocking or fish schooling. The PSO is inspired by the ability of flocks of birds, schools of fish and herds of animals to adapt to their environment, find rich sources of food and avoid predators by implementing an information sharing approach. PSO technique was invented in the mid 1990s while attempting to simulate the choreographed, graceful motion of swarms of birds as part of a socio cognitive study investigating the notion of collective intelligence in biological populations.

PSO procedures based on the above concept can be described as follows. Namely, bird flocking optimizes a certain objective function. Each agent knows its best value so far (pbest) and its XY position. Moreover, each agent knows the best value in the group (gbest) among pbests. Each agent tries to modify its position using the current velocity and the distance from the pbest and gbest. Based on the above discussion, the mathematical model for PSO is as follows, Velocity update equation is given by Eq. 8:

$$V_i = w \times V_i + C_1 \times r_1 \times (P_{best_i} - S_i) + C_2 \times r_2 \times (g_{best_i} - S_i) \tag{8}$$

Using Eq. 4, a certain velocity that gradually gets close to pbests and gbest can be calculated. The current position (searching point in the solution space) can be modified by the following Eq. 9:

$$S_{i+1} = S_i + V_i \tag{9}$$

Where:

V_i	=	Velocity of particle I
S_i	=	Current position of the particle
w	=	Inertia weight
C_1	=	Cognition acceleration coefficient
C_2	=	Social acceleration coefficient
P_{besti}	=	Own best position of particle i
g_{besti}	=	Global best position among the group of particles
r_1, r_2	=	The uniformly distributed random numbers in the range [0-1]
s_i	=	The current position
s_{i+1}	=	The modified position
v_I	=	The current velocity
v_{i+1}	=	The modified velocity
v_{pbest}	=	The velocity based on pbest
v_{gbest}	=	The velocity based on gbest

Figure 2 shows the searching point modification of the particles in PSO. The position of each agent is represented by XY-axis position and the velocity (displacement vector) is expressed by vx (the velocity of X-axis) and vy (the velocity of Y-axis). Particle are change their searching point from S_i-S_{I+1} by adding their updated velocity V_i with current position S_i.

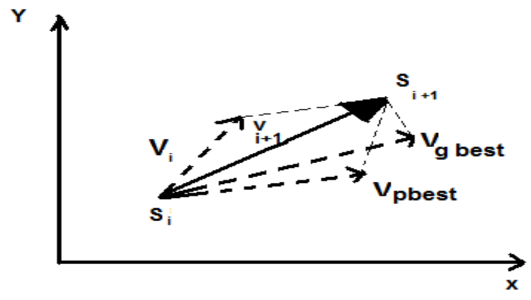

Fig. 2: Flow diagram of PSO

Each particle tries to modify its current position and velocity according to the distance between its current position S_i and V_{pbest} and the distance between its current position S_i and V_{gbest}.

Improved Particle Swarm Optimization (IPSO): Particle Swarm Optimization (PSO) is a popular swarm based optimization technique that mimics the bird's flocking behavior. Even though PSO shows better performance, premature convergence and local optima are the two major problems faced by the PSO approach for scheduling problem.

To overcome this difficulty, an Improved Particle Swarm Optimization (IPSO) is proposed in which a split up is made in the cognitive behavior. That is the particle is made to remember its worst position also. In this new proposed Improved PSO (IPSO) having better optimization result compare to general PSO by splitting the cognitive component of the general PSO into two different component. The first component can be called good experience component. This means the bird has a memory about its previously visited best position. This is similar to the general PSO method. The second component is given the name by bad experience component. The bad experience component helps the particle to remember its previously visited worst position. To calculate the new velocity, the bad experience of the particle also taken into consideration. On including the characteristics of P_{best} and P_{worst} in the velocity updation process along with the difference between the present best particle and current particle respectively, the convergence towards the solution is found to be faster and an optimal solution is reached in comparison with conventional PSO approaches. This infers that including the good experience and bad experience component in the velocity updation also reduces the time taken for convergence. This modification helps in exploring the search space very effectively to identify the promising solution region. The introduction of the worst particle, i.e.,

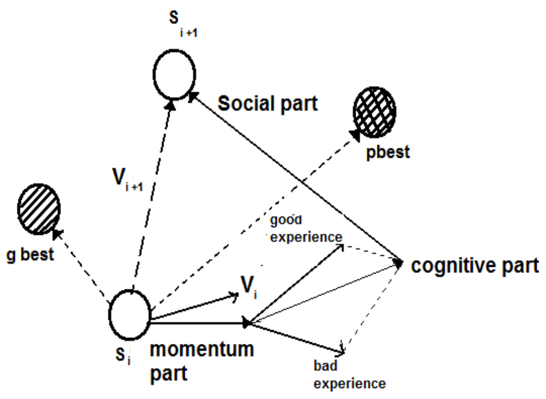

Fig. 3: Concept of iproved prticle sarm otimization search point

the bad experience component in the cognitive component yields a significant improvement in the results when applied to the task scheduling problem. The including of the worst particle plays a major role in the achievement of a better solution than with only using with good experience component called best particle in the velocity equation. The new velocity update Eq. 10 is given by:

$$V_i = w \times V_i + C_{1g} \times r_1 \times \left(P_{best_i} - S_i\right) \times P_{best_i} + C_{1b} \times r_2 \times \left(S_i - P_{worst_i}\right) \times P_{worst_i} + C_2 \times r_3 \times \left(G_{best_i} - S_i\right) \quad (10)$$

