CHAPTER 5
PARALLEL GENETIC FLOWER POLLINATION ALGORITHM FOR RSMCDD SCHEDULING

Objective: In the proposed genetic flower pollination algorithm, a population of size $2n$ has become huge for larger job sizes of 500 and 1000 which results in more computation time for scheduling these jobs on a single machine. The ultimate objective of developing parallel hybrid metaheuristics is to speedup the search process by processing huge amount of data to obtain high quality solutions in a reasonable time. This paves way for the development of Parallel Genetic Flower Pollination Algorithm (PGFPA) to successfully tackle job scheduling problems on a single machine against a restrictive common due date.

5.1 INTRODUCTION AND MOTIVATION

Many real-world optimization problems are large and complex. For any real-world optimization problems adopting JIT philosophy in the domain of manufacturing and production systems, applications need more computing power than a sequential algorithm can provide. Restricted single machine common due date scheduling problem is such a large optimization problem in which the jobs that need to be processed on a single machine have the following job sizes namely 10, 20, 50, 100, 200, 500 and 1000. It is quite impractical to solve these larger job sizes on a single computer. With the help of parallel computing, several processes can be used to execute these job sequences simultaneously to reduce computation time. This leads to more computing power for faster execution of job sequences in parallel computing architecture.

While solving RSMCDD scheduling problem using GFP, the
algorithm takes more computation time to generate high quality solutions for larger job sizes 500 and 1000 by maintaining a population of size $2n$. Steps involved in designing GFPA have already been discussed in Chapter 3. As population grows, computation time also grows exponentially. This motivates the concept of parallelization in this dissertation to speedup the search process while improving the quality of the solution. The constraint in parallelizing algorithm is that the data have to be distributed among different processes in the system. Hence the input job sequences are split into small chunks so that each chunk can be processed by different processes in the system characterizing Single Instruction Multiple Data (SIMD) model of computation according to (Flynn M. J. 1966) taxonomy.

As far as single machine common due date job scheduling problems are considered, till now, researchers have not focused on developing parallel metaheuristics for solving this problem. Researchers (Chae Lee and Seok Kim 1995) developed parallel genetic algorithms for single machine earliness-tardiness job scheduling problem with general penalty weights for the job sizes 30, 50, 80 and 100.

5.2 PROPOSED PARALLEL GENETIC FLOWER POLLINATION ALGORITHM

Parallel genetic flower pollination algorithm is a population-based genetic flower pollination algorithm. PGFPA is developed by parallelizing the population into subpopulations / subgroups. Each parallelized subgroup keeps only feasible job sequences. When solving RSMCDD scheduling problem using PGFPA, each solution is coded as a flower with a finite length. Each flower is considered as an individual. A collection of flowers of size $2 \times n$ forms a population where $n$ denotes jobsize. The flowers of the initial population are generated randomly that satisfy the three scheduling optimality properties. New
offsprings are generated from the current population by pollination strategy. The quality of each flower is measured by a fitness function. Algorithm terminates automatically when the flowers in the population are said to have same fitness values.

Parallel Genetic Flower Pollination Algorithm follows (Foster 1995) parallel algorithm design methodology and is implemented on multicore configuration using message passing interface. The key idea in message passing mechanism is to distribute the computations among several processes. The message passing mechanism involves a sender process and one or more receiver processes. The Message Passing Interface (MPI) environment for the proposed parallel algorithm is initialised using MPI_Init command in the MPI.COMM_WORLD communicator. The message passing environment creates number of processes in the MPI.COMM_WORLD communicator and the size of the processes in the communicator is determined using MPI.COMM_WORLD.Size() command. For each process in the communicator, rank has to be generated using MPI.COMM_WORLD.Rank() command to differentiate master process from slave processes. Master process always takes rank 0 whereas slave processes take the rank from 1 to #p-1 where p denotes the number of processes in the communicator. The MPI environment for the proposed parallel hybrid metaheuristics is depicted in Figure 5.1.