The position update Eq. 11 is given by:

$$S_{i+1} == S_i + V_i \quad (11)$$

Where:

C_{1g}	=	Self-confidence factor which accelerate the particle towards its best position
C_{1b}	=	Self-confidence factor which accelerate the particle away from its worst position
C_2	=	Swarm confidence factor (ranges from 2-2.5)
$P_{worst\ i}$	=	Worst position of the particle (i)
$r_{1,2,3}$	=	The uniformly distributed random numbers in the range [0-1]
S_i	=	The current position
S_{i+1}	=	The modified position
V_i	=	The current velocity
V_{i+1}	=	The modified velocity

The positions are updated using Eq. 11. The inclusion of the worst experience component in the behaviour of the particle gives the additional exploration capacity to the swarm. By using the bad experience component; the particle can bypass its previous worst position and try to occupy the better position. Figure 3 shows the concept of IPSO searching points.

The proposed improved PSO algorithm:

- Step1: Select the number of particles, generations, tuning accelerating coefficients C_{1g}, C_{1b} and C_2 and random numbers r_1, r_2, r_3 to start the optimal solution searching
- Step2: Initialize the particle position and velocity
- Step3: Select particles individual best value for each generation
- Step 4: Select the particles global best value, i.e. particle near to the target among all the particles is obtained by comparing all the individual best values
- Step 5: Select the particles individual worst value, i.e. particle too away from the target
- Step 6: Update particle individual best (P_{best}), global best (G_{best}), particle worst (P_{worst}) in the velocity equation (2.1) and obtain the new velocity
- Step 7: Update new velocity value in Eq. 11 and obtain the new position of the particle
- Step 8: Find the optimal solution with minimum 'F' by the updated new velocity and position

The flowchart for the proposed model formulation scheme is shown in Fig. 4.

Proposed parallel improved particle swarm optimization: To obtain an improved computational throughout and global search capability, the parallelization of an increasingly popular global search method is exploited, namely, the Parallel Particle Swarm Optimization (PPSO) algorithm. In view of improving the efficiency and performance in dynamic environment, more effort is still required. Hence, the present chapter aims at developing a new Parallel approach of Improved Particle Swarm Optimization to solve the multiprocessor scheduling problem. Parallel synchronous Improved Particle Swarm Optimization (PSIPSO) and Parallel Asynchronous Improved Particle Swarm Optimization (PAIPSO) methodologies are tested for the multiprocessor scheduling problem. The performances of the parallel Improved PSO (PSIPSO and PAIPSO) approaches are better than that of the Parallel Synchronous Particle Swarm Optimization (PSPSO) and Parallel Asynchronous Particle Swarm Optimization (PAPSO) when applied to the multiprocessor task scheduling problem (Fig. 4).

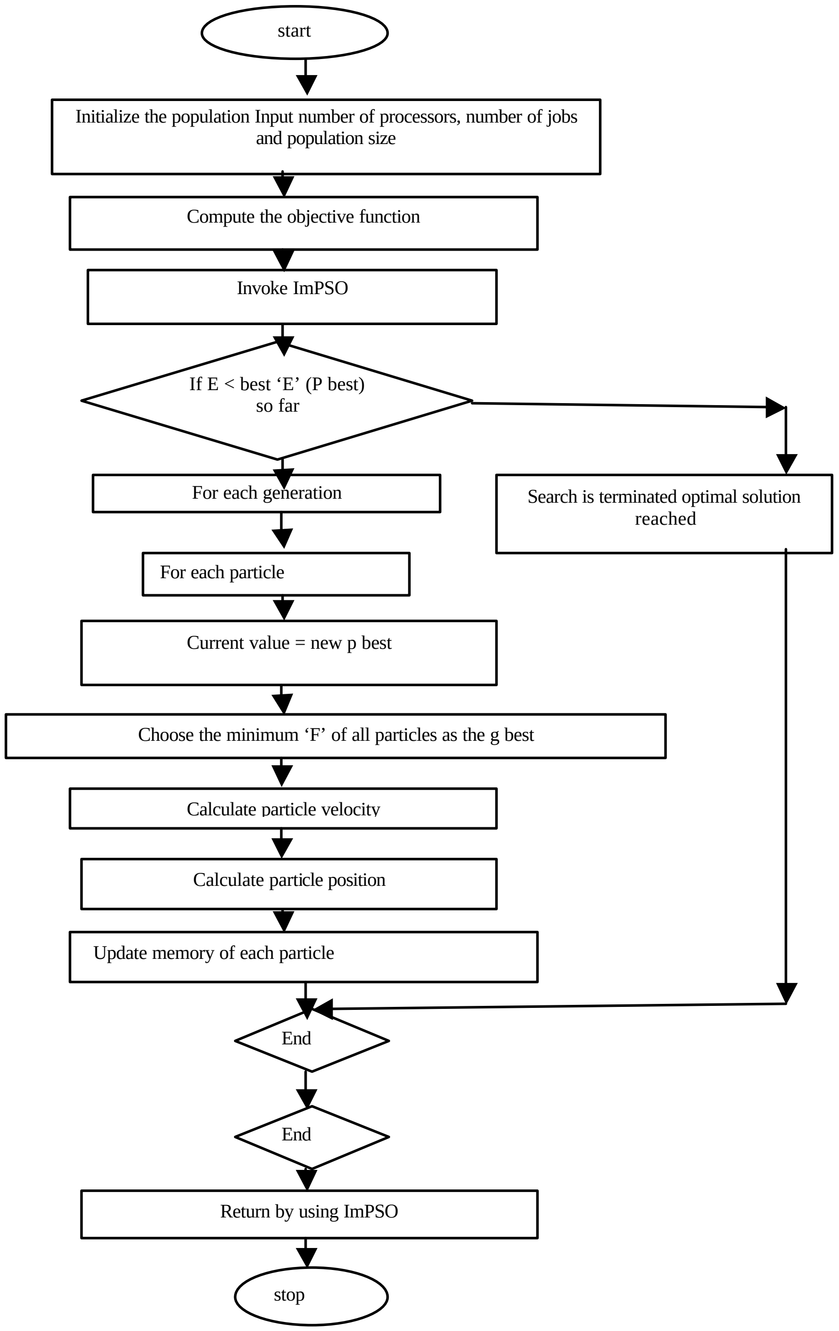
start
Initialize the population Input number of processors, number of jobs and population size
Compute the objective function
Invoke ImPSO
If E < best 'E' (P best) so far
For each generation
Search is terminated optimal solution reached
For each particle
Current value = new p best
Choose the minimum 'F' of all particles as the g best
Calculate particle velocity
Calculate particle position
Update memory of each particle
End
End
Return by using ImPSO
stop