5.3 STEPS INVOLVED IN PROPOSED PGFPA

PGFPA makes use of global master-slave parallel programming paradigm model in which fitness values are evaluated in master and all slave processes and the overall model is depicted in Figure 5.2.
Figure 5.1 Communication between five processes in MPI.COMM_WORLD communicator

Figure 5.2 Global master-slave parallel programming model of PGFPA
5.3.1 Initial Population Construction

The steps involved in the construction of initial population are already discussed in Chapter 3. Initial population contains only feasible job sequences of size $2^n$ and master process stores the entire population. Initial population generated for job size 10 with restrictive common due date parameter $h = 0.4$ stored in the master process is depicted in Figure 5.3.

5.3.2 Subpopulation Construction

The parallel hybrid algorithm is parallelized by subpopulation construction method. The entire population stored in master process is partitioned into several subpopulations based on the number of processes in the message passing environment. Number of flowers or job sequences to be assigned to each process is $2^n$.

The scenario for job size = 10 with $h = 0.4$, 20 feasible job sequences are randomly generated. No. of processes involved in the communicator is 4. Master process transmits the 20 flower sequences to all slave processes. The way by which subpopulation constructed is depicted in Figure 5.4.

5.3.3 Communication

PGPFA uses blocking synchronous point-to-point communication mechanism to pass $2^n$ pollen flower sequences between processes using send and receive primitives. Master process transmits $2^n$ sequences to all slaves processes using send() command. Master process collects the results back from the slaves using recv() command. Pollination is carried out in master processes as well as in $(p - 1)$ slave processes.
Figure 5.3 Master process storing initial population
Figure 5.4 Subpopulation construction using partitioning strategy
5.3.4 Evaluation of Fitness Function

During this phase, the flower sequences in the master as well as slave processes undergo fitness evaluation by Equation (1.5). Master process is not idle. In each subpopulation, the fitness evaluation of the pollens are performed in a parallel manner. Data parallelism is achieved as each process performs fitness function evaluation task on different individuals. Each slave process performs fitness evaluation task for the job sequences for every generation. Each population member contributes offsprings to the next generation.

5.3.5 Agglomeration

Each slave process computes fitness values for $2n$ job sequences stored in it and all slave processes send the computed fitness values back to the master process. Master process receives flower sequences of size $2np$ along with their fitness values from which the best $2n$ job sequences alone are stored in the master process for further fitness evaluation for subsequent generations.

5.3.6 Termination

Automatic termination scheme proposed for the parallel hybrid metaheuristics avoids unnecessary computations thereby results in less computation time. This has been already explained in Chapter 4.

5.4 PSEUDOCODE OF PGFPA

Pseudocode of the parallel algorithm to solve restricted single machine common due date scheduling problem is given in Algorithm 5.1.
Algorithm 5.1 Pseudocode of the Master-Slave process of proposed Parallel Genetic Flower Pollination Algorithm

1: procedure MASTER ALGORITHM OF PGFPA

2: input : \( n, P_c, p \) \( \triangleright \) \textbf{n : Job size, }\( P_c: \text{Switch Probability, } p : \text{No. of processes} \)

3: output : \( g_{best} \) \( \triangleright \) \textbf{Best sequence}

4: var noofcores : 4 \( \triangleright \) \textbf{Number of cores}

5: var winsize = \( n/(2p) \) \( \triangleright \) \textbf{To set the size of termination window}

6: for \( (i = 1 \text{ to } 2n \text{ pollen in the population}) \) do

7: Randomly generate pollen flowers without redundancy \( \triangleright \) \textbf{Initial population construction}

8: Compute fitness value for pollen flowers \( \triangleright \) \textbf{Fitness function evaluation}

9: end for

10: while \( (!\text{convergence}()) \) do

11: for \( (i = 0 \text{ to } p - 1) \) do

12: pollenset\([i]\) = GFPA\((n,\text{pollenset})\)

13: Call Slave procedure of GFPA

14: end for

15: Return fittest flower sequences of size \( 2n \) \( \triangleright \) \textbf{Filters from \( 2n^*(p - 1) \) flower sequences}