Fig. 4: Flowchart for the proposed improved particle swarm optimization

There are different types of parallelism namely Bit-level, instruction level, data parallelism and task parallelism. This research work uses data parallelism. Parallel computing has advantages such as time consumption and to solve complex problems at faster rate. In analogy to parallel GAs, the Parallel Particle Swarm Optimizers (PPSO) is classified into three categories namely global PSO, migration PSO and Diffusion PSO.

Proposed parallel approach: Advancement and the recent trends in the computer and network technologies have direct towards the developments of parallel optimization algorithms. Synchronous parallel PSO method is proposed first for the task scheduling problem, which requires a synchronization point at the end of each optimizer iteration before continuing to the next iteration. The heuristic algorithms such as simulated annealing, pattern matching genetic algorithm and particle swarm optimization methods wait for completion of all function evaluations by the population before determining a new set of design variables. In the same way, parallel gradient based algorithms determining a new search direction after gradients calculated for all design variables. Particle Swarm optimization algorithm is well suited for a coarse grained parallel implementation on a parallel or distributed computing network. This study uses Master-Slave model. The parallelism is carried out a master-slave approach.

Proposed parallel Synchronous Improved PSO (PSIPSO) algorithm: The procedure for parallel synchronous PSO is as follows (Fig. 5):

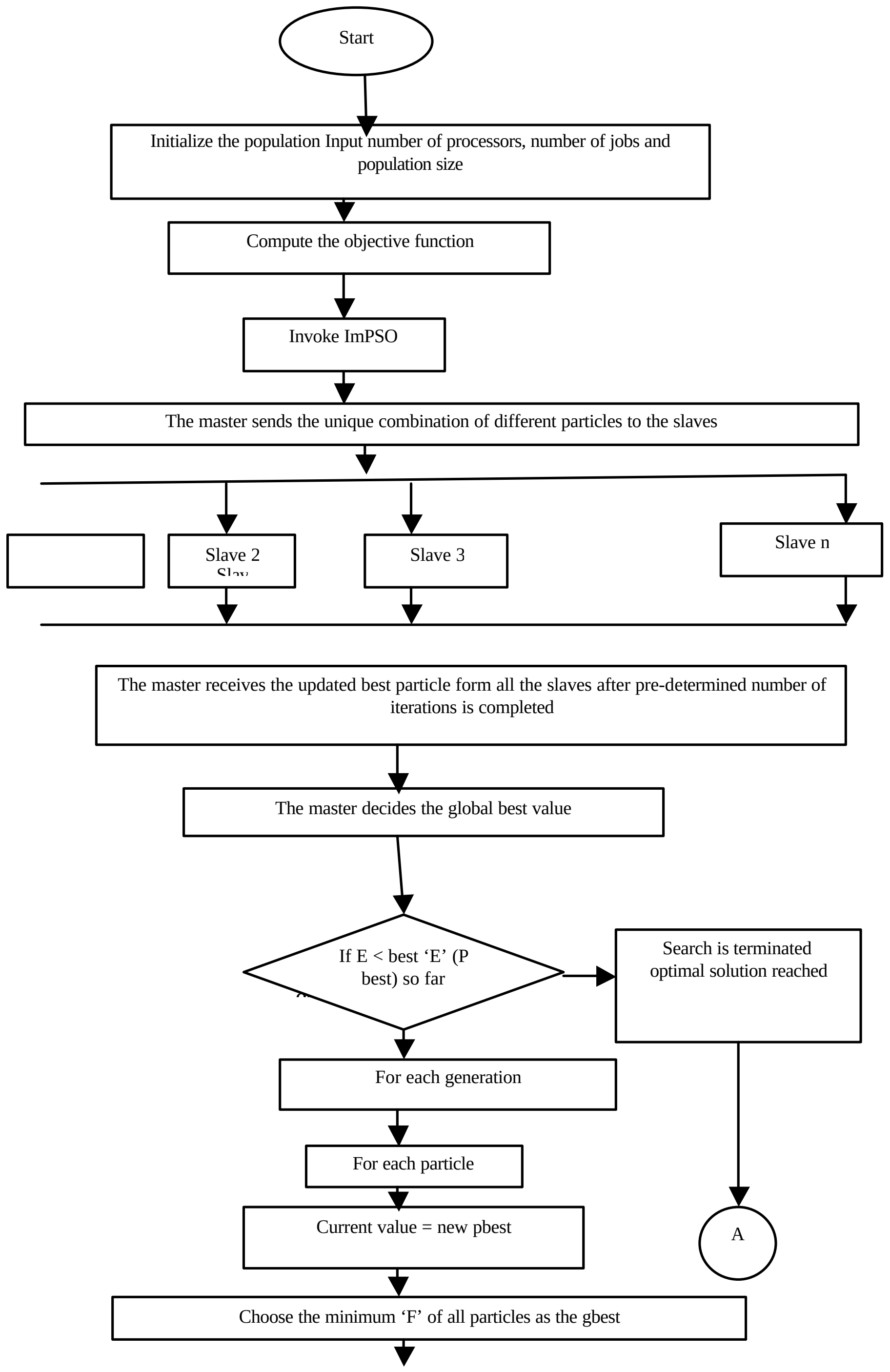
Start
Initialize the population Input number of processors, number of jobs and population size
Compute the objective function
Invoke ImPSO
The master sends the unique combination of different particles to the slaves
Slave 2
Slave 3
Slave n
The master receives the updated best particle form all the slaves after pre-determined number of iterations is completed
The master decides the global best value
If E < best 'E' (P best) so far
Search is terminated optimal solution reached
For each generation
For each particle
Current value = new pbest
A
Choose the minimum 'F' of all particles as the gbest

Fig. 5: Continue

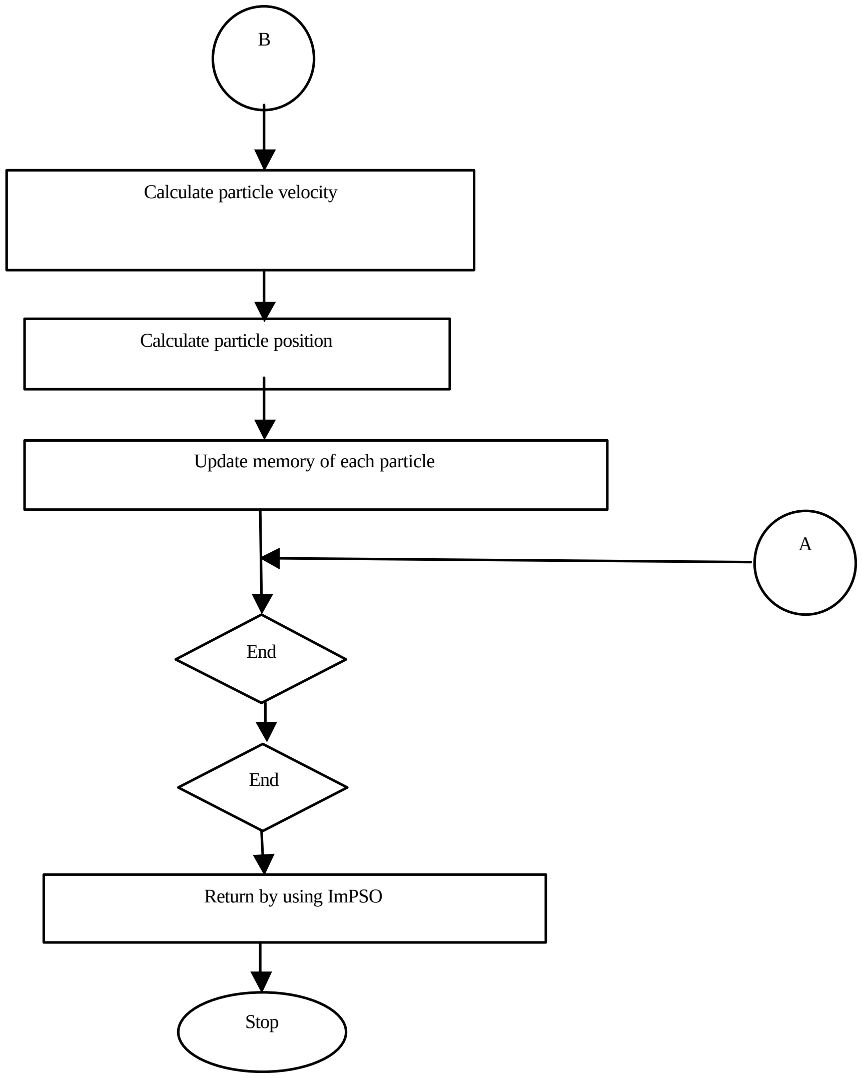
B
Calculate particle velocity
Calculate particle position
Update memory of each particle
A
End
End
Return by using ImPSO
Stop