16: for \( i = 1 \text{ to } 2n \) do

17: pollenset\([i]\) = minimum\( (\text{pollenset}[1],...,\text{pollenset}[i])\)

18: end for

19: end while

20: Return \( g_{best} \)

21: end procedure
procedure SLAVE ALGORITHM OF PGFPA

input : Population of size 2n
output : Return 2n pollen sequences

var threshold : 0.5 ▷ Equal probability

for (i = 1 to 2n pollen in the population) do

MPI.COMM_WORLD.Recv(recvbuf,offset,N_c,MPI.LONG,0,tag) ▷ Receive flower sequences from master process

Evaluate flower sequences ▷ Fitness Evaluation

Generate a random number \( rand \) in the range \([0..1]\)

if \( rand > \) switch probability then ▷ Local pollination

Select two pollens randomly from the flowers and swap

Generate a new offspring

else

Perform global pollination (PMX Crossover) ▷ Global pollination

Generate a float random number \( rand_1 \) in the range \([0..1]\)

if \( (rand_1 > \) threshold) then

Parent 1 ← \( g_{best} \)

Parent 2 ← current pollen

if \( (g_{best} = = \) current pollen) then

Perform local pollination

else

Generate a new offspring

end if

else

Parent 1 ← \( rand_1 \)

Parent 2 ← current pollen
if (Parent 1 == current pollen) then
  Perform local pollination
else
  Generate a new offspring
end if
end if
end if
if (fitness(new offspring) < fitness(current pollen)) then
  current pollen ← offspring
end if
MPI.COMM_WORLD.Send(recvbuf, offset, Nc, MPI.LONG, 0, tag)  

Returns fitness values back to master
end for
Return 2n pollen sequences
end procedure
The parallel genetic flower pollination algorithm is implemented on a multicore configuration system with number of cores to be 4. The input parameters to the master algorithm are number of jobs \((n)\), switch probability \((P_c)\), benchmark problem instances and number of processes \((p)\). Rank of the processes are determined and master process takes the rank 0 whereas slave processes take the rank from 1 to \(p-1\) respectively. The research work considers the number of processes to be 4 and 5 for parallel algorithm.

The task of master process is to construct initial population and to assign pollen sequences to slave processes. It also takes care of termination. The initial population construction method is explained in Section 3.4.2. Master process distributes the pollen sequences to \(p-1\) slave processes and fitness evaluation of the pollens are performed in a parallel fashion in master as well as slave processes. Master process collects \(2n(p-1)\) pollen sequences from slave processes and returns the fittest flower sequences of size \(2n\) to every slave process for pollination. The size of the termination window is kept as \(n/(2p)\). Master as well as every slave process increment generation count for every fitness calculation. The parallel algorithm finally returns the best sequence \(g_{best}\).

5.5 PERFORMANCE METRICS OF PGFPA

Several metrics are available to assess the performance of parallel algorithm namely speedup, throughput, utilization, scalability and efficiency. \(T_s(n)\) represents the time taken by the sequential hybrid metaheuristics to solve a problem of jobsize \(n\). \(T_p(n)\) represents the time taken by the parallel hybrid metaheuristics using \(p\) processes to solve a problem of jobsize \(n\). The speedup achieved by the parallel algorithm is \(S_p(n)\) and is defined in Equation 5.1.
\[ S_p(n) = \frac{T_s(n)}{T_p(n)} \] (5.1)

In this research work, speedup metric alone is used to assess the performance of parallel hybrid metaheuristic algorithm.

5.6 SUMMARY

The proposed parallel hybrid metaheuristics employs master-slave parallel programming paradigm in which the subpopulations are parallelized. Master uses send communication primitive to the slave processes for fitness evaluation and uses receive communication primitive to collect the results back from the slave processes. Less computation time is taken to produce high quality solutions for larger jobs of size 500 and 1000. Speedup metric is used to evaluate the performance of the proposed parallel hybrid metaheuristics. Experimental results of sequential hybrid metaheuristic algorithm while tackling RSMCDD scheduling problems are discussed in detail in Chapter 6.