Fig. 5: Proposed parallel synchronous PSO algorithm

- Configure the master slave environment
- The initial swarm is generated and initialized by the master Stop
- The master sends the initial swarm to all the salves
- The slaves evaluate the initial swarm using the fitness function and select the personal best and global best of the swarm
- Each of the slaves calculates the fitness function and then updates the velocity and position and sends each of its optimal solution to the master after the specified number of iteration is completed
- The master decides the best solution after receiving the results from all the slaves based on the objective function after the maximum number of iterations specified is completed

Proposed Parallel Asynchronous Improved Particle Swarm Optimization (PAIPSO): The weakness in the synchronous parallel PSO algorithm is schedule for the next iteration are analyzed after the current iteration is completed and it can be overcome by considering a parallel asynchronous algorithm. The goal is to have no idle processors as one move from one iteration to the next. To implement a parallel Asynchronous Improved PSO algorithm is to separate the update actions coupled with each sequence and those linked with entire swarm.

PAIPSO is implemented using a maser-slave approach. The master processor holds the queue of feasible particles to be sent to the slave processors. The master performs all decision making processes such as velocity updation, position updation and convergence

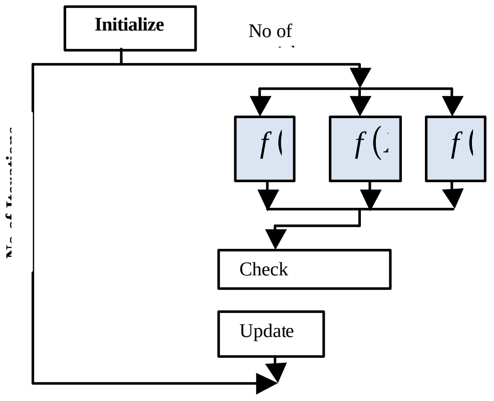

Fig. 6: Parallel synchronous IPSO algorithm

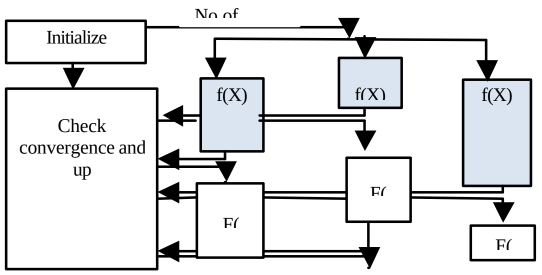

Fig. 7: Parallel asynchronous IPSO algorithm

checks. The slaves perform the function evaluations for the particles sent to them. The tasks performed by the master and slave are as follows:

- Master processor
  - Initialize all optimization parameters and particle positions and velocities
  - Holds a queue of particles for the slave processors to evaluate
  - Updates the particle positions and velocities based on the currently available information
  - Sends the next particle in the queue to an available slave processor
  - Receives cost function values from slave processors
  - Checks convergence
- Slave processor
  - Receives the particle from the master processor
  - Evaluates the objective function of the particle sent to all slaves
  - Sends the cost function value to the master processor

Block diagrams for parallel synchronous and parallel asynchronous PSO algorithms are shown in Fig. 6 and 7. Grey boxes indicate first set of particles evaluated by each algorithm. After completion of initialization step by the master processor, particles are sent to the slave

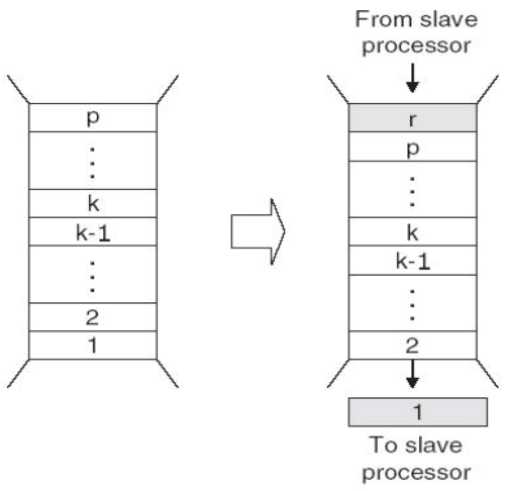

Fig. 8: Block diagram for first-in-first-out centralized task queuewith p particles on a k- processor system

processors to evaluate the objective (analysis) function. The initial step of the optimization is identical to that of the PSPSO algorithm. After initialization, the PAPSO algorithm uses a first-in-first-out centralized task queue to determine the order in which particles are sent to the slave processors.

Whenever, a slave processor completes a function evaluation, it returns the cost function value and corresponding particle number to the master processor which places the particle number at the end of the task queue. Since, the order varies in which particles report their results, randomness in the particle order occurs (Fig. 8).

Once a particle reaches the front of the task queue, the master processor updates its position and velocity and sends it to the next available slave processor. If the number of slave processors is the same as the number of particles, then the next available processor will always be the same processor that handled the particle initially. If the number of slave processors is less than the number of particles, then the next available processor will be whichever processor happens to be free when the particle reaches the front of the task queue. Even with heterogeneity in tasks and/or computational resources, the task queue ensures that each particle performs approximately the same number of function evaluations over the course of an optimization. The proposed parallel synchronous and asynchronous algorithms are experienced with multiprocessor task scheduling. Dynamic (with and without load balancing) task scheduling problems are simulated in MATLAB environment. The results of both the algorithms are compared.

RESULTS AND DISCUSSION

The present section provides the details of the simulation carried out for implementing the proposed parallel approaches PSIPSO and PAIPSO. Benchmark datasets are taken from EricTailard's site for dynamic task scheduling. Two datasets are taken for simulation. Data set 1involves 50 tasks and 20 processors. Data set 2 involves 100 tasks with 20 processors. To demonstrate the effectiveness of the proposed hybrid algorithm. Abut 30 independent trials with different values of random seeds and control parameters. The optimal result is obtained for following parameter settings.

Improved particle SWARM optimization:

- The initial solution is generated randomly
- C_{1g}, C_{1b} and C_2 = 2.2 and 2
- Population size = Twice the number of tasks
- W_{min} - W_{max} = 0.5
- Iteration = 500

The proposed hybrid approaches PSIPSO and PAIPSO aredeveloped using MATLAB R2009 and executed in a PC with Intel core i3processor with 3 GB RAM and 2.13 GHz speed.

Dynamic task scheduling without load balancing: The foremost goal of dynamic task scheduling problem is to reduce the makespan. Hence, to minimize the total execution time the objective function is the same as represented in the Eq. 1-3. The proposed parallel approaches PSIPSO and PAIPSO are tested for the dynamic tasks scheduling problem with datasets specified in the simulation procedure. The results obtained are shown in Table 1.

Parallelization of the Improved Particle Swarm Optimization is proposed to speed up the execution and to provide concurrence. The obtained results, best, average and worst cost for dynamic task scheduling using the parallel algorithms PSIPSO and PAIPSO have been compared with that of the hybrid algorithm IPSO-ACO. The results show that the best cost achieved usingPAIPSO is 2126 for data set 1 and 4196 for data set 2 and is shown in Table 1. When compared to the other methods, the average cost obtained is also better inthe case of the proposed algorithm. The convergence time is drastically reduced for the proposed parallel algorithm PAIPSO compared with hybrid approach IPSO-ACO and is 1.86s for dataset 1 and 2.56s for dataset 2. Thus, it

Table 1: Best cost, Worst cost, average cost and convergence time for IPSO-ACO, PSIPSO and PAIPSO for dynamic task scheduling without load balancing

Methods	IPSO -ACO		Proposed PSIPSO		Proposed PAIPSO	
No. of tasks	50	100	50	100	50	100
Best cost	21310	42260	2126	41960	2126	4196
Worst cost	28530	47930	2792	47510	2786	4703
Average cost	24920	4509.5	2459	4473.5	2456	4506.9
Convergence time in seconds	5.9822	8.1236	3.4862	4.4642	1.8674	2.5691

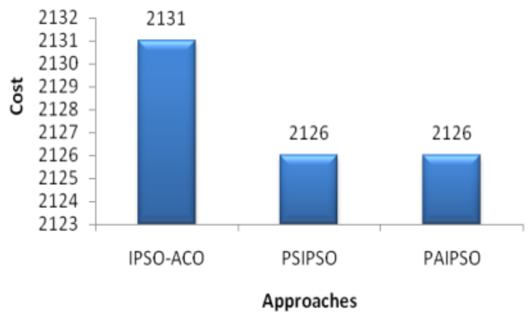

Fig. 9: Best cost for 50 tasks and 20 processors using IPSO-ACO, PSIPSO and PAIPSO

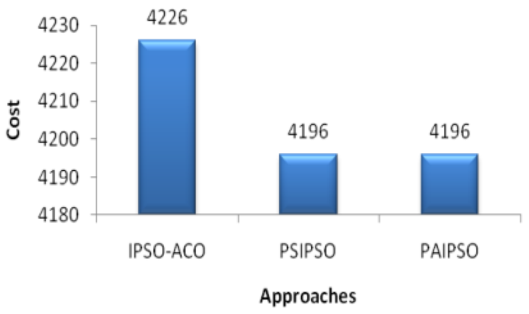

Fig. 10: Best cost for 100 tasks and 20 processors using IPSO-ACO, PSIPSO and PAIPSO

is inferred from the result that the Asynchronous version of IPSO performs better than the Synchronous parallel version of IPSO. PAIPSO is (4-6s) faster than the hybrid approach IPSO-ACO.

Figure 9 and 10 depicts the best cost obtained using the proposed hybrid method Improved Particle Swarm Optimization with Ant Colony Optimization for data set 1 and data set2. The Parallel approach PAIPSO is faster and performs better than with all other algorithms proposed in the present research.

Performance comparison: The performance of the proposed parallel approach PAIPSO iscompared with the previously proposed approaches PSPSO and PAPSO for the same datasets and for multiprocessordynamic task scheduling (Table 2).

Table 2: Performance comparisons of ImPSO-PAPSO with Parallel PSO approaches

Methods	PSPSO		PAPSO		Proposed PAIPSO	
No. of tasks	50	100	50	100	50	100
Best cost	2186	4496	2186	4496	2126	4196
Worst cost	2983	4968	2888	4712	2786	4703
Average cost	2594.5	4768.9	2477.6	4526.3	2456	4506.9
Convergence time in seconds	3.4256	4.4648	1.9619	2.7571	1.8674	2.5691

Table 3: Best cost, worst cost, average cost and convergence time using IPSO-ACO, PSIPSO and PAIPSO for dynamic taskscheduling with balancing

Methods	IPSO-ACO		Proposed PSIPSO		Proposed PAIPSO	
No. of tasks	50	100	50	100	50	100
Best cost	13.0582	22.1531	13.0942	22.1644	13.0942	22.1644
Worst cost	11.4922	20.9624	11.4983	20.9761	12.4386	21.9982
Average cost	12.2752	21.5576	12.2964	21.5703	12.1528	22.0814
Convergence time in seconds	7.56950	10.6314	4.8964	5.7687	2.10320	2.8712

The PSPSO produces the best cost for dataset 1 as 2186, PAPSO produces best cost as 4496 and the proposed PAIPSO produces 2126 as best cost for dataset 1 and 4196 as the best cost for dataset 2. Comparing the convergence time, the proposed PAIPSO is faster than the parallel approaches PSPSO and PAPSO.

This comparison reveals that the proposed parallel approach PAIPSO achieves better results and also that there is a significant difference in the convergence time, when tested for the multiprocessor task scheduling problem.

Dynamic task scheduling with load balancing: In order to improve the processor performance and utilization, load balancing of tasks have to be considered. Therefore the concept of load balancing is dealt, in which the objective function is the same as represented in the Eq. 4-6.

Table 3 shows, the best cost, worst cost, average cost and convergence time for the hybrid algorithms, IPSO-ACO, PSIPSO and PAIPSO for dynamic task scheduling with load balancing. The best cost obtained for dataset 1 using the parallel hybrid approach IPSO-ACO is 13.0582, using the proposed parallel approach PSIPSO is 13.0942 and using the proposed parallel approach PAIPSO is13.0942. For dataset 2, IPSO-ACO produces 22.1531 as the best cost, the proposed PSIPSO produces 22.1644 as the best cost and the proposed PAIPSO produces 22.1644 as the best cost. The convergence time for the hybrid approach is 7.6s, 10.63s for dataset1 and dataset2. The proposed parallel approaches PSIPSO and PAIPSO produces convergence time as 4.89s, 2.1s for dataset 1 and 5.76s, 2.87s for dataset 2, respectively. Thus, the convergence time achieved reveals that the proposed parallel approach PAIPSO produces better results faster than PSIPSO and the hybrid approach IPSO-ACO. The best cost achieved using the proposed parallel approaches PSIPSO and PAIPSO are compared with the hybrid approach IPSO-ACO for data set 1 and data set2. The best cost obtained using the proposed hybrid method Improved Particle Swarm Optimization with Ant Colony Optimization for data set 1 and data set2. The PAIPSO converges faster than the PSIPSO because the idle time of the processors is considerably reduced.

Table 4: Performance comparisons of ImPSO-PAPSO with Parallel PSO approaches

Method	PSPSO		PAPSO		Proposed PAIPSO	
No. of tasks	50	100	50	100	50	100
Best cost	12.982	21.998	12.982	21.998	13.0942	22.1644
Worst cost	10.863	19.429	12.348	21.008	12.4386	21.9982
Average cost	11.789	21.032	12.215	21.816	12.1528	22.0814
Convergence time in seconds	3.9831	5.1956	2.3041	3.1553	2.10320	2.8712

Performance comparison: The performance of the proposed parallel approaches PSIPSO and PAIPSO are compared with the previously proposed parallel methods PSPSO and PAPSO for the same datasets and for multiprocessor dynamic task scheduling (Table 4). The PSPSO produces the best cost for dataset 1 as 12.982, PSPSO produces 12.982 and the proposed parallel approach PAIPSO produces 13.0942. For dataset 2, PSPSO produces best cost as 21.998, PAPSO produces 21.998 and the proposed parallel approach PAIPSO produces 22.1644. The convergence time for the dataset 1 using PSPSO is 3.98s, using PAPSO is 2.3s and using the proposed PAIPSO is 2.1s.

The proposed parallel approach PAIPSO converges very fast whencompared with the other parallel approaches PSPSO and PAPSO. Thus, theresult reveals that the proposed parallel approach PAIPSO outperforms the other parallel approaches PSPSO and PAPSO, because of the inclusion of thebad experience particles in the velocity equation of IPSO which plays a majorrole along with parallelization concept, improves the results to near optimalsolution when applied to the task assignment problem with dynamic tasks.

CONCLUSION

The proposed parallel approaches PSIPSO andPAIPSO to solve dynamic task scheduling with and without load balancing. The proposedparallel approaches locate the optimum solution iteratively from the initialrandomly generated search space. The performance of the proposed PAIPSOis tested using random and bench mark data sets.

The proposed parallel approach PAIPSO is applied to dynamic taskscheduling without load balancing. The results achieved by the proposedparallel approach PAIPSO is compared with parallel approaches proposedearlier namely PSPSO and PAPSO. For dataset1, the best cost achieved byPSPSO is 2186, PAPSO achieves best cost as 2186 and the proposed parallelapproach PAIPSO achieves the best cost as 2126 which is better thanapproaches compared. The proposed parallel approach PAIPSO is 5.5s fasterthan the hybrid heuristic approach IPSO-ACO for dataset 2 and 4.11s fasterthan the result produced by the hybrid approach IPSO-ACO for dataset 1.Theproposed parallel approach have been compared with the other previouslyproposed parallel approaches namely PSPSO and PAPSO and the comparisonresults concludes that the proposed parallel approach is 0.18s faster thanPAPSO and 1.89s faster than PSPSO for dataset2.

The proposed parallel approaches PSIPSO and PAIPSO are appliedto dynamic task scheduling with load balancing. The results achieved by theproposed approaches are compared with parallel approaches proposed earliernamely PSPSO and PAPSO. For dataset 2, the best cost achieved by PSPSOis 21.998, the best cost achieved by PAPSO is 21.998 and the best costachieved by the proposed parallel approach PAIPSO is 22.1644. Theproposed parallel approach PAIPSO converges 1.8799s faster than PSPSO,0.2s faster than PAPSO for dataset 1. For dataset 2, PAIPSO is 2.3244 timesfaster than PSPSO, 0.2841s faster than PAPSO. The proposed parallelapproach yields better results for both static and dynamic task schedulingproblem.

From the results of simulation, it is observed that the proposedparallel approach PAIPSO rapidly increases the performance of the solutionand prevents trapping to a local optimal value. Further, the proposed PAIPSOenhances the probability to find the global best solution, thereby allowingfaster convergence for all the data sets. Thus, the proposed parallel approachPAIPSO produced significant result than GA, standard PSO and hybridapproaches (IPSO-SA, IPSO-AIS and IPSO-ACO).

REFERECNES

Adam, T.L., K.M. Chandy and J.R. Dickson, 1974. A comparison of list schedules for parallel processing systems. Commun. ACM, 17: 685-690.

Ahmad, I. and Y.K. Kwok, 1999. On parallelizing the multiprocessor scheduling problem. Parallel Distrib. Syst. IEEE. Trans., 10: 414-431.

Alba, E. and J.M. Troya, 2001. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Generation Comput. Syst., 17: 451-465.

Allahverdi, A., C.T. Ng, T.C.E. Cheng and Y.M. Kovalyov, 2008. A survey of scheduling problems with setup times or costs. Eur. J. Operat. Res., 187: 985-1032.

Casavant, T.L. and J.G. Kuhl, 1988. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans. Software Eng., 14: 141-154.

Deeba, K. and K. Thanushkodi, 2009. An evolutionary approach for job scheduling in a multiprocessor architecture. Artif. Intell. Syst. Mach. Learn., 1: 122-126.

Eberhart, R.C. and J. Kennedy, 1995. A new optimizer using particle swarm theory. Proceedings of the 6th International Symposium on Micro Machine and Human Science, October 4-6, 1995, Nagoya, Japan, pp: 39-43.

Eberhart, R.C. and Y. Shi, 1998. Comparison Between Genetic Algorithms and Particle Swarm Optimization. In: Evolutionary Programming VII. Porto, V.W., N. Saravanan, D. Waagen and A.E. Eiben (Eds.). Springer Berlin Heidelberg, Berlin, Germany, ISBN: 978-3-540-64891-8, pp: 611-616.

Garey, M.R. and D.S. Johnson, 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, CA., USA., Pages: 338.

Higginson, J.S., R.R. Neptune and F.C. Anderson, 2005. Simulated parallel annealing within a neighborhood for optimization of biomechanical systems. J. Biomech., 38: 1938-1942.

Lee, C.Y., J.J. Hwang Y.C. Chow and F.D. Anger, 1988. Multiprocessor scheduling with interprocessor communication delays. Oper. Res. Lett., 7: 141-147.

Lee, S.Y. and K.G. Lee, 1996. Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. Parallel Distrib. Syst. IEEE. Trans., 7: 993-1008.

Mitten, L.G., 1970. Branch-and-bound methods: General formulation and properties. Oper. Res., 18: 24-34.

Salman, A., I. Ahmad and S. Al-Madani, 2002. Particle swarm optimization for task assignment problem. Microproc. Microsyst., 26: 363-371.

Schutte, J., J. Reinbolt, B. Fregly and R. Haftka1, 2004. Parallel global optimization with the particle swarm algorithm. Int. J. Numerical Methods Eng., 127: 465-474.

Selvakumar, S. and C. Murthy, 1994. Scheduling precedence constrained task graphs with non-negligible intertask communication onto multiprocessors. Parallel Distrib. Syst. IEEE. Trans., 5: 328-336.

Wu, M.Y. and D.D. Gajski, 1990. Hypertool: A programming aid for message-passing systems. IEEE Trans. Parallel Distrib. Syst., 1: 330-343.